Computer Vision Lab 1

Image smoothing and simple geometric operations in Matlab

Authors: Emer Rodriguez Formisano and Jorge Alexander

Date: 23th October 2017

1. Creating images of 3 channels (color images)

The RGB images are formed by 3 matrices, commonly called channels. Complete the script "exercise1.m" to implement the following steps:

1.1. Create 3 images in gray scale of size 200x200 as shown in Figure 1.

```
img1 = zeros([200, 200]); img1(1:200,100:200) = 1;
img2 = zeros([200, 200]); img2(100:200,1:200) = 1;
img3 = zeros([200, 200]); img3(1:100,1:100) = 1;
```

1.2. Combine the 3 obtained images to construct the color image shown in Figure 1(right).

```
img4 = cat(3, img1, img2, img3);

set(gcf,'Position',[0 0 1000 200])
subplot(1,4,1), imshow(img1);
subplot(1,4,2), imshow(img2);
subplot(1,4,3), imshow(img3);
subplot(1,4,4), imshow(img4);
```

1.3. Save the color image as 3channels.jpg.

```
imwrite(img4, '3channels.jpg');
```

2. Displaying color images

Complete the script "exercise2.m" to implement the following steps:

2.1. Read the image buffet.jpg

```
buffetRGB = imread('images\buffet.jpg');
```

2. Display the 3 different channels RGB of the image and explain the differences and similarities in pixel values.

```
set(gcf,'Position',[0 0 1000 200])
subplot(1,4,1), imshow(buffetRGB), title('Original');
subplot(1,4,2), imshow(buffetRGB(:,:,1)), title('Red Channel');
subplot(1,4,3), imshow(buffetRGB(:,:,2)), title('Green Channel');
subplot(1,4,4), imshow(buffetRGB(:,:,3)), title('Blue Channel');
```

3. What would happen if we interchange the channels 1 and 2? Make the test, save the result as channel_interchange.jpg and explain what you see.

```
buffet_interchannel = buffetRGB;
buffet_interchannel(:,:,1) = buffetRGB(:,:,2);
buffet_interchannel(:,:,2) = buffetRGB(:,:,1);
subplot(1,2,1), imshow(buffetRGB), title('Original'), ...
subplot(1,2,2), imshow(buffet_interchannel), title('Red <-> Green channels');
```

We observed that the green cabinet becomes red as well as the green vase. The green information has been swaped with the green information. We also observed that the yellow sofa has changed colour slightly as the image has not a perfect balance of red and green channels. The blackground has not changed at all.

```
imwrite(buffet_interchannel, 'channel_interchange.jpg')
```

4. What would happen if we multiply one of the channels by 0? Make the test, save the result as channelx0.jpg and explain what you see.

```
buffet_bluex0 = buffetRGB;
```

```
buffet_bluex0(:,:,3) = buffet_bluex0(:,:,3) * 0;

subplot(1,2,1), imshow(buffetRGB), title('Original');
subplot(1,2,2), imshow(buffet_bluex0), title('Blue channel deleted');
```

Most of the blue information in the picture is contained in the white colour. Thus, when the blue channel is deleted, the white colour changes to yellow.

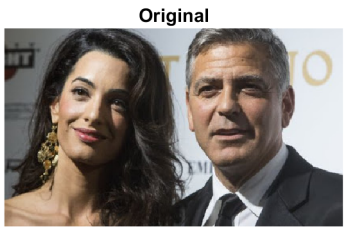
```
imwrite(buffet_bluex0, 'channelx0.jpg')
```

3. Simple Geometric operations

Complete the script "exercise3.m" to implement the following steps. The objective is to create the images in Figure 3. Read the image clooney.jpg and change the place of both figures so that George Clooney stands on the left. The function should have 2 parameters: the original image and a number being the column number of the cut. It should return a new image with the persons interchanged. You can use imcrop to find the optimal cut column.

Using the function mirror_image defined at the end of the report (LiveScript requirement):

```
clooney = imread('images/clooney.jpg');
cut_column = 225;
subplot(1,2,1), imshow(clooney), title('Original');
subplot(1,2,2), imshow(mirror_image(clooney, cut_column)), title('Mirrored Image');
```



4 Managing different size and filters

Complete the script "exercise4.m" to implement the following steps:

1. Read the image corals.jpg from the folder images.

```
CORALS = imread('images/corals.jpg');
```

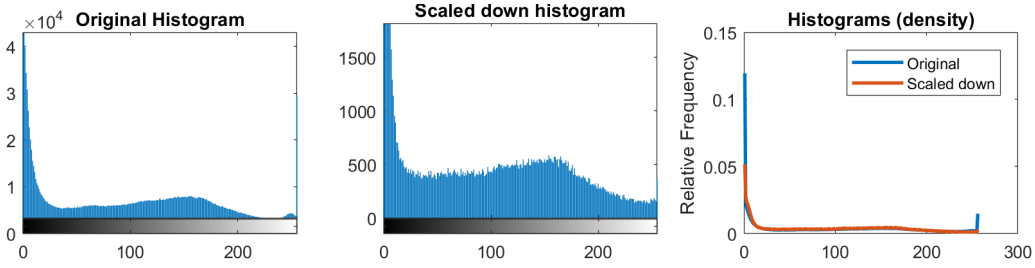
2. Show how details of the image disappear when rescale (make smaller) the size of the image. Help: use imresize with a scale of 0.25.

```
reCorals = imresize(CORALS, 0.25);
subplot(1,2,1), imshow(CORALS), title("Original");
subplot(1,2,2), imshow(reCorals), title("Scaled down");
```

3. Does the histogram change of both images (the original and the rescaled one)?

The histogram shows the frequency of pixels for each level of intensity. The scaled down image has less pixels, so the counts (absolute freq.) will be lower. We should compare the shape or density (relative freq.) instead.

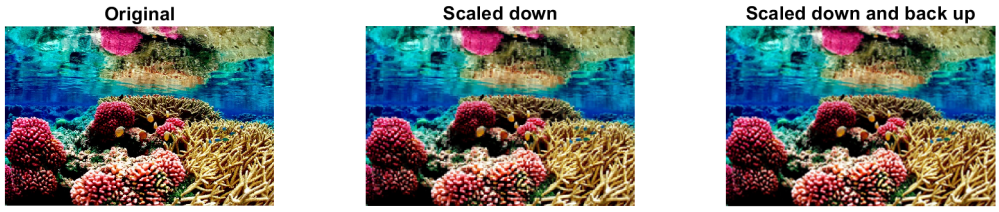
```
set(gcf,'Position',[0 0 1000 200])
coralsN = imhist(CORALS);
reCoralsN = imhist(reCorals);
subplot(1,3,1), imhist(CORALS), title("Original Histogram"), ...
subplot(1,3,2), imhist(reCorals), title("Scaled down histogram"), ...
subplot(1,3,3), plot([coralsN/sum(coralsN) reCoralsN/sum(reCoralsN)], 'LineWidth',2);
legend('Original','Scaled down'), ylabel('Relative Frequency'), title('Histograms (density)')
```



Both lines are very close to each other which means that scaling the image down do not affect the intensity distribution.

4. Return back the smaller image to the original size. Compare to the original one and make a comment on that.

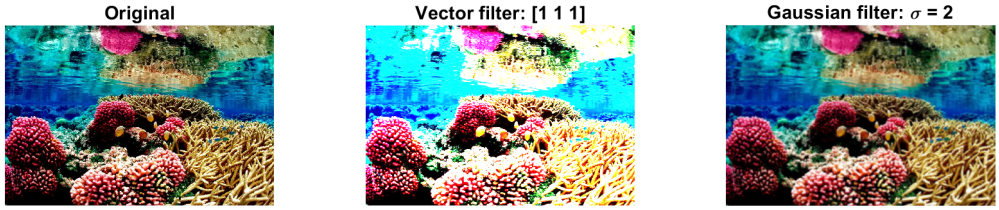
```
reCorals2 = imresize(reCorals, 4);
subplot(1,3,1), imshow(CORALS), title("Original"), ...
subplot(1,3,2), imshow(reCorals), title("Scaled down"), ...
subplot(1,3,3), imshow(reCorals2), title("Scaled down and back up");
```



Here we see that even though the image is now the same size as the original again, the definition (information) is still lost.

5. As an alternative of removing image details, apply different smoothing filters (user-defined mask as a vector e.g. [1 1 1] vs. Gaussian filter).

```
VECTOR_FILTER = ones(1,3);
filterCorals = imfilter(CORALS, VECTOR_FILTER);
gaussFilterCorals = imgaussfilt(CORALS,2);
subplot(1,3,1), imshow(CORALS), title("Original"), ...
subplot(1,3,2), imshow(filterCorals), title("Vector filter: [1 1 1]"), ...
subplot(1,3,3), imshow(gaussFilterCorals), title("Gaussian filter: \sigma = 2");
```



Here we see that VECTOR_FILTER has made the image brighter and has lost definition (for example the lines can no longer be seen on the fish). As the vector is not normalised, the mask modifies the pixels by adding the value of the left and right pixels for each channel. Thus, the value for each pixel is increased making a whiten and slighlty blur image. The gaussian filter has not made the image brighter as the weights are normalised, but has only blurred the image so we can neither see the lines on the fish.

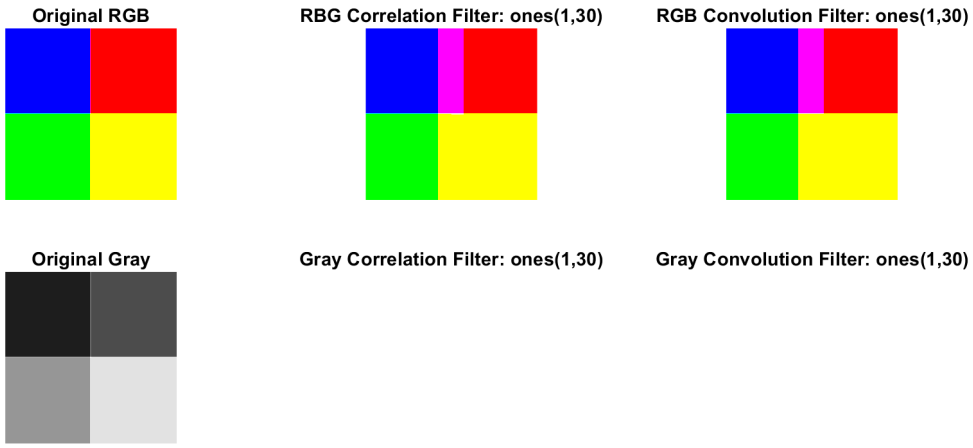
Discuss how the size of the filter affects the final outcome.

```
LARGER_VECTOR_FILTER = ones(1,30);
filterCorals = imfilter(CORALS, LARGER_VECTOR_FILTER);
gaussFilterCorals = imgaussfilt(CORALS,9);
subplot(1,3,1), imshow(CORALS), title("Original"), ...
subplot(1,3,2), imshow(filterCorals), title("Vector filter: ones(1,30)"), ...
subplot(1,3,3), imshow(gaussFilterCorals), title("Gaussian filter: \sigma = 9");
```

Increasing the size will make the in the mask vector, will make it whiter and more blurred by using only pixels in the horizontal axis. In other hand, increasing the value of sigma, the wider the gaussian distribution will be, which takes further pixels (more information). As a result, the image will be blurrer with better image choerence as the bell shape of the weights reflects the idea that the pixels sourranding to the central pixel are more relavant than further away pixels.

6. Can you apply the filter on the RGB image? What type should be the image before convolving and why? What dimensions should be the mask?

```
set(gcf,'Position',[0 0 1000 400])
% JPG has lost definition, better to create it from scratch:
img1 = zeros([200, 200]); img1(1:200,100:200) = 1;
img2 = zeros([200, 200]); img2(100:200,1:200) = 1;
img3 = zeros([200, 200]); img3(1:100,1:100) = 1;
rgb = cat(3, img1, img2, img3);
gray = rgb2gray(rgb);

filterRgbCorr = imfilter(rgb, LARGER_VECTOR_FILTER, "corr");
filterRgbConv = imfilter(rgb, LARGER_VECTOR_FILTER, "conv");
filterGrayCorr = imfilter(gray, LARGER_VECTOR_FILTER, "corr");
filterGrayConv = imfilter(gray, LARGER_VECTOR_FILTER, "conv");

subplot(2,3,1), imshow(rgb), title("Original RGB"), ...
subplot(2,3,2), imshow(filterRgbCorr), title("RBG Correlation Filter: ones(1,30)"), ...
subplot(2,3,3), imshow(filterRgbConv), title("RGB Convolution Filter: ones(1,30)"), ...
subplot(2,3,4), imshow(gray), title("Original Gray"), ...
subplot(2,3,5), imshow(filterGrayCorr), title("Gray Correlation Filter: ones(1,30)"), ...
subplot(2,3,6), imshow(filterGrayConv), title("Gray Convolution Filter: ones(1,30)");
```

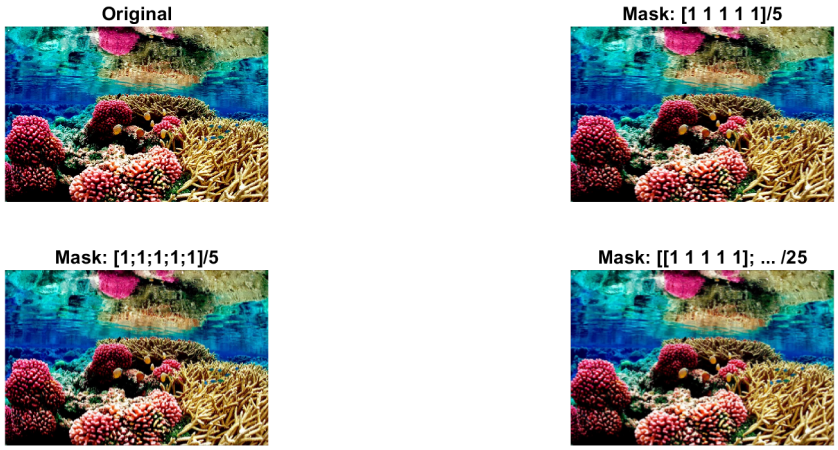


You can filter the RGB image, the mask on gray images is not . The mask should have the same dimensions as the image, one for each color

7. What is the difference using the following three masks: [1 1 1 1 1], [1;1;1;1;1] and [[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]]? Do we need to normalize the mask (divide by the sum of all its numbers)?

- [1 1 1 1 1]: Convolving with this will make each element in the matrix the sum of itself, and up to 2 elements before it and 2 elements after it horizontally.
- [1;1;1;1;1]: Convolving with this will make each element in the matrix the sum of tself,and up to 2 elements above it and 2 elements below it vertically.
- [[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]]: Convolving with this mask will make each element in the matrix the s the sum of all the other values covered by the mask (including itself) when the mask is placed with the element at the centre of it

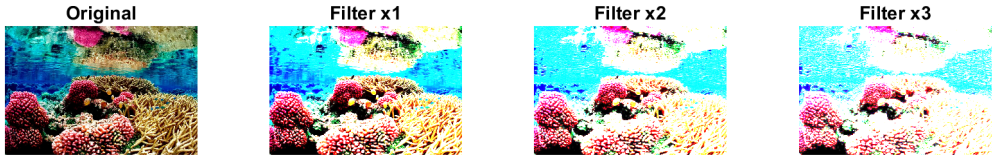
```
set(gcf,'Position',[0 0 1000 400])
maskA = [1 1 1 1 1]/5;
maskB = [1;1;1;1;1]/5;
maskC = [[1 1 1 1 1]; [1 1 1 1 1]; [1 1 1 1 1]; [1 1 1 1 1]; [1 1 1 1 1]]/25;
subplot(2,2,1), imshow(CORALS), title("Original"), ...
subplot(2,2,2), imshow(imfilter(CORALS, maskA)), title("Mask: [1 1 1 1 1]/5"), ...
subplot(2,2,3), imshow(imfilter(CORALS, maskB)), title("Mask: [1;1;1;1;1]/5"), ...
subplot(2,2,4), imshow(imfilter(CORALS, maskC)), title("Mask: [[1 1 1 1 1]; ... /25");
```



Well the some of the weights affects the overall intensity of the ouput image, so we can normalise the weights of the mask so that they sum to one so that we dont affect the intensity of the image as much

8. Apply the filter several times in order to observe the effects.

```
set(gcf,'Position',[0 0 1000 200])
onceFiltered = imfilter(CORALS, VECTOR_FILTER);
twiceFiltered = imfilter(onceFiltered, VECTOR_FILTER);
thriceFiltered = imfilter(twiceFiltered, VECTOR_FILTER);
subplot(1,4,1), imshow(CORALS), title("Original"), ...
subplot(1,4,2), imshow(onceFiltered), title("Filter x1"), ...
subplot(1,4,3), imshow(twiceFiltered), title("Filter x2"), ...
subplot(1,4,4), imshow(thriceFiltered), title("Filter x3");
```

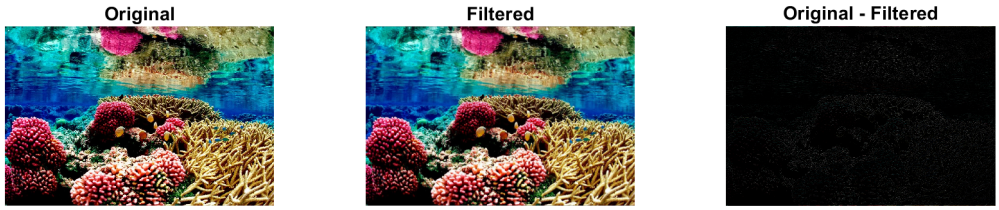


It gets whiter due to applying the effect describe in 4.5 several times.

9. You can subtract the original and smoothed images in order to illustrate the difference between them. Use subplot in order to show the original smooth and the difference image in the same figure.

```
set(gcf,'Position',[0 0 1000 200])
USER_DEFINED_FILTER = [[3 5 3];[1 6 1];[2 7 3]]/31;
subplot(1,3,1), imshow(CORALS), title("Original"), ...
subplot(1,3,2), imshow(imfilter(CORALS, USER_DEFINED_FILTER)), title("Filtered"), ...
subplot(1,3,3), imshow(CORALS - imfilter(CORALS, USER_DEFINED_FILTER)), ...
```

```
title("Original - Filtered");
```



The resulting image (Original - Filtered) is the difference between them. The image is dark because the differences are small, hence most of the values are close to 0 (black) a part from whiter pixels.

5. Image binarization

Complete the script "exercise5.m" to implement the following steps: The binarization $B_I(x, y)$ of an image $I(x, y)$ from a threshold (Th) consists in turning the image into a binary image (of 0s and 1s) that will depend on whether the level of the pixel intensity of the original image is above or below a threshold.

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

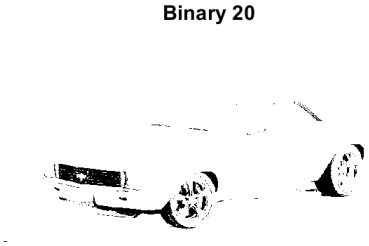
Given the image car_gray.jpg, create the function thresholdImg() which implements the following points:

```
CAR = imread('images/car_gray.jpg');
```

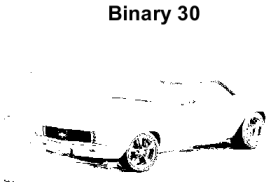
1. Create the binary version of the original image by a threshold value of 20. What does it happen if we use different threshold values (30, 150, 255)? Why?

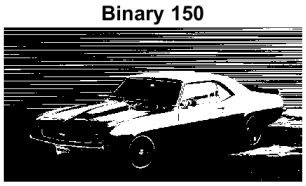
Implementation code in the Defined Function section at the end of the report.

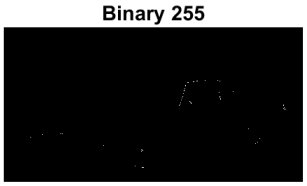
```
% binary with threshold 20
bin20 = thresholdImg(CAR, 20);
% threshold of 30, 150, 255
bin30 = thresholdImg(CAR, 30);
bin150 = thresholdImg(CAR, 150);
bin255 = thresholdImg(CAR, 255);

% plot the different images
set(gcf,'Position',[0 0 1000 200])
subplot(1,2,1), imshow(CAR), title("Original"), ...
subplot(1,2,2), imshow(bin20*255), title("Binary 20");
```




```
subplot(1,3,1), imshow(bin30*255), title("Binary 30"), ...
subplot(1,3,2), imshow(bin150*255), title("Binary 150"), ...
subplot(1,3,3), imshow(bin255*255), title("Binary 255");
```







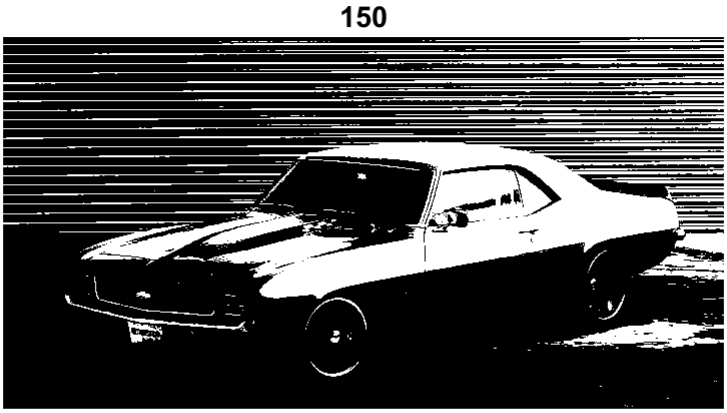
We observe that for a low threshold the image is very bright and hard to see detail. This is because a high portion of the image is being converted to ones.

While that for a medium threshold, there is a good contrast to see the edges of the car and detail. This is because there is a balanced proportion of elements being binarized to zero with those being binarized to 1.

For a high threshold, the opposite effect of a low threshold is observed. The image is very dark and it is hard to see detail of the image. This is because a large portion of the inormation of the image is being binarized to 0.

2. Visualize and save the image of threshold 150 as car_binary.jpg.

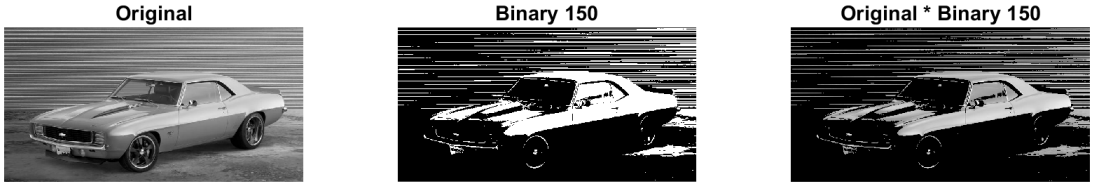
```
subplot(1,1,1), imshow(bin150*255), title("150");
```




```
imwrite(bin150*255, 'car_binary.jpg');
```

3. What will happen if you multiply the original image by the binary image?

```
set(gcf,'Position',[0 0 1000 200]);
subplot(1,3,1), imshow(CAR), title("Original"), ...
subplot(1,3,2), imshow(bin150*255), title("Binary 150"), ...
subplot(1,3,3), imshow(CAR .* bin150), title("Original * Binary 150");
```

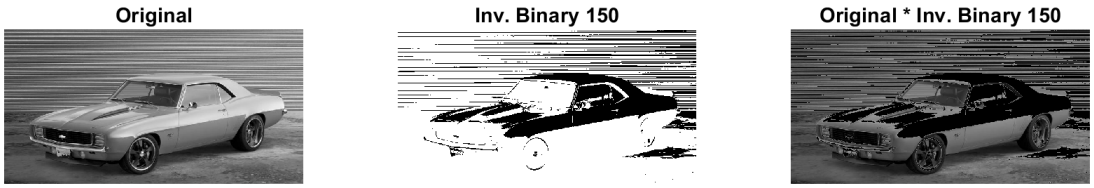


The resulting image contains the original intensity in the areas where the binary image has the value of 1 while it is black in the areas of with value 0.

4. What will happen if you multiply the original image with the inverted binary image?

We consider the inverted image as the complementary image.

```
subplot(1,3,1), imshow(CAR), title("Original"), ...
subplot(1,3,2), imshow(not(bin150)), title("Inv. Binary 150"), ...
subplot(1,3,3), imshow(CAR .* uint8(not(bin150))), ...
title("Original * Inv. Binary 150");
```



Inverting the binary image makes the resulting image to be black where it was previously white. What was previously black, now it has the intensity from the original image. It is the same effect of the previous question but inverted.

6. Treating color images

Complete the script "exercise6.m" to implement the following steps: Given the images hand.jpg (Figure 3(left)) and mapfre.jpg (Figure 3(middle)), create the function fuseImg(), which implements the following points:

The function fuseImg executed as follows, runs the steps detailed in the next subsections.

```
fuseImg()
```

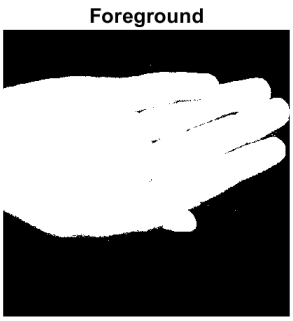
The implementation of the function can be found in the Defined Function section at the end of the report.

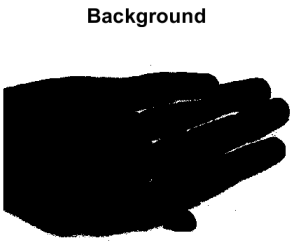
1. Open hand.jpg and convert it in gray scale image.

```
HAND = imread('images/hand.jpg');
MAPFRE = imread('images/mapfre.jpg');
grayscaleHand = rgb2gray(HAND);
```

2. Perform a binarization to obtain a binary image of 2 regions: the hand (called foreground) and the rest (called background). Create the inverse binary image changing the areas of foreground and background.

```
th = 20;
foregroundHand = grayscaleHand >= th;
backgroundHand = not(foregroundHand);

set(gcf,'Position',[0 0 1000 600]), ...
subplot(2,2,1), imshow(HAND), title("Original Hand"), ...
subplot(2,2,2), imshow(MAPFRE), title("Original Mapfre"), ...
subplot(2,2,3), imshow(foregroundHand), title("Foreground"), ...
subplot(2,2,4), imshow(backgroundHand), title("Background");
```





3. Use the binary matrices created in (2) to merge the images hand and mapfre (Fig. 3(right))

```
background = MAPFRE.*uint8(backgroundHand);
foreground = HAND.*uint8(foregroundHand);
handMapfre = background + foreground;
subplot(1,1,1), imshow(handMapfre), title("Hand + Mafre");
```

Hand + Mafre

4. Save the image as hand_mapfre.jpg.

```
imwrite(handMapfre, 'hand_mapfre.jpg');
```

Defined Functions

```
function fuseImg()
%fuseImg Function that executes steps 1 to 4 of Exercise 6

% 1 Read Input
HAND = imread('images/hand.jpg');
MAPFRE = imread('images/mapfre.jpg');
grayscaleHand = rgb2gray(HAND);

% 2 Binarization
th = 20;
foregroundHand = grayscaleHand >= th;
backgroundHand = not(foregroundHand);

% 3 Merge
background = MAPFRE.*uint8(backgroundHand);
foreground = HAND.*uint8(foregroundHand);
handMapfre = background + foreground;

% 4 Save output
imwrite(handMapfre, 'hand_mapfre.jpg');

end

function binImg = thresholdImg(img, thresh)
```

```
%THRESHOLDIMG Convert Gray Image into Binary Image
%    Transform the image (img) into a binary image (of 0s and 1s) that will
%    depend on  whether the level of the pixel intensity of the original image
%    is above or below a threshold (tresh).
%    The result type is uint8 for better compatibility with gray images.
    binImg = uint8(img >= thresh);
end

function mirror = mirror_image(image, cut_col)
%MIRROR Take an image and return its mirror, reflected at the specified cut column
    width = length(image(1, :, 1));
    left = image(:,1:cut_col,:);
    right = image(:, cut_col:width, :);
    mirror = [right, left];
end
```