Advanced Topics in Computer Vision - Lab 4

Leonardo Palacios Fidalgo Ce Xu Zheng

June 5, 2022

Frequency domain filtering

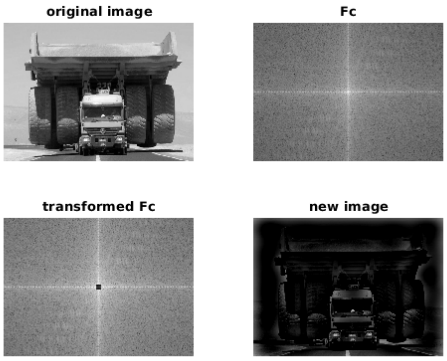
In this Lab session we are going to do several exercises studding the properties of the Fourier Transformation of an image.

Exercise 4-1. Modify manually the transform spectrum and show the results of the inverse transform. Is it any evident change in the original image?

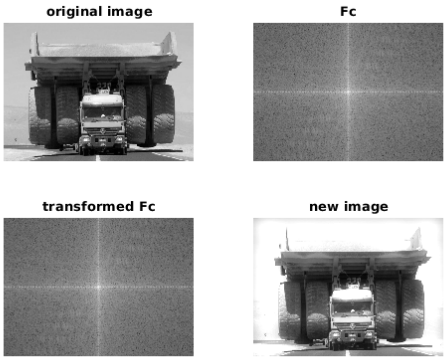
To see the properties of the image in the Fourier space, we can perform some changes to the image there and turn it back to the gray scale to see the effects of the modifications. Firstly we know that the Fourier space is periodic, so we will represent the Fourier space with only the modulus of the complex numbers and the lower frequencies in the center of the image. To have more representative image, we are going to represent the values in a logarithmic scale.

First we are going to modify the lower frequencies by modifying a square region around the center of the shifted Fourier space. As we can see in the Fig.1, by reducing the modulus of the terms with lower frequency, we erase the offset of the brightness of the whole image, making it dimmer, but we can still observe the bright details with a higher frequency.

In the other hand, if we make it a little bit greater (by multiplying the modulus by 1.4, we can clearly see that the brightness saturates the resulting image.



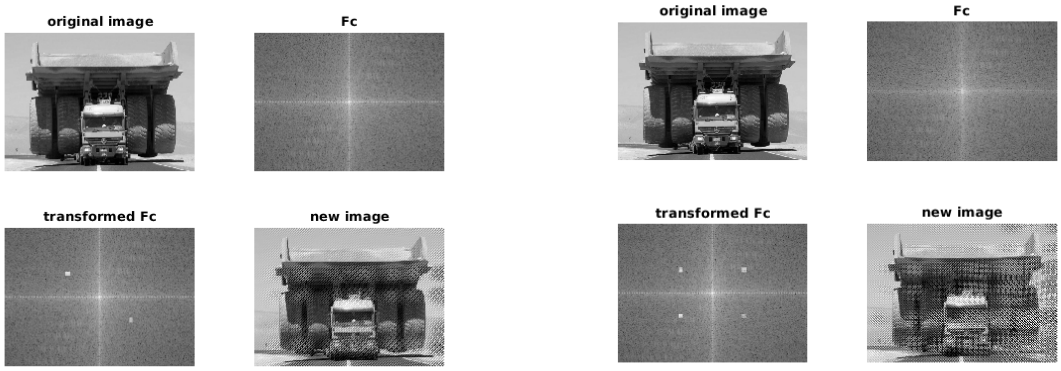
(a) Reducing the modulus of the central square.



(b) Increasing the modulus of the central square.

Figure 1: Central square modifications.

After that, we can also look for the changes that adding a great frequential component can make by adding a high modulus square (preserving the phase) in random sections of the frequency domain.



(a) Two opposed squares. (b) Four squares at the same distance to the center.

Figure 2: Central square modifications.

As we can see in the Fig.2, the addition of the squares will create artifacts around a certain frequency and in a particular direction. The distance to the center will determine the frequency of the artifacts, and if we add two similar squares symmetrical the center, the effect will not be noticeable due to the periodic nature of the Frequency domain. More over, if we also add the symmetrical components perpendicular to the previous two, the direction of the new artifacts will be perpendicular to the previous ones and the resulting total artifacts will be of squared shape.

So finally, we can conclude that, although we can not predict all the properties of the image in the Fourier space, the significative changes on it will affect visibly in the gray scale representation of it.

Exercise 4-2. Check the effects in the inverse transform whether fftshift is executed over the spectrum or not.

Figure 3: fftshift effect.

As we can see in the Fig.3, if we don't perform any shifting in the Fourier space, we will be able to recover the original image without any noticeable loss of quality. But if we shift the Fourier representation of the image with the *fftshift* function and transform back the image, we have inverted the values of the frequency components of the image. The most dominant one will be the one with the higher frequency (pixel by pixel), and the offset of the image will have a very low value, so the image will tend to be dimmer than the original one.

Surprisingly, we are still able to appreciate the overall shape of the image but without a high fidelity on the details.

Exercise 4-3. Using the dftuv function:

1. find a meshgrid of 9x5 elements,
2. compute the distance squared from every point in this rectangle of size 9x5 to the origin of the rectangle,

3. obtain the same distances if the origin of the rectangle is moved to its center, which will be the coordinates of the center?

We have seen that the *dftuv* function computes the matrix of horizontal and vertical distances between each pixel and the first one in a toroidal way. To invoke it you only have to provide the dimensions of the matrix that you need to compute, normally the sizes of an image.

For the first point, we only have to call the function with the sizes of the desired meshgrid *[U,V] = dftuv(9,5);*.

For the second point, it is enough to square and add up the two resulting matrices element-wise to obtain the distances matrix. We can see the output in the Fig.4a, with the zero in the first pixel and the squared distance to it or each other pixel.

For the third point it is enough to use the the *fftshift* function to change the center of the distances matrix to the middle of the matrix as we can see in the Fig.4b. Now the coordinates of the center will be the $(5, 3)$ in matlab indices notation.

```
D =

     0     1     4     4     1
     1     2     5     5     2
     4     5     8     8     5
     9    10    13    13    10
    16    17    20    20    17
    16    17    20    20    17
     9    10    13    13    10
     4     5     8     8     5
     1     2     5     5     2
```

(a) Distance matrix.

```
D2 =

    20    17    16    17    20
    13    10     9    10    13
     8     5     4     5     8
     5     2     1     2     5
     4     1     0     1     4
     5     2     1     2     5
     8     5     4     5     8
    13    10     9    10    13
    20    17    16    17    20
```

(b) Distance matrix shifted.

Exercise 4-4. Implement a function that allows to compute a lowpass filter with the following specifications:

To do so, we used the previously used *dftuv* function to compute the squared distances matrix to the first pixel as it will be useful to detect the distance to the lower frequency pixel in the Fourier space. After obtaining that, we only need to implement one by one the three types of filter that we are going to accept. In the Listing 1, we can see the implementation of the filter.

Listing 1: Low pass filter function

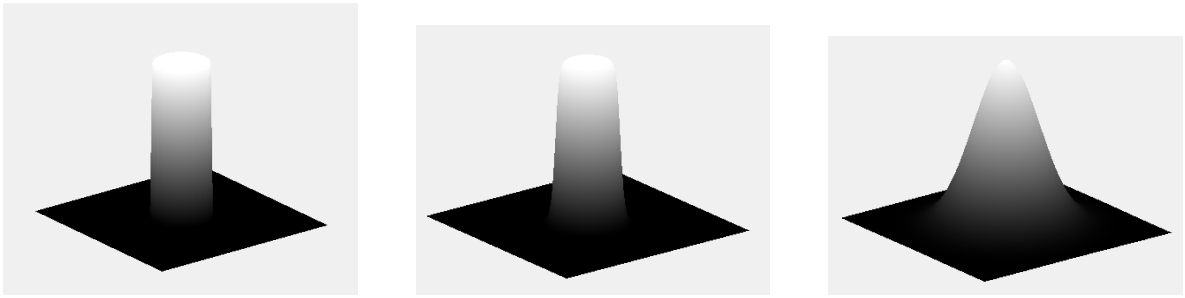
```
function H = lpfilter(tipus, M, N, D0, n)
% LPFILTER computes frequency domain lowpass filters
% H = LPFILTER(TYPE, M,N,D0,N) creates the transfer
% function of a lowpass filter, H, of the specified TYPE and
% size (M-by-N). To view the filter as an image or mesh
% plot, it should be centered using H=fftshift(H).
%
% Valid values for TYPE, DO and n are:
%
% 'ideal' ideal lowpass filter with cutoff frequency D0. n
% need not be supplied. D0 must be positive.
%
% 'btw' Butterworth lowpass filter of order n, and cutoff
% D0. The default value for n is 1.0. D0 must be
% positive.
%
% 'gauss' gaussian lowpass filter with cutoff (standard
% deviation) D0. n need not be supplied. D0 must
% be positive.

[U,V] = dftuv(M,N);

```

```
D = sqrt(U.^2 + V.^2);

switch tipus
    case 'ideal'
        H = double(D ≤ D0);
    case 'btw'
        if( nargin == 4)
            n = 1;
        end
        H = 1./( 1 + (D/D0).^(2*n));
    case 'gauss'
        H = exp(-D.^2./(2*D0^2));
    otherwise
        print("wrong filter type")
end
```



(a) Ideal filter. (b) Btw filter. (c) Gauss filter.

Figure 5: Low pass filters

In the Fig.5, we can see the resulting filters for each one of the filter types in an image of 128 by 128 and a cutting distance of $D0 = 18$. For the BTW filter, we have chosed a fourth degree one.

Exercise 4-5. Filter with a lowpass filter the image 'moon.tif' according the following rules and show the results:

1. with an ideal filter
2. with a Butterworth filter order 2 and 4.
3. with a gaussian filter.

Once the lowpass filter in the Fourier space can be easily defined with previously defined *lpfilter* function, and as we have it normalized between 0 and 1, we can perform the filtering just by multiplying the filter with the transformed image element-wise. After that, we only need to perform the inverse fft to the filtered image.

Figure 6: Filtered moon

In the Fig.6, we can see the results of applying the different ow pass filters to the moon image with a cutting distance of $D0 = 60$. We can appreciate that the Ideal filter generates some rounded artifacts in the moon. And as we increase the degree of the BTW filter, we can also observe similar artifacts as it will be more closer to the ideal filter.

Exercise 4-6. Implement a function (hpfilter) that allows to find the transfer function of a highpass filter, with exactly the same parameters that the function lpfilter.

Once we have the lowpass filter defined, we can easily define the high pass filter taking advantage of the previous one. We just have to know that the high pass filter of any type is equivalent to the one minus the equivalent lowpass filter. With this in mind the implementation is pretty simple as we can see in List.2. The only possible complication is to consider the case where the the degree of the BTW filter n must sometimes taken into account and sometimes not.

Listing 2: High pass filter function

```
function H = hpfilter(tipus, M, N, D0, n)
% HPFILTER computes frequency domain highpass filters
% H = HPFILTER(TYPE, M,N,D0,N) creates the transfer
% function of a highpass filter, H, of the specified TYPE and
% size (M-by-N). To view the filter as an image or mesh
% plot, it should be centered using H=fftshift(H).
%
% Valid values for TYPE, DO and n are:
%
% 'ideal' ideal highpass filter with cutoff frequency D0. n
% need not be supplied. D0 must be positive.
%
% 'btw' Butterworth highpass filter of order n, and cutoff
% D0. The default value for n is 1.0. D0 must be
```

```
% positive.
%
% 'gauss' gaussian highpass filter with cutoff (standard
% deviation) D0. n need not be supplied. D0 must
% be positive.
if( nargin == 4)
    n = 1;
end
h = lpfilter(tipus, M, N, D0, n);
H = 1-h;
```

Exercise 4-7. Filter the 'moon.tif' image with a high-frequency emphasis filter with a = 0.5 and b=2.0. Try to change the offset and the multiplier in order to improve the result.

In this exercise we need to preserve some information of the low frequencies to have a good representation of the original image. So instead of filtering out the low frequencies, we are going to reduce them in some degree (a) and emphasize the high frequencies with some other factor (b) that multiplies the high frequency filter. The resulting filter can be applied by multiplying element-wise the filter and the Fourier representation of the image as we have done before.

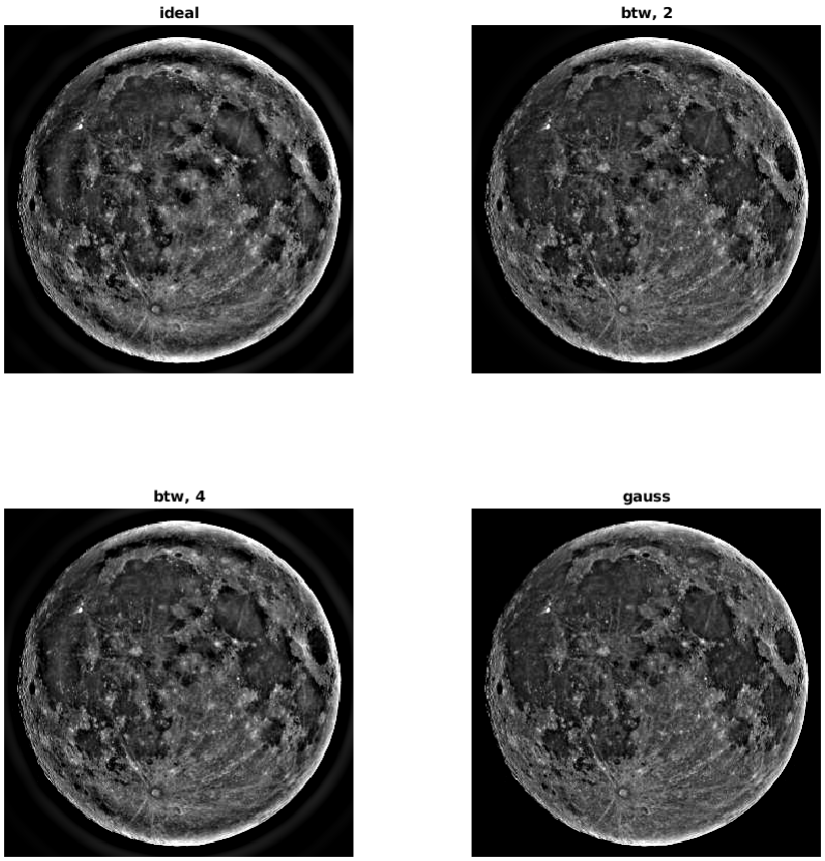


Figure 7: High frequency emphasis with $a = 0.5$, $b = 2.0$, D0 = 20.

As we can see in the Fig.7, the suggested values of the parameters does not result in a filter that

preserves the image and emphasize the high frequency details. As the image is pretty dim, we can assume that the weight of the low frequencies is too low, and the hard edges that we can see may suggest a too hih b value.

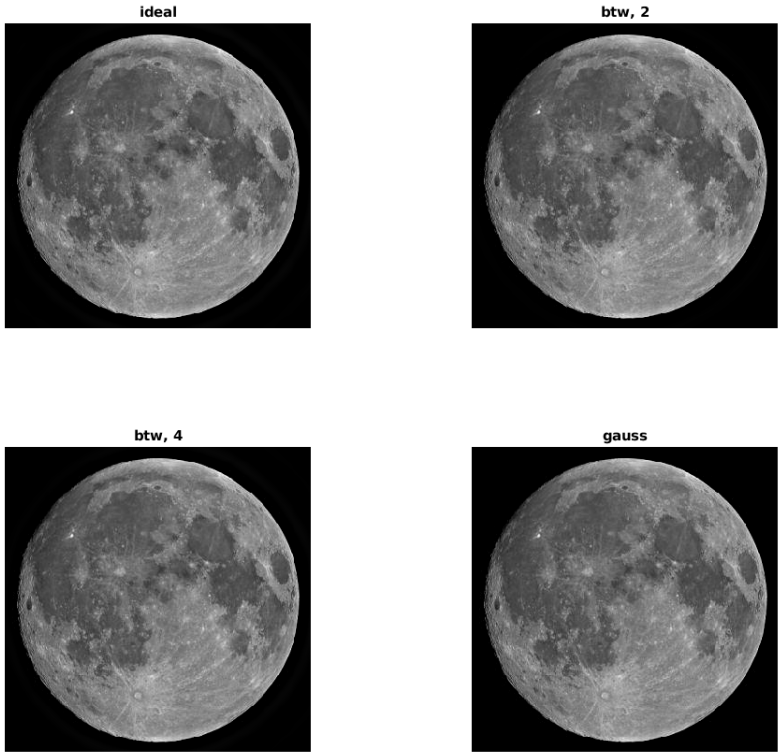


Figure 8: High frequency emphasis with $a = 0.8$, $b = 0.4$, D0 $= 20$.

In the Fig.8, we have chosen an $a = 0.8$ and a $b = 0.4$ to preserve most of the low frequency data, and add the high frequency information just a little bit over the unity to sharpen the image. With these values, we can consider that we have performed a good high-frequency emphasis filter.

Notch Filters to remove noise

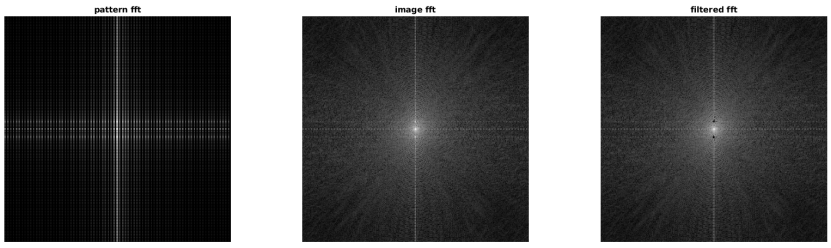
In this section we are going define a function that is able to extract strong periodic patterns that pollute the image. To do so, we are going to use the ideal high pass filter to focus in the values of the high frequencies avoiding the high peaks of the low frequencies. If we are sure that there is some strong periodic pattern with a higher frequency than the high pass filter cut off distance (that we can chose). Then it will be high peak in that spot and the surrounding pixels with similar values. I all the noise patterns have similar intensities, we can erase all of them in one step by erasing all the frequencies with a modulus higher than a certain fraction of this maximum value. To have a better representation of it, we are going to use the threshold in the logarithmic scale instead of the original scale. In the List.3, we can see the implementation of the filter with some default values of the cut off distance an the filtering threshold.

Listing 3: Periodic noise filter

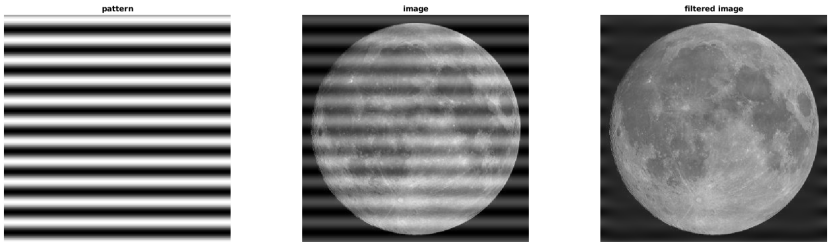
```
1 function [clean_im, source_fft, clean_fft] = filter_pattern_noise(source_im, D0, th)
```

```
2
3  if(nargin == 1)
4      D0 = 20;
5      th = 0.7;
6  else if(nargin == 2)
7      th = 0.7;
8  end
9  PQ = paddedsize(size(source_im));
10 source_fft = fft2(source_im, PQ(1), PQ(2));
11
12 H = hpfilter("ideal", PQ(1),PQ(2),D0);
13 noise_fft = H.*source_fft;
14
15 Fc_norm = abs(noise_fft);
16 S_patt = log(1+Fc_norm)/log( max(Fc_norm, [], 'all') );
17 m = max(S_patt, [], 'all');
18
19 clean_fft = source_fft;
20 clean_fft(S_patt>th*m) = 0;
21
22
23 clean_im = ifft2(clean_fft);
24 clean_im = clean_im(1:size(source_im, 1), 1:size(source_im, 2));
25 end
```

To evaluate the algorithm, we are going add some regular noise to the "moon.tif" image and try to erase without previous knowledge of the regular noise. The final image will be composed by a weighted addition of the moon image with a 0.7 factor and the patter with a 0.3 factor.



(a) Fourier space of the images.

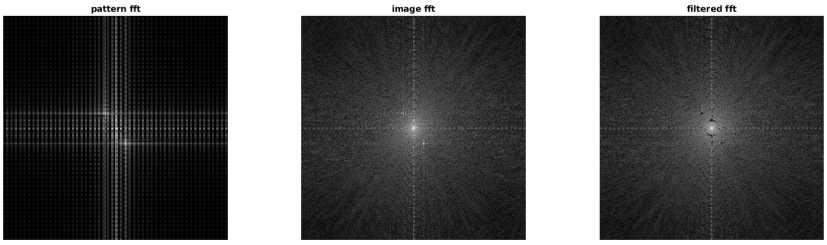


(b) Gray scale of the images.

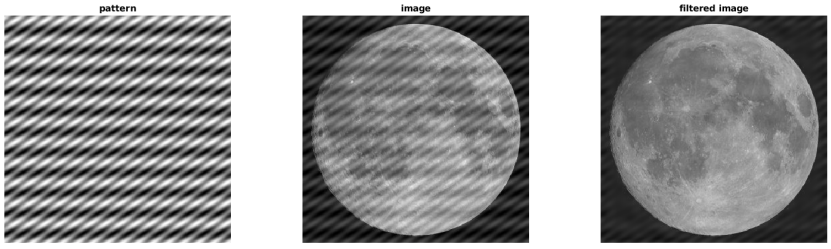
Figure 9: Periodic pattern noise filtering.

As we can see in the Fig.9, for a vertical sinusoidal pattern we can see that removing the minimum number of pixels in the Fourier space. The obtained results can be seen in the last image, that can be considered satisfactory in the brighter parts of the image, but we can still perceive a certain pattern in the black background, maybe due to the lack of details in that region and the proximity to the image limits.

Now we are going to make a new pattern with the combination of two sinusoids in different directions at the same level of intensity to check the algorithm that automatically erase the higher peaks.



(a) Fourier space of the images.



(b) Gray scale of the images.

Figure 10: Two periodic patterns noise filtering.

As we can see, the algorithm will work perfectly with as many patterns as we have as long as they are of a similar magnitude in the logarithmic scale and they are strong enough to distinguish it from the regular frequencies of the image.

Where is the information? In the Magnitude or in the Phase?

In this problem we have to explore where the most of the information in a frequency transformation is, in the magnitude or in the phase of the spectrum? To do so we are going to transform two images of the same size (one truck and on elephant) into the frequency domain and then generate new images mixing the magnitudes of one image with the angles of the other. After performing the swapping, we will loo for the reconstructed images with the inverse fft to determinate the information of the matrix.

Firs we are going to do it in the gray scale images, so we can see the first results of the problem

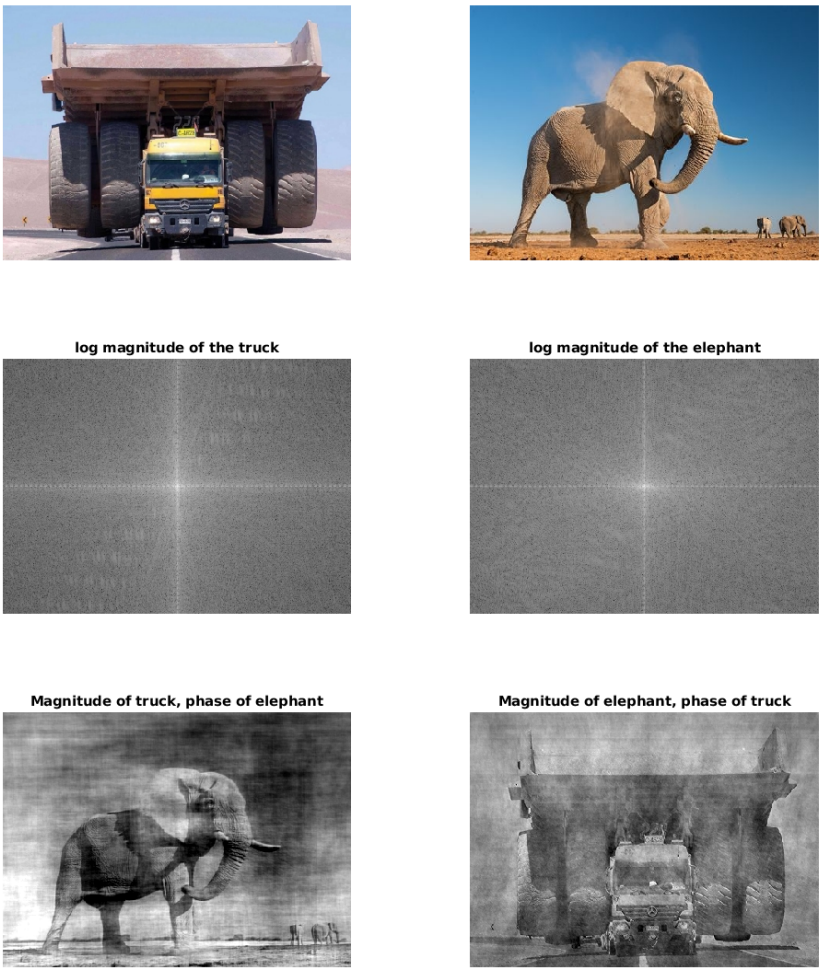


Figure 11: Gray scale reconstruction of the new images.

As we can see in the Fig.11, the image phase is very important to define the image because it will determine how the different frequency responses intertwine between them more than the relative intensity of the frequencies.

Finally, we are going to extend the problem to a colored image by performing the same operations with each one of the color channels at the same time.

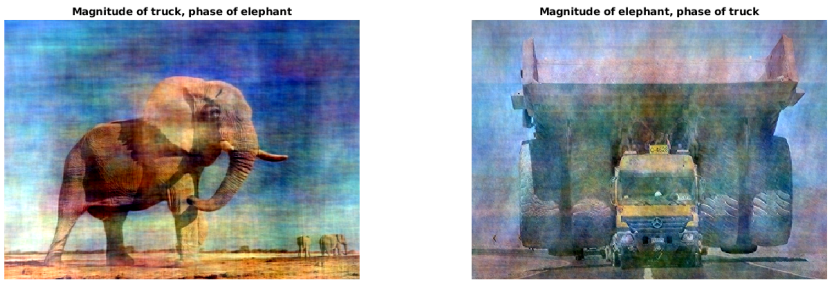


Figure 12: RGB reconstruction of the new images.

As we can see now in the colored image (Fig.12), the Magnitude will affect to the overall tone of the colors. The image with the truck magnitudes will remain with the same yellowish tone in most of the image while having the shape of the elephants image. And in the other image, the more blueish tones of the elephant images will also remain. With this, we can assume that the magnitude will have more relevance in the lower frequencies at time of creating the image to set the overall offsets, but most of the detailed information will remain in the phase pixels.

Advanced Topics in Computer Vision - Lab 5

Leonardo Palacios Fidalgo Ce Xu Zheng

June 5, 2022

Hadamard Transform

In this Lab session we are going to take advantage of the Walsh-Hadamard transformation to add a water mark to the images that we desire. This could be useful to protect the images against the stealing of intellectual property or simply marking the source of them without loosing the capability of recovering the original image if we also have the source of the watermark.

Adding watermark

To add a watermark to the image, the authors of [1] proposed a pipeline were we add the watermark in a gray scale to the luminance of the luminance of the image that we want to watermark in the Hadamard space, and then reconstruct the watermarked image with the new luminance.

To do so, we first need to obtain the gray-scale image of the watermark and transform the source image to the $YCbCr$ color space, were the first Y states for the luminance of the image. Then we need to reescale both the logo and the luminance of the image to the next power of 2 of the greatest dimension for the *fwht* function and finally obtain the Images in the Hadamard space. We can see this preprocessing code in the List.1

Listing 1: Preprocessing of the image

```
%% Adding the watermark

im = imread("photo2.jpg");
wm = imread("water_mark.jpg");
wm = rgb2gray(wm);
wm = double(wm);

PQ = paddedsize( size(im, 1:2), size(wm) , "PWR2");

yuv_im = rgb2ycbcr(im);
y_im = double(yuv_im(:,:,1));
u_im = yuv_im(:,:,2);
v_im = yuv_im(:,:,3);

y_im_augmented = zeros(PQ);
y_im_augmented(1:size(y_im,1), 1:size(y_im,2)) = y_im;

wm_augmented = zeros(PQ);
start_index = int32( (size(im, [1,2])-size(wm))/2 );
h_ind = start_index(1);
w_ind = start_index(2);
wm_augmented( h_ind:h_ind+size(wm, 1)-1, w_ind:w_ind+size(wm, 2)-1 ) = wm;

hadamard_y_im = fwht(y_im_augmented, PQ(1), 'hadamard');
hadamard_w_wm = fwht(wm_augmented, PQ(1), 'hadamard');
```

Once we have the images in the Hadamard space, we just have to mix them by adding a scaled version of the logo to the luminance of the source image. To do so, we need to specify that scaling

factor α that following the suggestions in [1], we are going to use a modified version of the sigmoid function.

$$\alpha = \frac{1}{\pi}\left(arctg(\mu(HI')) + \frac{\pi}{2}\right) \cdot \frac{1}{10^m}, \tag{1}$$

where $\mu(HI')$ is the average value of the source luminance in the Hadamard space, and m is a factor of visibility of the watermark. If $m = 0$, then a watermark prevails the host image, destroying it factually. If $m = 1$, then a watermark visibility is good without destroying the host image (visible watermarking scheme). If $m = 2$, then a watermark become invisible without degrading the quality of a host image (invisible watermarking scheme).

After that, we already have the scaling factor of the watermark and the luminance of the watermarked image in the Hadamard space will be the solution of the element wise addition of the previous luminance and the scaled logo. This could involve a risk of surpassing the maximum limit of the luminance and therefore destroying the image without possibility of a clean reconstruction if the scaling factor α is too high. With the new luminance we only have to concatenate it with the $CbCr$ matrices and transforming it back to an RGB image to obtain the watermarked image.

To test the results of the watermark, we are going to use a photo in the beach as the source image and the logo of Pinterest with a white background changed to grey-scale as the watermark (see Fig.1).

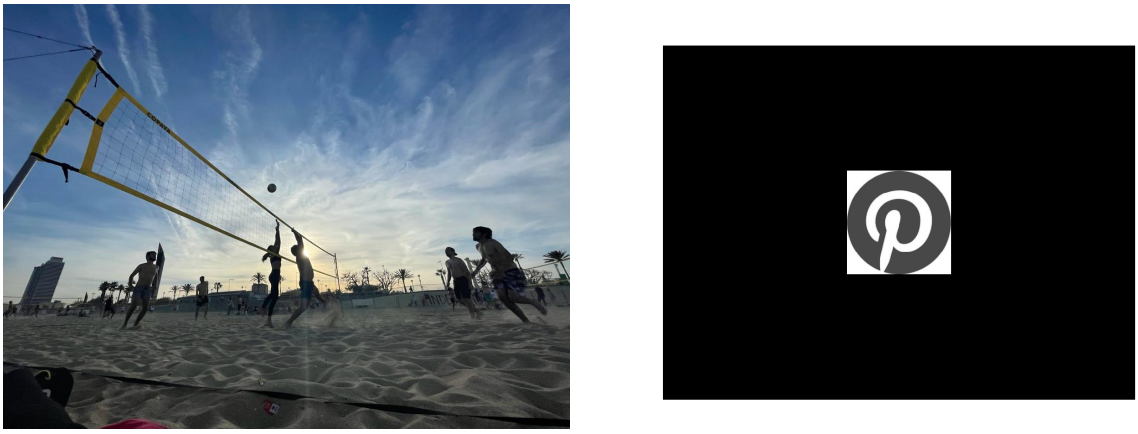


Figure 1: Source image (left) and watermark (right).

In the first place, we are going to see how the visibility factor will affect to the scaling factor by 10 times each, and how this one will affect the appreciation of the watermark.

(a) $m = 0$, $\alpha = 0.5060$. (b) $m = 1$, $\alpha = 0.0506$. (c) $m = 2$, $\alpha = 0.0051$.

Figure 2: Different grades of intensity of the watermark.

As we can see in the Fig.2, the expected behavior of the watermark is accomplished as predicted above. For $m = 0$, the watermark is very strong and saturates the luminance, In the other hand, for $m = 1$, the scaling factor is 10 times lower and we can appreciate the logo in the center of the image. However, if $m = 2$, the watermark is hardly perceptible.

The code for this part will be pretty simple as we can see in the List.2.

Listing 2: Obtaining watermarked image

```
1
2  m = 1;  % controlling parameter if 0 => image destroyed, 1 => good wm, 2 => not ...
       visible wm
3  alpha = 1/pi*( atan(mean(  hadamard_y_im , 'all')) + pi/2 )/(10^m)
4
5  hadamard_y2_im = hadamard_y_im + alpha*hadamard_w_wm;
6
7  y_im2 = ifwht(hadamard_y2_im, PQ(1), 'hadamard');
8  y_im2 = y_im2(1:size(y_im,1), 1:size(y_im,2));
9  yuv_im2 = cat(3,  y_im2 ,u_im,v_im);
10 im2 = ycbcr2rgb( yuv_im2 );
```

Remove the watermark

Once we have a watermarked image, we can extract it only with the source image of the watermark well positioned and the knowledge of which m they used to impact it.

To do so, we have to extract the luminance of the image in the Hadamard space as we have done before, and compute the α as we have done before. Note that if the logo changed the overall luminance considerably, this factor will not be the same as before and therefore, the extraction of it could not be satisfactory.

We can estimate the original luminance in the Hadamard space by extracting the scaled watermark and finally reconstruct the image as we have done before. The code will also be pretty simple in this case thanks to the matlab built in functions as we can see in the List.3.

Listing 3: Remove watermark

```
1  yuv_im2 = rgb2ycbcr(im2);
2  y_im2 = yuv_im2(:,:,1);
3  u_im = yuv_im2(:,:,2);
4  v_im = yuv_im2(:,:,3);
5
6  y_im2_augmented = zeros(PQ);
7  y_im2_augmented(1:size(y_im2,1), 1:size(y_im2,2)) = y_im2;
8
9  hadamard_y_im2 = fwht(y_im2_augmented, PQ(1), 'hadamard');
10
11 alpha = 1/pi*( atan(mean(  hadamard_y_im2 , 'all')) + pi/2 )/(10^m)
12
13 hadamard_y_im3 = hadamard_y_im2 - alpha*hadamard_w_wm;
14
15 y_im3 = ifwht(hadamard_y_im3, PQ(1), 'hadamard');
16 y_im3 = y_im3(1:size(y_im2,1), 1:size(y_im2,2));
17 yuv_im3 = cat(3,y_im3   ,u_im,v_im);
18 im3 = ycbcr2rgb( yuv_im3 );
```

The results of the logo extraction can be seen in the Fig.3.

(a) $m = 0$.

(b) $m = 1$.

(c) $m = 2$.

Figure 3: Different grades of intensity of the watermark.

As we expected, the saturation of the luminance for $m = 0$ is makes the extraction impossible and it will remain a shading where the luminance was saturated. For $m = 1$, the extraction results in a not appreciable watermark, and for the last case, the watermark was not appreciable since the beginning, so we cannot evaluate the result beyond knowing that it didn't worsen the image.

To look for the scaling factor threshold, we tried with different values of the α manually and transmitted to all the extractions. The results are shown in the Fig.4.

(a) $\alpha = 0.1$.

(b) $\alpha = 0.15$.

(c) $\alpha = 0.2$.

Figure 4: Different scaling factors of the watermark.

As we can see here, for $\alpha = 0.1$, the watermark is not appreciable in the recovered image, and for $\alpha = 0.2$, the watermark is clearly visible, but mostly in the more bright part of the original image, where the luminance might have saturated. We have found the threshold more or less in $\alpha = 0.15$, were you can still appreciate the logo if you look closely but might have gone unappreciated if it was not payed too much attention.

Extract the watermark

The last problem that we are going to solve is the extraction of the watermark when you have both the original image and the watermarked one. As we have done before, we can extract the luminance components of both images and transform them to the Hadamard space. Then we can compute the scaling factor with the Eq.1 and the mean of the original image, and finally obtain the logo in the Hadamard space by subtracting the original image luminance to the watermarked one and divide it by the scaling factor. The employed code can be seen in the List.4

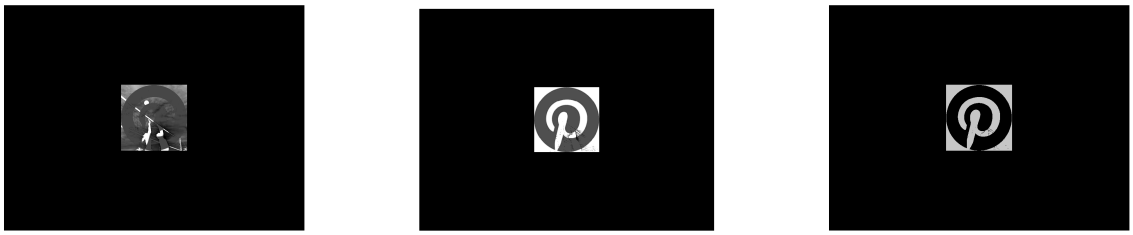
Listing 4: Extracting watermark

```
yuv_im = rgb2ycbcr(im);
y_im = double(yuv_im(:,:,1));
u_im = yuv_im(:,:,2);
v_im = yuv_im(:,:,3);
y_im_augmented = zeros(PQ);
y_im_augmented(1:size(y_im,1), 1:size(y_im,2)) = y_im;
hadamard_y_im = fwht(y_im_augmented, PQ(1), 'hadamard');

yuv_im2 = rgb2ycbcr(im2);
y_im2 = yuv_im2(:,:,1);
y_im2_augmented = zeros(PQ);
y_im2_augmented(1:size(y_im2,1), 1:size(y_im2,2)) = y_im2;
hadamard_y_im2 = fwht(y_im2_augmented, PQ(1), 'hadamard');

% alpha = 1/pi*( atan(mean(  hadamard_y_im , 'all')) + pi/2 )/(10^m)
```

```
16
17 hadamard_extracted_wm = (hadamard_y_im2-hadamard_y_im)/alpha;
18
19 wm_2 = ifwht(hadamard_extracted_wm, PQ(1), 'hadamard');
20 wm_2 = wm_2(1:size(y_im2,1), 1:size(y_im2,2));
```



(a) $m = 0$. (b) $m = 1$. (c) $m = 2$.

Figure 5: Different grades of intensity of the watermark.

As we can see in the Fig.5, the extraction of the quality of the extracted watermark is correlated with the quality of the removal of it. For $m = 0$, the watermark has a great influence of the background, where the lower lights has a greater prevalence in the new logo, and the white background is definitely not recovered. For $m = 1$, the extracted watermark is almost perfect although a small shadow is still casted over the extracted logo, but both the tone of the gray and the white background is distinguishable. Finally, for $m = 2$, the extracted watermark is much darker than the expected, what may mean that the watermarked image had almost no new information of the logo in the luminance after the quantization of it and posterior transformation to RGB.

Here we conclude the Laboratory Session about the Hadamard Transformation and it's application for the watermaking of images.

1 References

[1] Margarita Favorskaya, Eugenia Savchina and Aleksei Popov. Adaptive visible image watermarking based on Hadamard transform. IOP Conf. Series: Materials Science and Engineering 450 (2018) 052003