Texture segmentation: Co-occurrence matrix and Laws' texture masks methods

Guillaume Lemaître - Miroslav Rodojević
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com - miroslav.radojevic@gmail.com

I. Co-occurrence matrix - Introduction

To segment an image, different approach can be used: color, shape or texture. In this part, we will present a statistical method using co-occurrence matrix. This method allows to compute some statistics describing texture. Moreover, co-occurrence matrix method permits to carry information regarding the position of pixels where we compute statistics and solve problems known using only histograms of an image. In this section, we will introduce how to compute co-occurrence matrices, statistics allowing to describe textures. Then we will present a Matlab implementation of co-occurrence matrix method. We will conclude presenting results and influence of each parameter.

II. Theorical approach

A. Co-occurrence matrices

In this section, we will present how to compute co-occurrence matrices.

1) Parameters: The construction of gray-levels co-occurrence matrix depends on different parameters:

- The window size of the neighborhood where we will compute the matrix.
- The number of gray levels used to simplify and reduce the size of the matrix.
- The distance and the orientation that defined the pixel supervised at each iteration to construct the matrix.

2) Construction: To construct co-occurrence matrix, we considered a central pixel with a neighborhood defined by the window size in parameter. For each pixel of the neighborhood, we count the number of times that a pixel pairs appear specified by the distance and orientation parameters. Figure 1 represents an image.

0	0	1	1
0	0	1	1
0	2	2	2
2	2	3	3

Figure 1. Example of an image

The co-occurrence matrix with a distance of one pixel and an orientation of ninety degrees is as follows:

	0	1	2	3
0	6	0	2	0
1	0	4	2	0
2	2	2	2	2
3	0	0	2	0

Table I
Co-occurrence matrix corresponding to the figure 1 with an orientation of 90 degrees and a distance of one pixel

After computing the co-occurrence matrix, we need to describe the texture using statistics. The next section will present different statistics used to characterized a texture.

B. Statistics computation

In this section, we introduce different statistics used to describe texture after construction of the co-occurrence matrix. We will present how to compute these statistics and what is the meaning of these statistics. Usually, six descriptors exist to describe an image: maximum probability, correlation, contrast, uniformity or energy, homogeneity and entropy [1]. In the following sections, we will present statistics that we used inside

the algorithm. Hence, we will present contrast, energy, homogeneity and entropy.

1) Contrast: Contrast is a measure of intensity contrast between a pixel and its neighbor over the entire image. If the image is constant, contrast equal 0 while the biggest value can be obtained when the image is a random intensity image and that pixel intensity and neighbor intensity are very different. The equation of the contrast is as follows:

$$Con = \sum_{i=1}^{K}\sum_{j=1}^{K}(i-j)^2 p_{ij} \tag{1}$$

2) Energy: Energy is a measure of uniformity where is maximum when the image is constant. The equation of the contrast is as follows:

$$Ene = \sum_{i=1}^{K}\sum_{j=1}^{K}p_{ij}^2 \tag{2}$$

3) Homogeneity: Homogeneity measures the spatial closeness of the distribution of the co-occurrence matrix. Homogeneity equal 0 when the distribution of the co-occurrence matrix is uniform and 1 when the distribution is only on the diagonal of the matrix. The equation of the contrast is as follows:

$$Hom = \sum_{i=1}^{K}\sum_{j=1}^{K}\frac{p_{ij}}{1+|i-j|} \tag{3}$$

4) Entropy: Entropy measures the randomness of the elements of the co-occurrence matrix. Entropy is maximum when elements in the matrix are equal while is equal to 0 if all elements are different. The equation of the contrast is as follows:

$$Ent = -\sum_{i=1}^{K}\sum_{j=1}^{K}p_{ij}log_2(p_{ij}) \tag{4}$$

III. Algorithm design

In this section, we will present how we implemented co-occurrence matrix to compute statistics allowing to describe textures. The algorithm is available in the appendix A-A.

A. Parameters

We saw in the previous section that some parameters need to be fixed to compute the co-occurrence matrix. Hence, the user in the main function defined the following parameters:

- The window size of the neighborhood
- The number of gray levels
- The orientation and the distance parameters to construct the co-occurrence matrix

B. Matlab functions implemented

All functions needed are implemented Matlab. Hence, we used the following function to perform operations asked in the assignment:

- *graycomatrix*: this function allows to compute the co-occurrence matrix using parameters previously defined by the user.
- *graycoprops*: this function allows to compute all statistics describing the texture of the neighborhood. Hence, we can compute **contrast, homogeneity and energy**. However, we wrote a code (Appendix A-A2) allowing to compute the entropy which is not implemented in Matlab.
- *mat2gray*: this function allows to normalize an image between either two values defined by the user or 0.0 and 1.0.

We used other Matlab functions to display result images.

IV. Results and parameters influences

In this section, we will present results and the parameters influences on these results. We saw previously that we can play with the following set of parameters:

- The window size of the neighborhood
- The number of gray levels
- The orientation and the distance parameters to construct the co-occurrence matrix

A. Reference results

In this section, we will present results that we will consider than references to see the influences of different parameters. The set of parameters used was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.

Figure 2. Subtraction between homogeneity and energy descriptors for hand2.tif

- Distance: 1 pixel.

Result images are presented in Appendix B-A. We computed statistics for every sample images given. For the first image (*feli.tif*) 13, entropy statistic is the best descriptor to segment. On the second image (*hand2.tif*) 14, a combination between several should be used to obtain a good segmentation. Figure 2 shows a subtraction between homogeneity and energy descriptors which allow to segment the image.

In third case (*mosaic8.tif*) 15, we have to use a combination of the different descriptors to be able to do the segmentation. Finally, for the last image (*pingpong2.tif*) 16, we can use combinations allowing to segment easily. These combinations will depend on what we want to detect. In the next sections, we will present the roles of the different parameters.

B. Window size parameter

In the appendix B-B, we computed the set of images with different size of window to see the influence of this parameter. For all examples, the results is the same. The descriptor images are more and more blurring when the size of the window is bigger and bigger. Adjusting this parameter, we can deal with scale variation and periodic textures.

C. Number of gray levels parameter

This parameters allows to choose, the number of gray level of the input image. Hence, we simplify the image. Matlab offers two settings: 2 bits or 8 bits. 2 bits corresponds of binary image and can be used only to have the most simplify image possible. We try to put the number of gray levels at 2 bits and we are showing results in the appendix B-C. We can observe that results of the first and second images are not useful to help at a better segmentation (figures 33, 34). However, in the case of the two last pictures, this simplification can help to segment (figures 35, 36).

D. Distance parameter

We compute result images with a distance of 2 pixels and 4 pixels. These images are presented in the appendix B-D. Adjusting this parameter, we can detect periodic texture at different scale.

E. Orientation parameter

With this parameter, we can choose the orientation where we want to check pixels to create the co-occurrence matrix. We compute result images for different orientations: 45, 90, 135 degrees. These results are presented in the appendix B-E. We can observe that when the orientation is perpendicular to edges, we see these edges more accurate 56.

V. Conclusion

We implemented on this section, a method computing statistics using co-occurrence matrix. This method allows to find easier way to segment texture. In a first part, we saw a theoretically aspect of this method. In a second part, we presented the algorithm and how we implemented the method. Finally, we presented results and roles of each parameter.

VI. Laws' Texture Measures - Introduction

The texture energy measures developed by K. I. Laws [2] have been used for many diverse applications. These measures are computed by first applying small convolution kernels to a digital image, and then performing a non-linear windowing operation. We will first introduce the convolution kernels that we will refer to later. The 2-D convolution kernels typically used for texture discrimination are generated from the following set of one-dimensional convolution kernels of length three and five: $L_3 = \begin{bmatrix}1 & 2 & 1\end{bmatrix}$, $E_3 = \begin{bmatrix}1 & 0 & -1\end{bmatrix}$, $S_3 = \begin{bmatrix}1 & -2 & 1\end{bmatrix}$

$L_5 = \begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$, $E_5 = \begin{bmatrix}-1 & -2 & 0 & 2 & 1\end{bmatrix}$, $S_5 = \begin{bmatrix}-1 & 0 & 2 & 0 & -1\end{bmatrix}$, $W_5 = \begin{bmatrix}-1 & 2 & 0 & -2 & 1\end{bmatrix}$, $R_5 = \begin{bmatrix}1 & -4 & 6 & -4 & 1\end{bmatrix}$ These mnemonics stand for *Level - average grey level*, *Edge - extract edge features*, *Spot - extract spots*, *Wave - extract wave features*, and *Ripple - extract ripples* [2]. All kernels except L_5 and L_3 are zero-sum. Convolution of texture with Laws' masks and calculation of energy statistics gives description features of a texture that can be used for texture discrimination.

Table II
LIST OF 3×3 AND 5×5 LAW MASKS (2-D KERNELS) USED TO PERFORM TEXTURE ANALYSIS.

Kernel name	Kernel value using 1-D kernels	description - features extracted from texture
L_3L_3	$L_3^T L_3$	expressing grey level intensity of 3 neighbouring pixels in both horizontal and vertical direction
L_3E_3	$L_3^T E_3$	edge detection in horizontal direction and grey level intensity in vertical direction
L_3S_3	$L_3^T S_3$	spot detection in horizontal direction and grey level intensity in vertical direction
E_3L_3	$E_3^T L_3$	grey level intensity in horizontal direction and edge detection in vertical direction
E_3E_3	$E_3^T E_3$	edge detection in both vertical and horizontal direction
E_3S_3	$E_3^T S_3$	spot detection in horizontal direction and edge detection in vertical direction
S_3L_3	$S_3^T L_3$	grey level intensity in horizontal direction and spot detection in vertical direction
S_3E_3	$S_3^T E_3$	edge detection in horizontal direction and spot detection in vertical direction
S_3S_3	$S_3^T S_3$	spot detection in both horizontal and vertical direction

possible 5×5 Laws' masks

L5L5	E5L5	S5L5	W5L5	R5L5
L5E5	E5E5	S5E5	W5E5	R5E5
L5S5	E5S5	S5S5	W5S5	R5S5
L5W5	E5W5	S5W5	W5W5	R5W5
L5R5	E5R5	S5R5	W5R5	R5R5

VII. ALGORITHM ANALYSIS

A. Step I. Apply Convolution Kernels

From these one-dimensional convolution kernels, we can generate different two-dimensional convolution kernels by convolving a vertical 1-D kernel with a horizontal 1-D kernel. As an example, the L_5E_5 kernel is found by convolving a vertical L_5 kernel with a horizontal E_5 kernel. In matrix multiplication, this would be expressed as $L_5E_5 = L_5^T \times E_5$. A listing of all 3×3 and 5×5 kernel masks used in this analysis is given in Table II. Obtained kernels are then used for calculating convolution (Figure 3). Given a sample image with N rows and M columns that we want to perform texture analysis on, we first apply each of our selected convolution kernels to the image. For certain applications, only a subset of all offered kernels will be used. The result is a set of grey-scale images, each having dimension $N - window\_size + 1 \times M - window\_size + 1$ where $window\_size$ can be 3 or 5, depending on kernel size. These convolution outputs will form the basis for our textural analysis.

B. Step II. Performing Windowing Operation

Next step is performing windowing operation (Figure 4) where we replace every pixel in images obtained after convolution with a texture energy measure (TEM) at the pixel. We do this by looking in a local neighbourhood

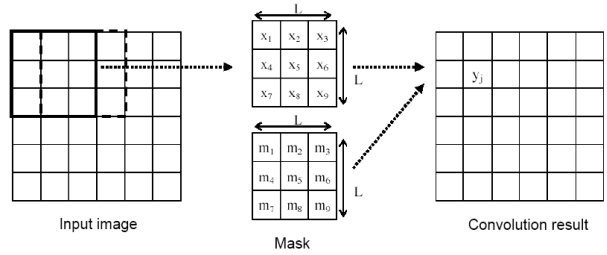


Figure 3. Mask convolution. $Pixels = N = L \times L$, $y_i = \Sigma_{i=1}^{N} x_i \times m_i$.

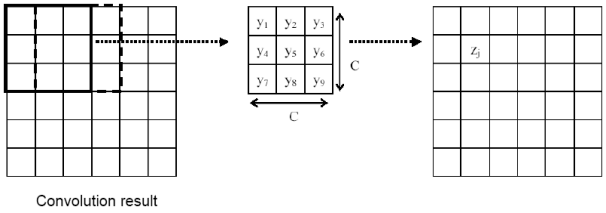


Figure 4. Statistic computation. $Pixels = N = C \times C$, $z_j = f(y_1, y_2, ...y_N)$ z_j can be mean of the neighbourhood values, mean of the absolute values, and standard deviation of the neighbourhood values.

of different size (for instance, lets use a 15×15 square) around each pixel and count the following three descriptors:

- averaging the absolute values of the neighbourhood pixels $\mu = \frac{\Sigma_N neighbouring\_pixels}{N}$
- averaging the values of the neighbourhood pixels $\sigma = \frac{\Sigma_N abs(neighbouring\_pixels)}{N}$
- calculating standard deviation for the neighbourhood pixels $abs\_\mu = \sqrt{\frac{\Sigma_N (neighbouring\_pixels - \mu)^2}{N}}$

We generate a new set of images, which we will be referred to as the TEM images, in this stage of image processing. Enumerated non-linear filters are applied to each of our selected images.

C. Step III. Normalize Features

Next step, all images obtained after windowing operation should be normalized in order to be presented well as images. Min-max normalization maps values of image matrix within [0 1] range so that they can be later used as arguments of *imshow()* function when plotting the image.

VIII. Design and implementation of the solution

MATLAB function *Law_mask()* was designed to implement Laws' texture measurements for a particular image, using several arguments as parameters. This function is further used in main program *main_law_filters.m* that calls this function in a for loop so that results are compared for various input images and input arguments, resulting in a discussion and recording of resulting figures.

A. Law_mask() function

is the heart of the implementation. The function outputs normalized Texture Energy Measure (TEM) image [2], and takes

- name of the image with extension as first argument,
- type of the filter as second argument (for instance 'L3L3' for $L3L3$ Law filter),
- size of the window used for statistical calculations when extracting descriptors as third argument,
- type of the statistic calculated as fourth argument (can take values 'MEAN', 'ABSM', and 'STDD' as explained),
- type of normalization as the last, fifth argument (can take values 'MINMAX', and 'FORCON' representing classic min-max normalization and normalizing any TEM image pixel-by-pixel with the L*x*L*x* TEM image for better contrast, respectively).

All of the input arguments, except window size, are given as strings. At the beginning of the realization, function first defines kernel that will be used for convolving (Law mask). It is set according to user defined input string and 1-D kernels using *switch* program structure. All the possible combinations are enabled for using $\rightarrow$ 9 of them for 3×3 masks and 25 for 5×5 masks (Table II). After reading and converting image to grey-scale, MATLAB function *conv2()* is used to work out the convolution taking image matrix and filter kernel matrix as argument. Additional input parameter is set to *'valid'* so that it returns only those parts of the convolution that are computed without the zero-padded edges. This will imply shortening of the resulting image depending on the size of the kernel that was used. Next step in function design is statistic windowing computation taking into account neighbourhood of the pixel. Since statistical descriptor can be (1.) mean, (2.) mean of absolute values, or (3.) standard deviation, each of these is possible to calculate depending on string input - *statistic_type* string. In cases of mean ('MEAN') and mean of absolute pixel values ('ABSM'), execution is fast and defaults usage of *fspecial()* MATLAB function for obtaining the average window, with *'average'* option set, and subsequently calling *conv2()*that takes "post-convolution" image matrix and average window as arguments and gives out TEM image. For 'STDD' statistic, execution takes longer and additional function *image_stat_stand_dev()* is used.

```
image = im2double(rgb2gray(imread(file_name)));
image_conv = conv2(image,filter_mask,'valid');
av_filter = fspecial('average', [window_size window_size]);
image_conv_TEM = conv2(image_conv,av_filter,'valid');
image_out = normalize(image_conv_TEM);
```

Final step is to normalize the image matrices before presenting. Two options are available for normalization: min-max normalization ('MINMAX') and normalizing any TEM image pixel-by-pixel with the LxLx TEM image (for contrast, 'FORCON'). First option is defaul option for this assignment, but the second one is treated as possible improvement. Additional functions than are used include *normalize()* and *image_stat_stand_dev()*. First one does classic min-max normalization, and the second performs windowing operation, computing standard deviation value for each window sequentially. Both are defined at the end of *Law_mask()* function code. Appropriate syntax that would accomplish Laws' mask for an image named 'image.tif', with filter: 'L3S3', window size: 15, statistic type: absolute mean and normalization type: 'MINMAX'. is shown in next MATLAB code line:

```
image_out = Law_mask('image.tif', 'L3S3', 15, 'ABSM', 'MINMAX');
```

IX. Results and analysis

This section will introduce the results after using introduced Laws' masks and texture energy measurement values for different images: *feli.tif, hand2.tif, pingpong2.tif* and *mosaic8.tif* (Figure 5).

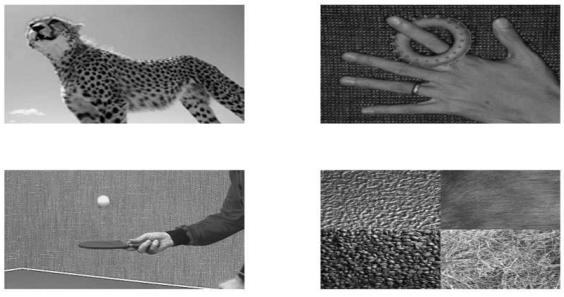


Figure 5. Test images in grey-scale.

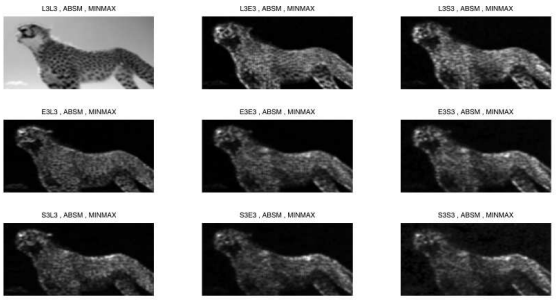


Figure 6. *feli.tif* after convolution with all possible 3×3 Laws' filters, using window size 7 and average of the absolute values $(ABSM)$ as statistic descriptor.

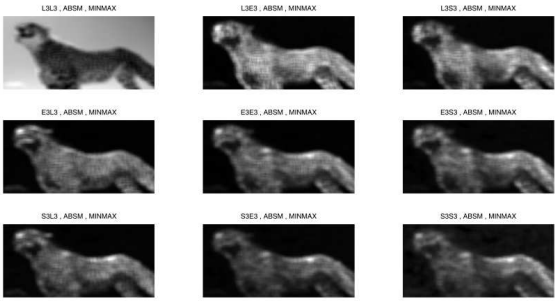
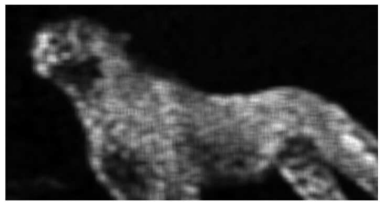


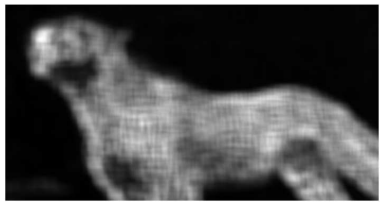
Figure 7. *feli.tif* after convolution with all possible 3×3 Laws' filters, using window size 15 and average of the absolute values $(ABSM)$ as statistic descriptor.

A. *feli.tif image*

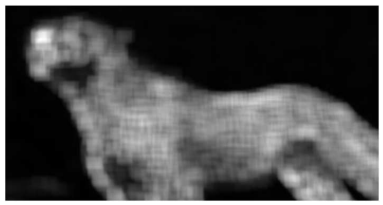
feli.tif image has two distinctive textures: one that is uniform (background area) and the other one that covers the object area - animal. Judging on filter output, and knowing that $L3L3$ and $L5L5$ kernels are the only ones without zero-mean, they tend to be suitable suitable for distinction of the areas of the image with uniform grey level. In this case, any combination of 1-D kernels L3 with S3 or E3; or L5 with S5 or E5 outcomes with distinction of two textures. Technically, S (Spots) kernel should be the one suitable for application, but since texture spots are bigger than the size of the Law filter, E (Edge) kernel gives similar results. In order to have clear border between two textures, L (grey level) is used to "filter" the background. In Figure 6, all the possible filtering results were considered, for window size 7 which compromises between sharpness, contrast and uniformity of resulting output (Figures 6 and 7). Mean of absolute values ('ABSM') and standard deviation ('STDD') statistics descriptors are suitable. Mean statistic descriptor equalizes results of kernel convolution for two textures, proved to be less efficient therefore, not suitable for this image. Some results of Laws' method for *feli.tif* are shown together with execution times in Figure 8. Execution time significantly increases when calculating standard deviation descriptor due to more demanding calculus.



(a)
$Law\_mask('feli.tif','L3S3',11,'ABSM','MINMAX');$



(b)
$Law\_mask('feli.tif','L3E3',15,'ABSM','MINMAX');$



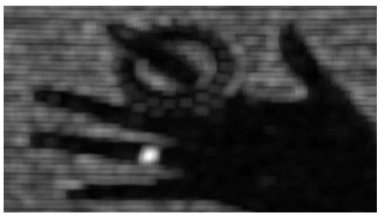
(c)
$Law\_mask('feli.tif','L3S3',15,'STDD','MINMAX');$

Figure 8. Results after applying 3×3 filters to *feli.tif* using *absolute mean* and *standard deviation* descriptor with different window sizes. Execution times: Figure 8(a) 0.023 s, 8(b) 0.039 s and 8(c) 4.681 s.

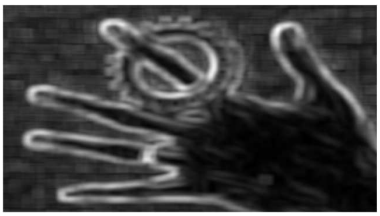
B. *hand2.tif image*

Background of this image is a texture with distinctive lines that cross: edges and spots. In order to extract it well using Laws' kernels, S (Spot) and E (Edge) seem to be appropriate choice. Combined among themselves (Figure 9(a)) or with L (grey level intensity) as in Figure 9(b). As statistical descriptors, mean of absolute values and standard deviation are chosen, since they respectively refer to frequency and grey level intensity that describes certain texture. Higher intensity when using 'ABSM' can be a sign of higher intensity for certain feature, and intensity keeps constant if the feature is uniformly distributed in a texture area. On the other hand, higher intensity when using 'STDD' can be a sign of higher frequency of intensity change for a certain feature. It is evident that involving L in kernel brings more attention to grey level intensity (extracting fingerring that was higher uniform grey level intensity in Figure 9(b)). Figure 9(c) can be an example of using

(a)
$Law\_mask('hand2.tif',' E5S5', 7,' ABSM',' MINMAX');$



(b)
$Law\_mask('hand2.tif',' L5S5', 9,' STDD',' MINMAX');$



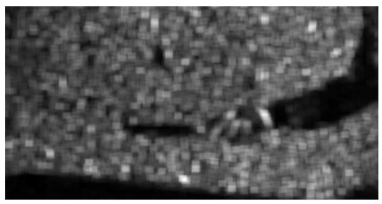
(c)
$Law\_mask('hand2.tif',' L5L5', 9,' STDD',' MINMAX');$

Figure 9. Results after applying 5×5 filters to *hand2.tif* using *absolute mean* and *standard deviation* descriptor with window sizes 7 or 9. Execution times: Figure 9(a) 0.013 s, 9(b) 2.848 s and 9(c) 2.873 s.

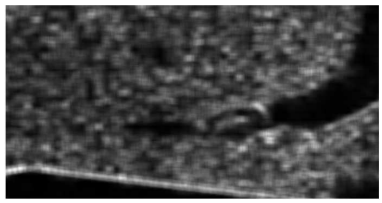
standard deviation feature to extract frequency changes of grey-level change in all directions. At the places of transition, frequency becomes high, which is visible when measuring its intensity, like in Figure 9(c).

C. *pingpong2.tif image*

Similarly to previous comments about conform background, *pingpong2.tif* image has a textured background that can be isolated from areas with constant grey level using edge (E) kernel in combination with spots detection (S), like it was presented in Figure 10(a). Figure 10(b) examines changes in grey level intensity in vertical direction and detects spots in horizontal direction which leads to extraction of the border line of the table. Same detection of the line would be accomplished using 'E3L3' or 'L3E3' kernel because both emphasize edges in one of the directions, just that in that case background would not have as bright intensity as it has in previous results (Figure 10(c)).



(a)
$Law\_mask('pingpong2.tif',' S3E3', 7,' ABSM',' MINMAX');$



(b)
$Law\_mask('pingpong2.tif',' L3S3', 7,' STDD',' MINMAX');$

(c)
$Law\_mask('pingpong2.tif',' E3L3', 7,' STDD',' MINMAX');$

Figure 10. Results after applying 3×3 filters to *pingpong2.tif* using *absolute mean* and *standard deviation* descriptor with window size 7. Execution times: Figure 10(a) 0.015 s, 10(b) 3.712 s and 10(c) 3.578 s.

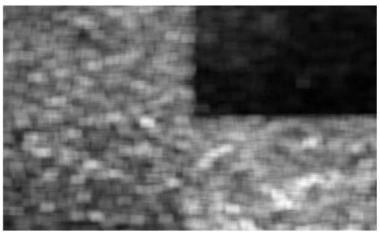
D. *mosaic8.tif image*

Results of treating textures with Laws' masks are shown in Figure 11. Again, as in previous cases, frequency and intensity of the texture features are calculated. It appears that such features are not informative enough to describe each texture with uniform intensity level.

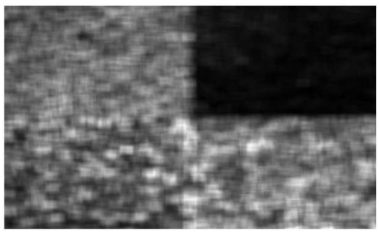
X. Improvements

A. Normalize Features for Contrast

All convolution kernels used thus far are zero-mean with the exception of the $L3L3$ kernel, and $L5L5$. In accordance with Laws' suggestions, we could therefore use this as a normalization image; normalizing any TEM image pixel-by-pixel with the $L5L5$ TEM image will normalize that feature for better contrast (Figure 12). For this purpose, normalization argument string for the function *Law_mask()* is set to 'FORCON'.



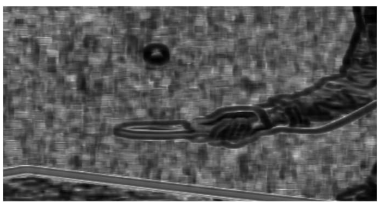
(a)
$Law\_mask('mosaic8.tif','E3S3',9,'STDD','MINMAX');$



(b)
$Law\_mask('mosaic8.tif','L3S3',9,'ABSM','MINMAX');$

Figure 11. Results after applying 3×3 filters to *mosaic8.tif* using *absolute mean* and *standard deviation* descriptor with window size 9. Execution times: Figure 11(a) 2.946 s and 11(b) 0.014 s.

(a) 'MINMAX' normalization



(b) 'FORCON' normalization using $LxLx_{x=3,5}$ TEM image as normalization factor

Figure 12. Results after applying 3×3 filters to *mosaic8.tif* using *standard deviation* descriptor with window size 7. Normalization type: Figure 12(a) 'MINMAX' and 12(b) 'FORCON'.

B. Combine Similar Features

For many applications, "directionality" of textures might not be important. If this is the case, then similar features can be combined to remove a bias from the features from dimensionality. For example, $L5E5$ TEM image is sensitive to vertical edges and $E5L5$ TEM is sensitive to horizontal edges. If we add these TEM images together, we have a single feature sensitive to simple "edge content". Following this example, features that were generated with transposed convolution kernels are added together. For instance, $E5L5TR = E5L5T + L5E5T$ or $S5L5TR = S5L5T + L5S5T$, or $E5E5TR = E5E5T * 2$ etc [3].

XI. Conclusions

In this assignment we discussed about two different texture feature extraction methods that are used in image analysis. Results show that there is considerable performance variability between texture methods. Gray Level Co-occurrence Method (GLCM) yields better results compare to Laws' Texture measures method. However, time complexity of GLCM is a bigger problem since this method is time consuming, depending on statistics complexity obtained and significantly slower than Laws method. Laws method shows good time performance, but it, similarly to GLCM, depends on calculated statistics complexity, hence performing much slower when working with demanding descriptors such as standard deviation.

APPENDIX A
CODE

A. Co-occurrence matrix algorithm

1) Main function:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Main file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Matlab parameters
clc; close all; clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters
window_size = 7;
nb_graylevels = 8;
distance = 1;
direction = [0 distance];
min_norm = 0;
max_norm = 255;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Start the counter time
tic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Read image
im_in = imread('feli.tif');

%%% Convert the image in grayscale
imgray_in = rgb2gray(im_in);

%%% Resize the image
imgray_in = imresize(imgray_in, 0.5);

%%% Compute parameters of image
[height_im, width_im] = size(imgray_in);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% R-IMG pre-allocation
contrast_rimg = zeros(height_im -(window_size-1),width_im -(window_size-1));
energy_rimg = zeros(height_im -(window_size-1),width_im -(window_size-1));
homogeneity_rimg = zeros(height_im -(window_size-1),width_im -(window_size-1));
entropy_rimg = zeros(height_im -(window_size-1),width_im -(window_size-1));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Computation of stats')

%%% Create R-Img for each possible pixel
for row = ((window_size + 1)/2):(height_im - ((window_size - 1)/2))
    for col = ((window_size + 1)/2):(width_im - ((window_size - 1)/2))
        %%% Compute statistics for the new pixel
        comatrix = graycomatrix(imgray_in((row - ((window_size -1)/2)):
        (row + ((window_size -1)/2)),(col - ((window_size - 1)/2)):
        (col + ((window_size -1)/2))),'NumLevels',nb_graylevels,'Offset',direction);
        stats = graycoprops(comatrix,{'Contrast','Energy','Homogeneity'});
        contrast_rimg((row - (window_size + 1)/2) + 1,
        (col - (window_size + 1)/2) + 1) = stats.Contrast;
        energy_rimg((row - (window_size + 1)/2) + 1,
        (col - (window_size + 1)/2) + 1) = stats.Energy;
        homogeneity_rimg((row - (window_size + 1)/2) + 1,
        (col - (window_size + 1)/2) + 1) = stats.Homogeneity;
        entropy_rimg((row - (window_size + 1)/2) + 1,
        (col - (window_size + 1)/2) + 1) = entropy_comp(comatrix);
    end
end

disp('stats computed')

%%% Apply the normalization
%%% Normalize the matrix image between 0.0 and 1.0
contrast_rimg_abs_norm = mat2gray(contrast_rimg);
energy_rimg_abs_norm = mat2gray(energy_rimg);
homogeneity_rimg_abs_norm = mat2gray(homogeneity_rimg);
entropy_rimg_abs_norm = mat2gray(entropy_rimg);
%%% Normalize the matrix image between a minimum value and maximum value
contrast_rimg_min_max_norm = mat2gray(contrast_rimg,[min_norm max_norm]);
energy_rimg_min_max_norm =  mat2gray(energy_rimg,[min_norm max_norm]);
homogeneity_rimg_min_max_norm =  mat2gray(homogeneity_rimg,[min_norm max_norm]);
entropy_rimg_min_max_norm = mat2gray(entropy_rimg,[min_norm max_norm]);
%%% Normalize the matrix image between 0 and 255
contrast_rimg_0_255 = uint8(contrast_rimg_min_max_norm * 255);
energy_rimg_0_255 = uint8(energy_rimg_min_max_norm * 255);
homogeneity_rimg_0_255 = uint8(homogeneity_rimg_min_max_norm * 255);
entropy_rimg_0_255 = uint8(entropy_rimg_min_max_norm * 255);

%%% Stop the counter time
toc

%%% Plot image
figure;
%%% Contrast
subplot(2,2,1);
imshow(contrast_rimg_abs_norm);
title('Contrast image');
```

```
%%% Energy
subplot(2,2,2);
imshow(energy_rimg_abs_norm);
title('Energy image');
%%% Homogeneity
subplot(2,2,3);
imshow(homogeneity_rimg_abs_norm);
title('Homogeneity image');
%%% Entropy
subplot(2,2,4);
imshow(entropy_rimg_abs_norm);
title('Entropy image');
```

2) Entropy function:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to compute entropy statistic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [entropy] = entropy_comp (comatrix)

%%% Computation of the size of the comatrix
[height_im,width_im] = size(comatrix);

%%% Intialisation
entropy = 0;

for i = 1:height_im
    for j = 1:width_im
        if( comatrix(i,j) ≠ 0 )
            entropy = entropy + (comatrix(i,j)*log2(comatrix(i,j)));
        end
    end
end

entropy = - entropy;
```

B RESULT IMAGES

A. Reference results

In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

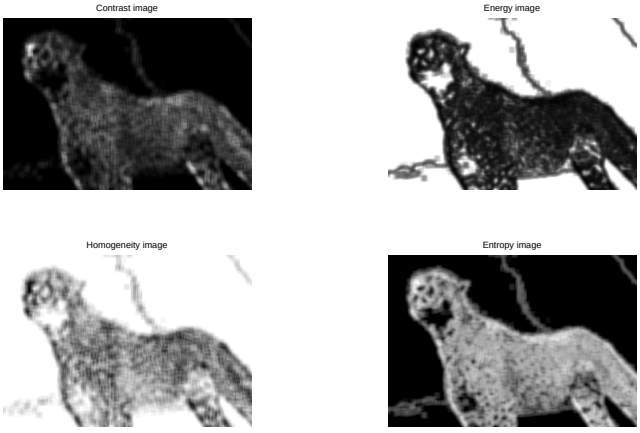


Figure 13. Reference results for feli.tif with execution time equal to 132.10 seconds

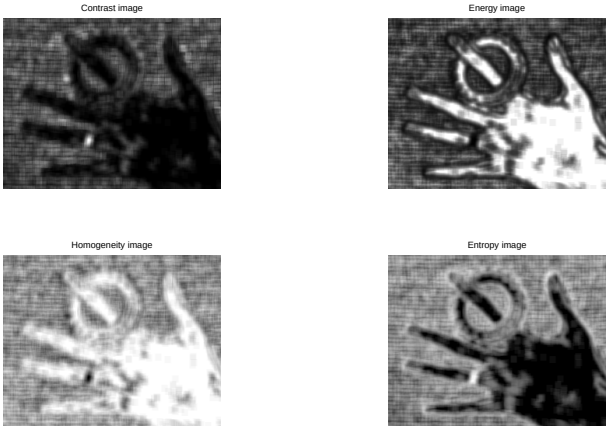


Figure 14. Reference results for hand2.tif with execution time equal to 85.57 seconds

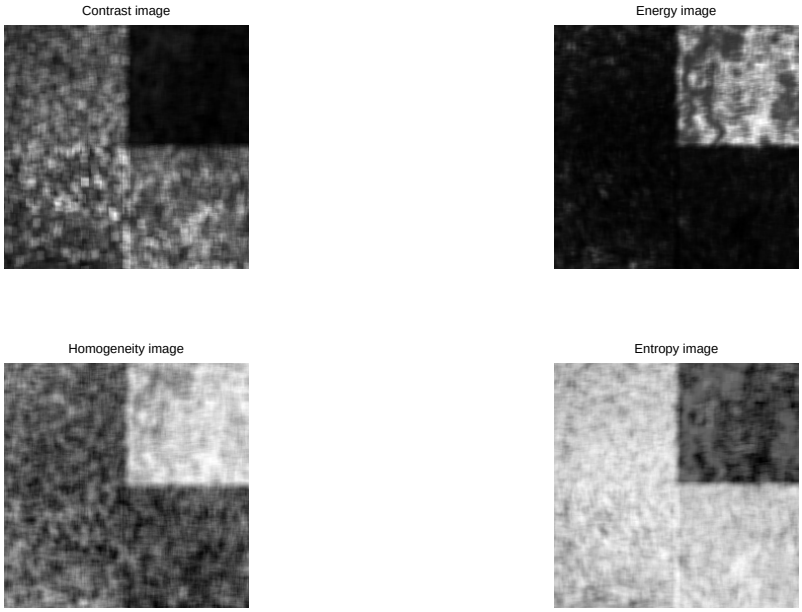


Figure 15. Reference results for mosaic8.tif with execution time equal to 85.70 seconds

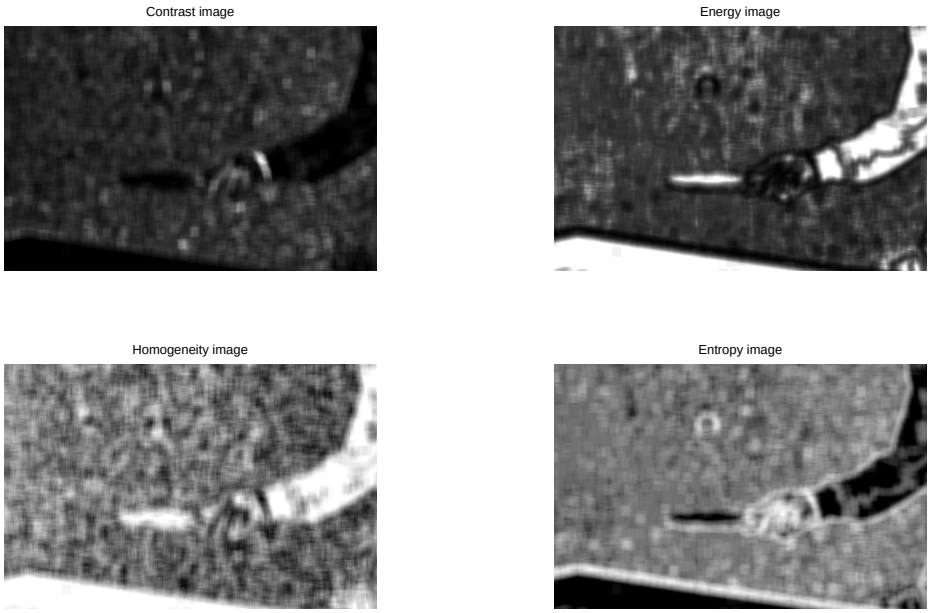


Figure 16. Reference results for pingpong2.tif with execution time equal to 104.36 seconds

B. Window size variations

In this section, we changed the value of the window size.

1) Window size: 5 pixels: In this section, the set of parameters was:

- Window size: 5 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

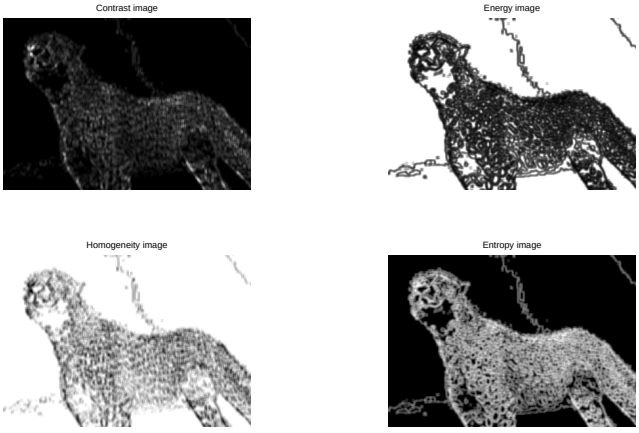


Figure 17. Window size of 5 pixels for feli.tif with execution time equal to 139.54 seconds

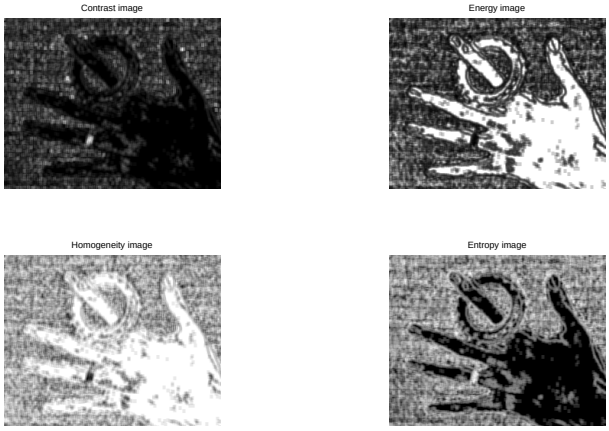


Figure 18. Window size of 5 pixels for hand2.tif with execution time equal to 89.04 seconds

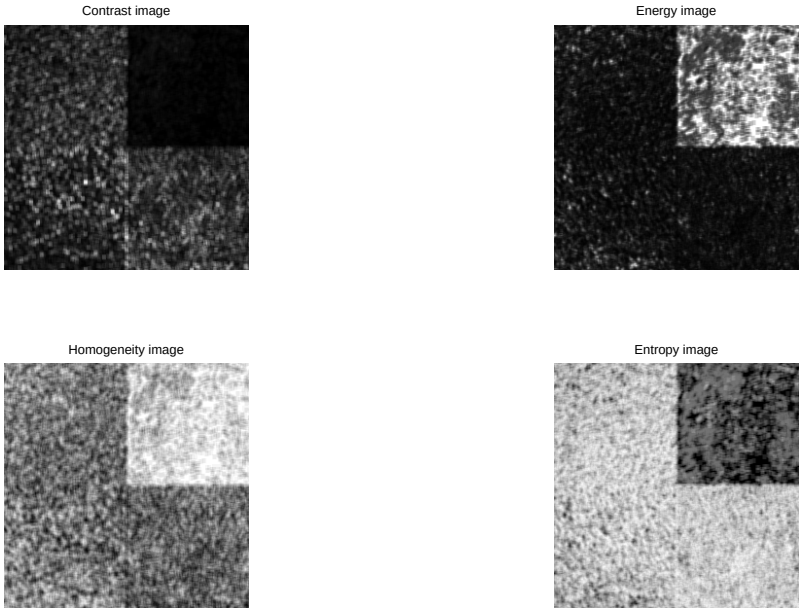


Figure 19. Window size of 5 pixels for mosaic8.tif with execution time equal to 94.84 seconds

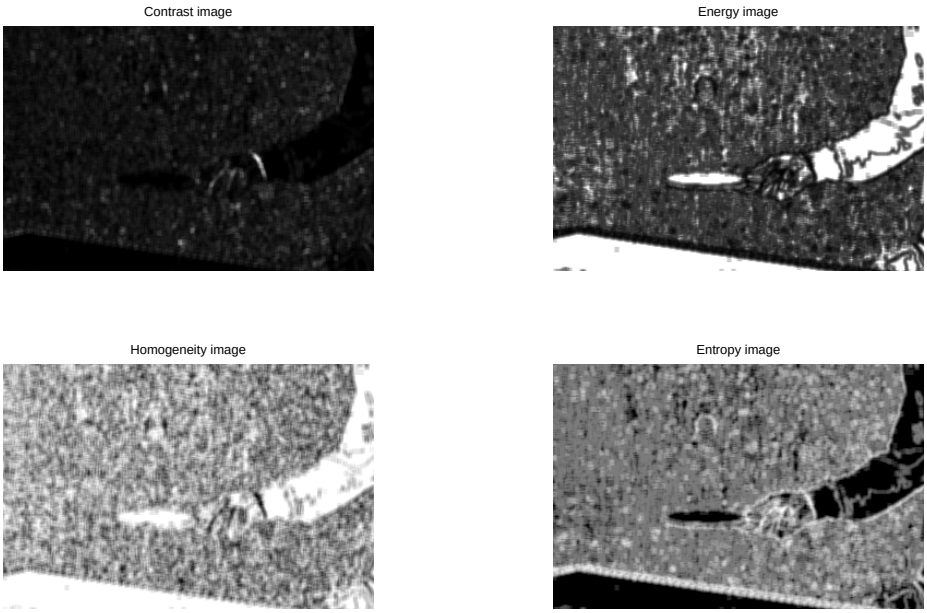


Figure 20. Window size of 5 pixels for pingpong2.tif with execution time equal to 110.38 seconds

2) Window size: 7 pixels: In this section, the set of parameters was:

- Window size: 7 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

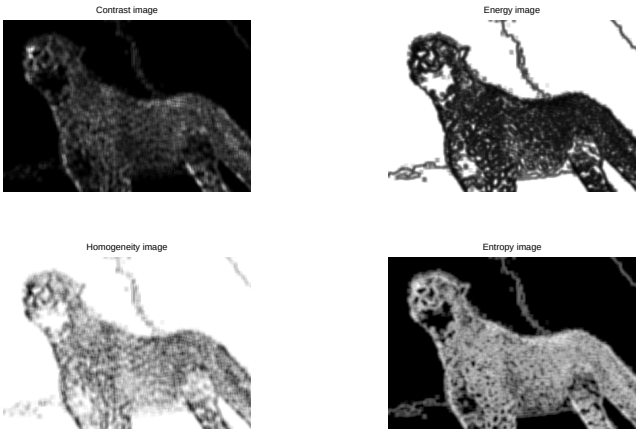


Figure 21. Window size of 7 pixels for feli.tif with execution time equal to 141.31 seconds

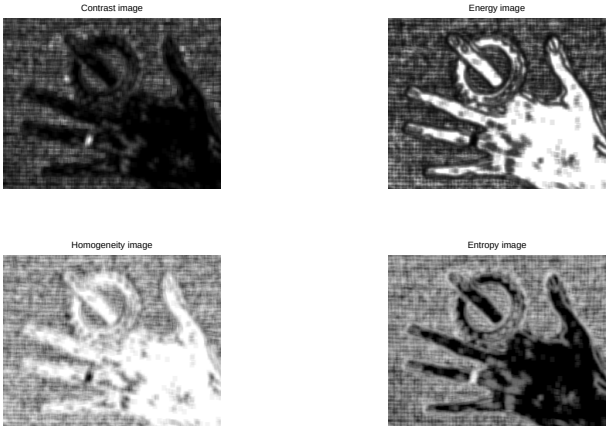


Figure 22. Window size of 7 pixels for hand2.tif with execution time equal to 84.91 seconds

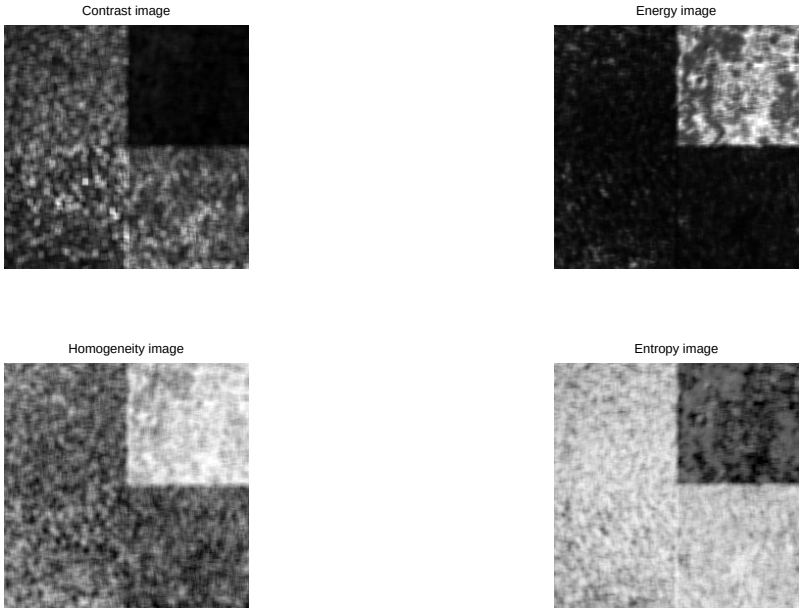


Figure 23. Window size of 7 pixels for mosaic8.tif with execution time equal to 86.79 seconds

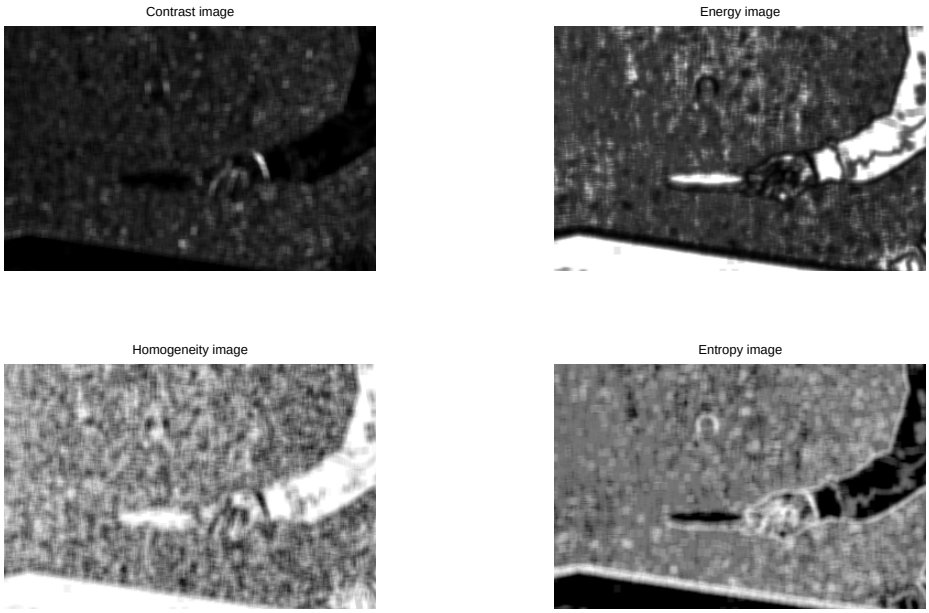


Figure 24. Window size of 7 pixels for pingpong2.tif with execution time equal to 108.46 seconds

3) Window size: 11 pixels: In this section, the set of parameters was:

- Window size: 11 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

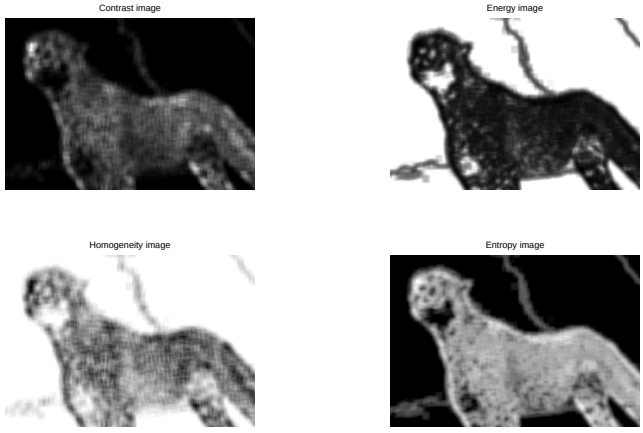


Figure 25. Window size of 11 pixels for feli.tif with execution time equal to 133.13 seconds

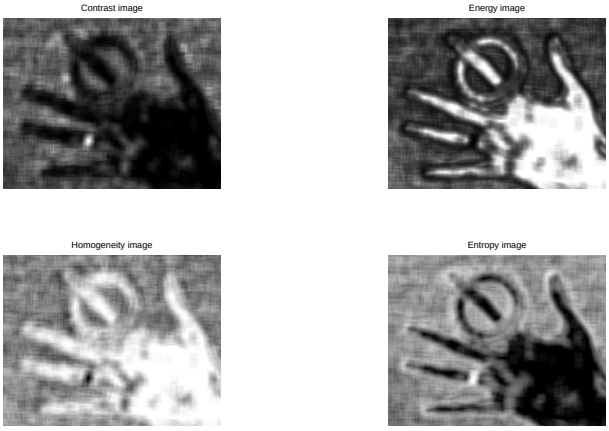


Figure 26. Window size of 11 pixels for hand2.tif with execution time equal to 89.23 seconds

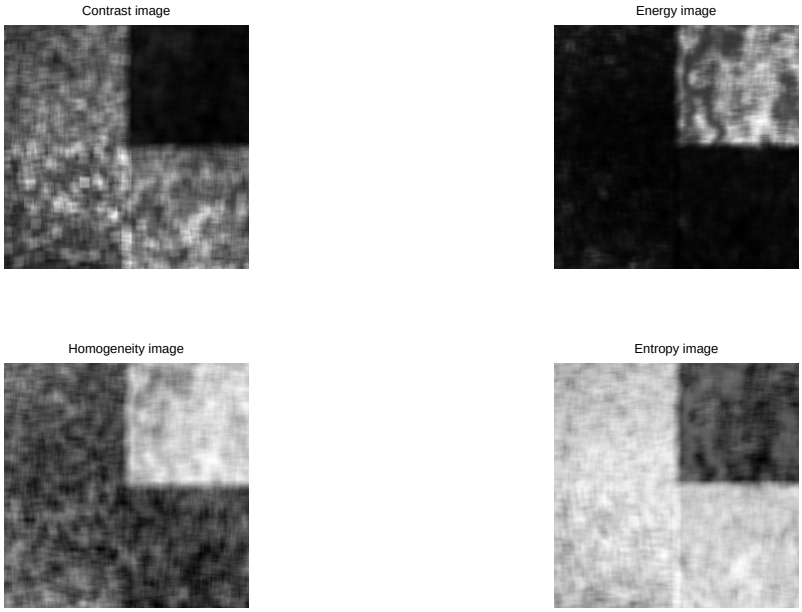


Figure 27. Window size of 11 pixels for mosaic8.tif with execution time equal to 88.02 seconds

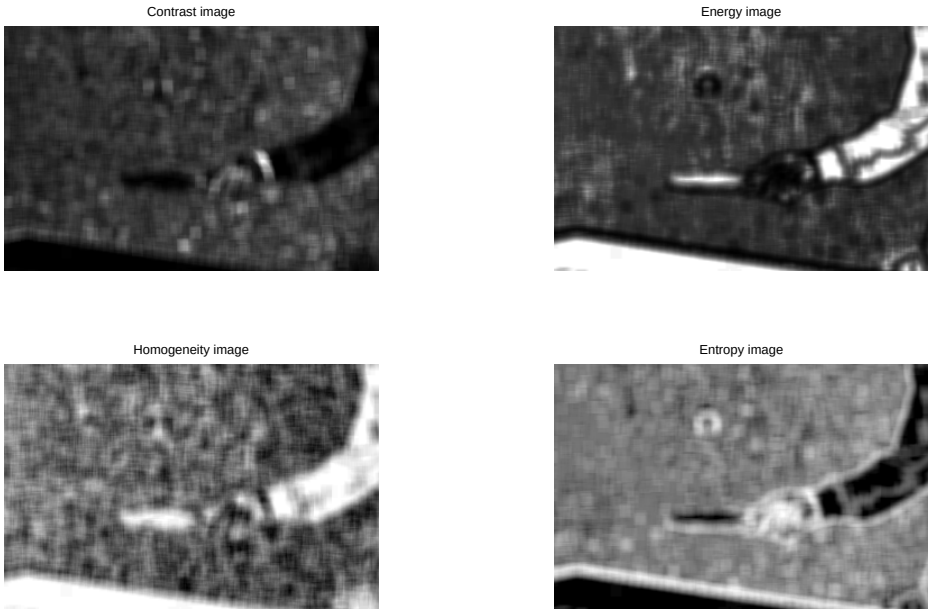


Figure 28. Window size of 11 pixels for pingpong2.tif with execution time equal to 103.07 seconds

4) Window size: 13 pixels: In this section, the set of parameters was:

- Window size: 13 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

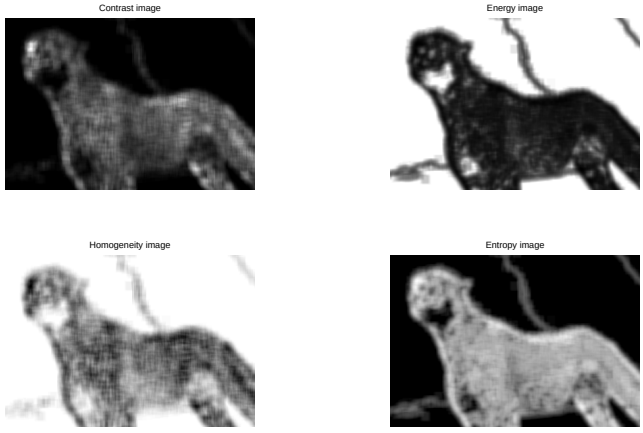


Figure 29. Window size of 13 pixels for feli.tif with execution time equal to 135.15 seconds

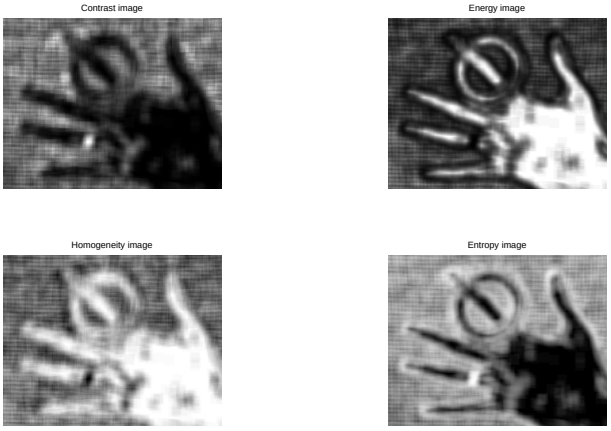


Figure 30. Window size of 13 pixels for hand2.tif with execution time equal to 94.67 seconds

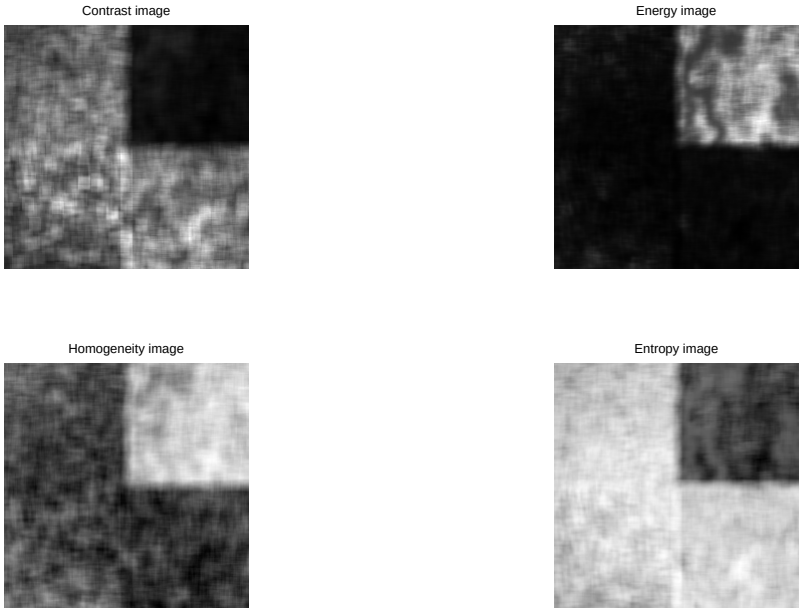


Figure 31. Window size of 13 pixels for mosaic8.tif with execution time equal to 84.52 seconds

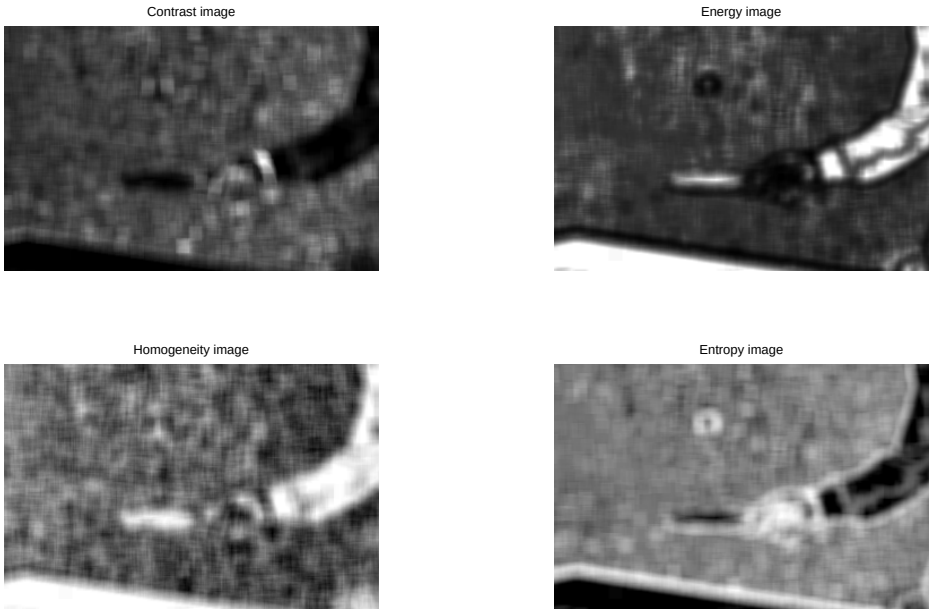


Figure 32. Window size of 13 pixels for pingpong2.tif with execution time equal to 100.89 seconds

C. Number of levels variations

In this section, we changed the value of the number of gray level which can be 2(binary) or 8(numeric).

1) Number of gray levels: 2 bits: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 2 bits.
- Orientation: 0 degree.
- Distance: 1 pixel.

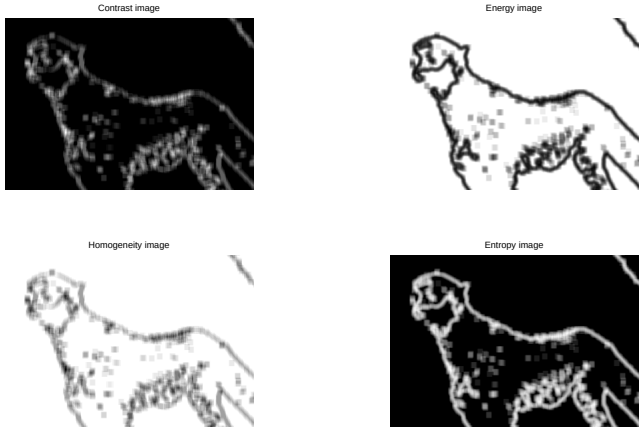


Figure 33. Number of gray levels of 2 bits for feli.tif with execution time equal to 130.67 seconds

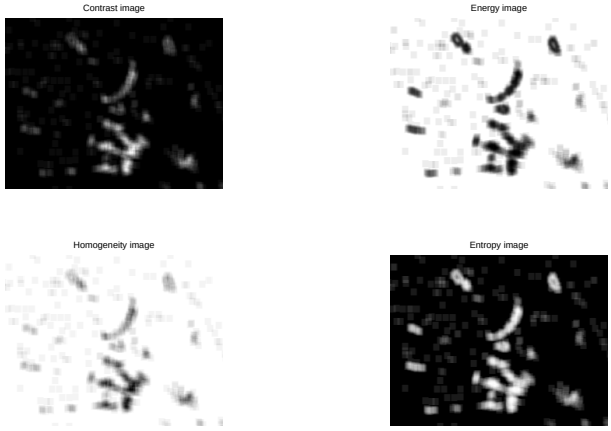


Figure 34. Number of gray levels of 2 bits for hand2.tif with execution time equal to 86.94 seconds

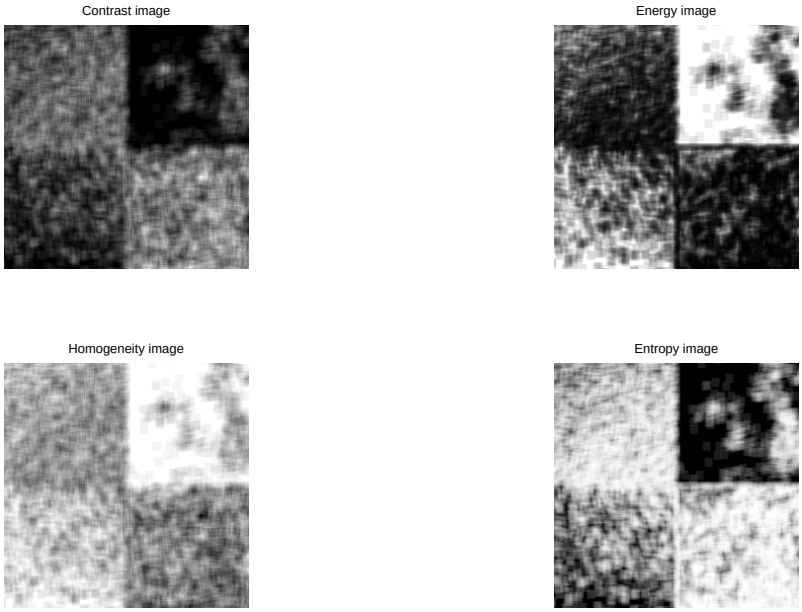


Figure 35. Number of gray levels of 2 bits for mosaic8.tif with execution time equal to 85.09 seconds

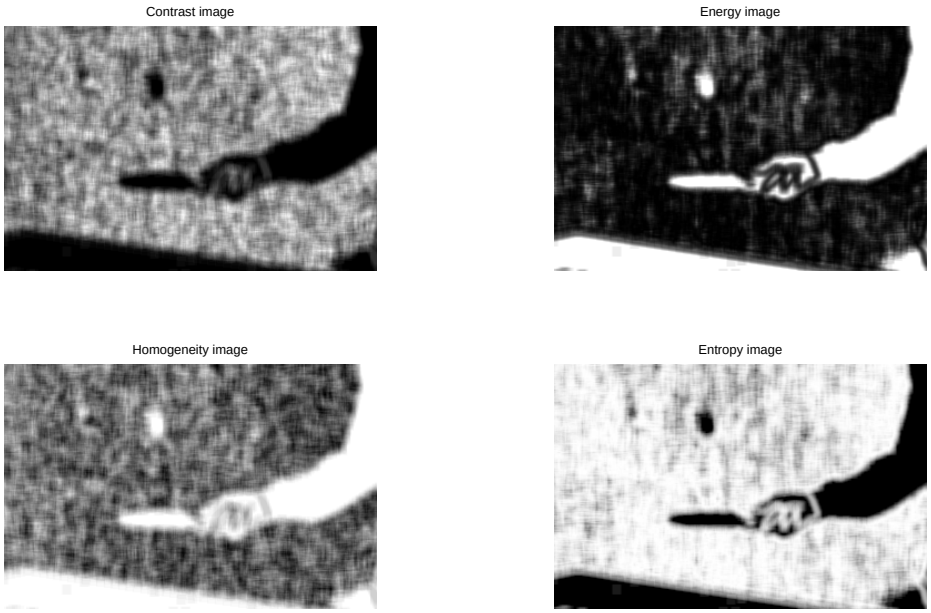


Figure 36. Number of gray levels of 2 bits for pingpong2.tif with execution time equal to 106.75 seconds

D. Distance variations

In this section, we changed the value of the distance.

1) Distance: 2 pixels: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 2 pixels.

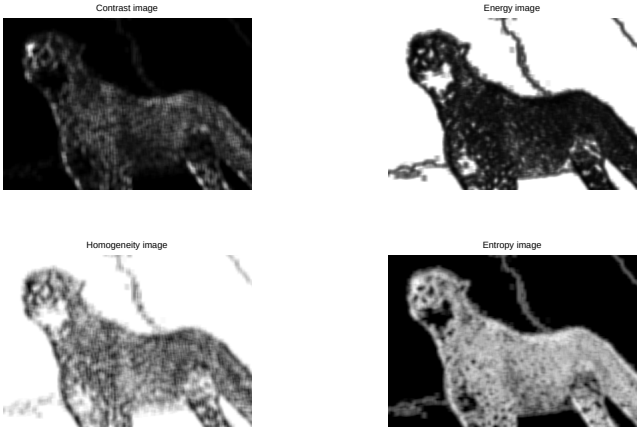


Figure 37. Distance of 2 pixels for feli.tif with execution time equal to 137.67 seconds

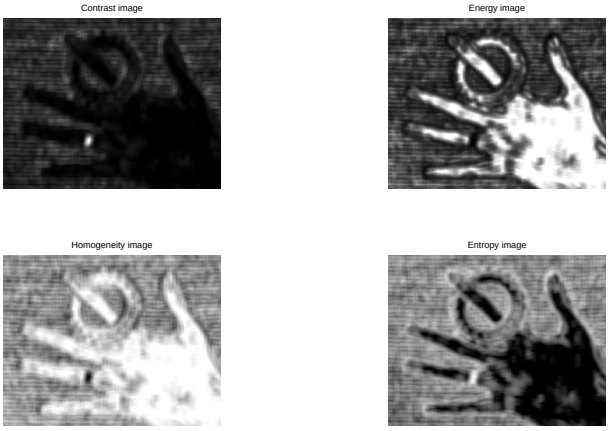


Figure 38. Distance of 2 pixels for hand2.tif with execution time equal to 89.55 seconds

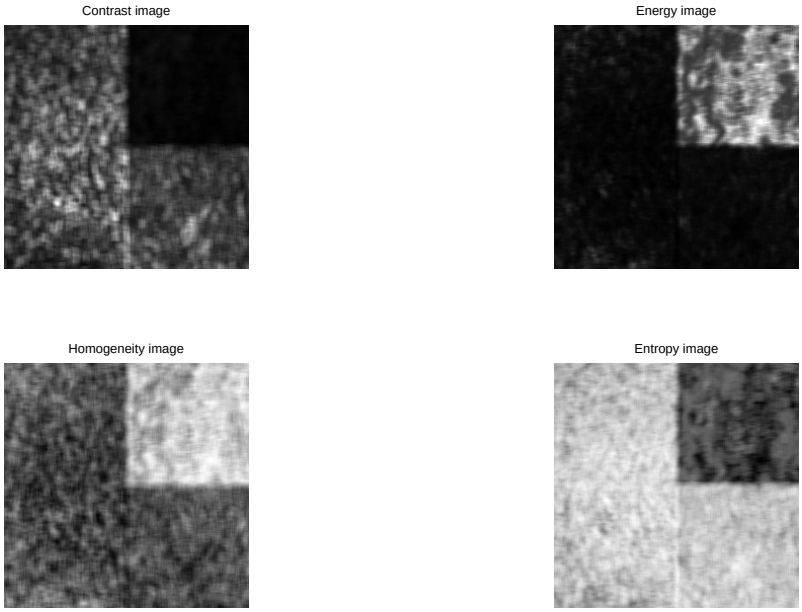


Figure 39. Distance of 2 pixels for mosaic8.tif with execution time equal to 92.60 seconds

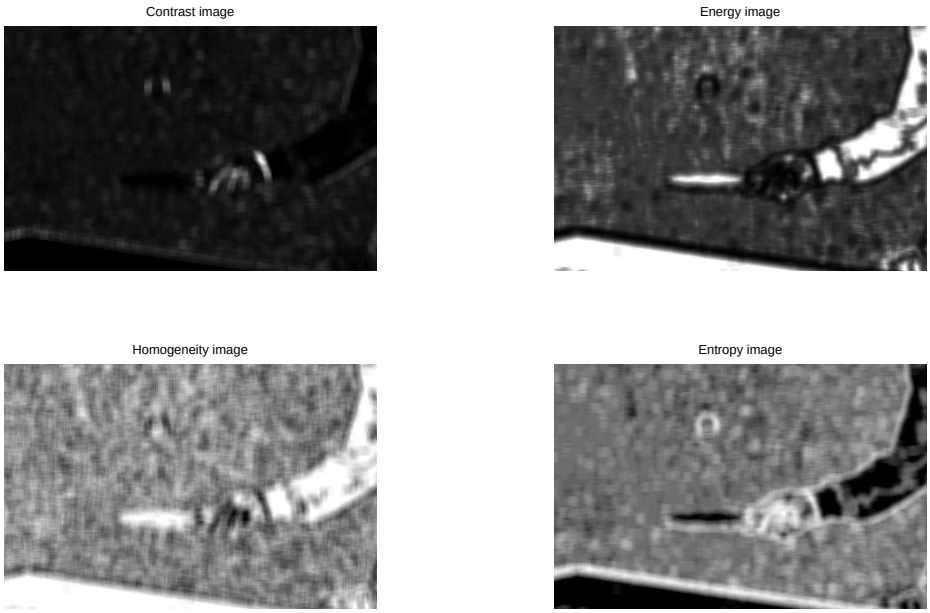


Figure 40. Distance of 2 pixels for pingpong2.tif with execution time equal to 110.17 seconds

2) Distance: 4 pixels: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 0 degree.
- Distance: 4 pixels.

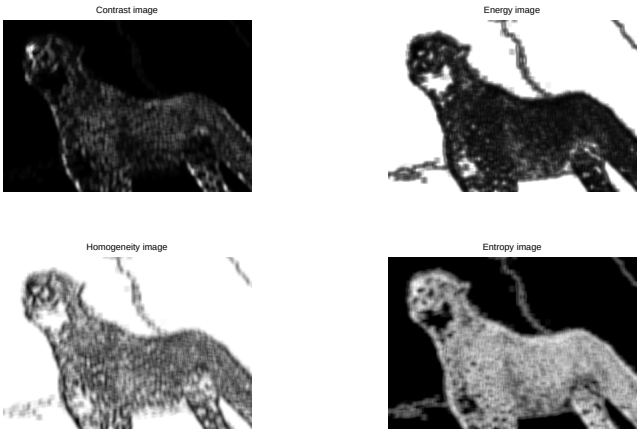


Figure 41. Distance of 4 pixels for feli.tif with execution time equal to 135.67 seconds

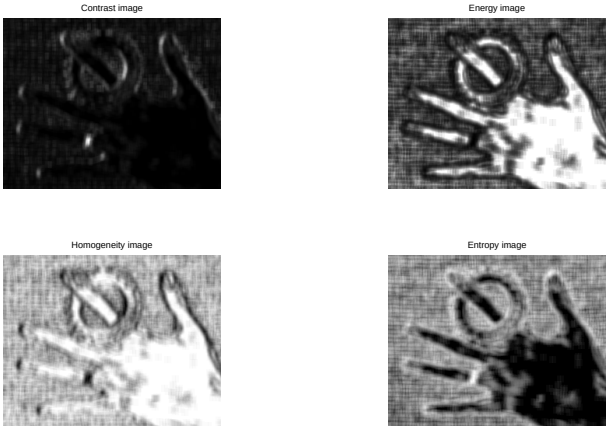


Figure 42. Distance of 4 pixels for hand2.tif with execution time equal to 92.93 seconds

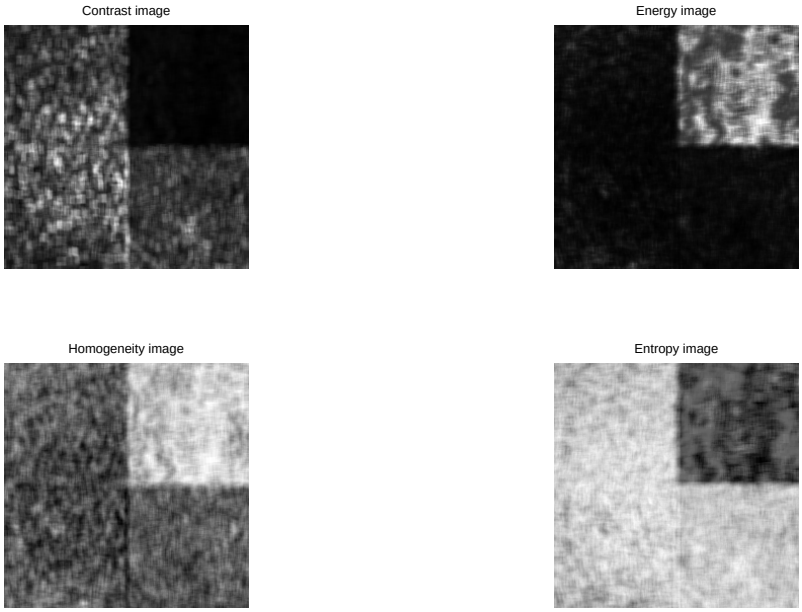


Figure 43. Distance of 4 pixels for mosaic8.tif with execution time equal to 93.38 seconds

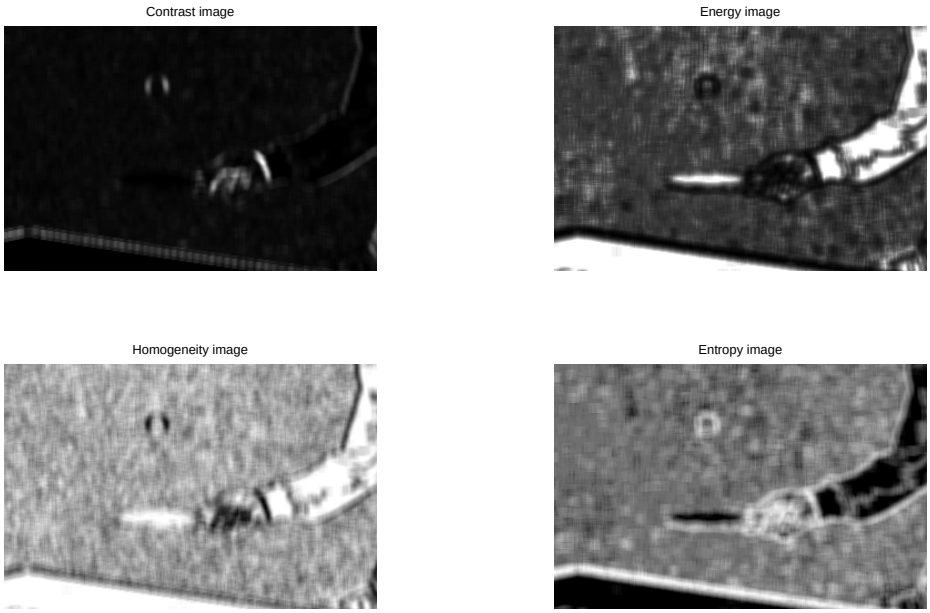


Figure 44. Distance of 4 pixels for pingpong2.tif with execution time equal to 103.88 seconds

E. Orientation variations

In this section, we changed the value of the orientation.

1) Orientation: 45 degrees: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 45 degrees.
- Distance: 1 pixels.

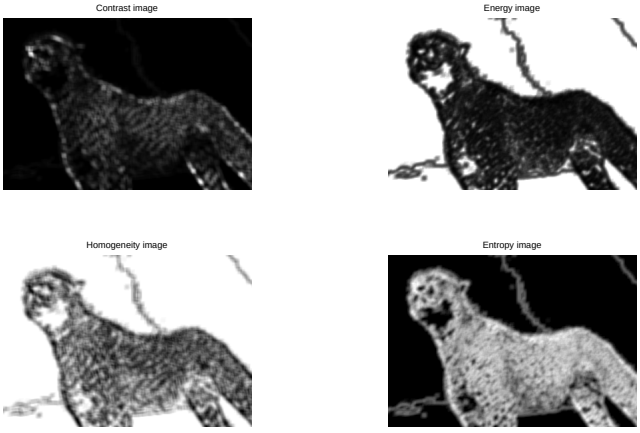


Figure 45. Orientation of 45 degrees for feli.tif with execution time equal to 138.49 seconds

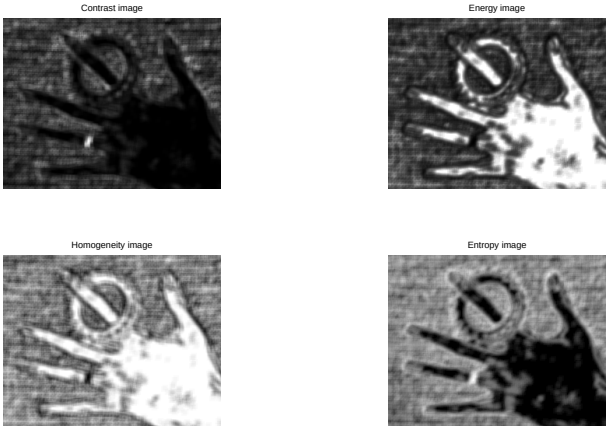


Figure 46. Orientation of 45 degrees for hand2.tif with execution time equal to 88.12 seconds

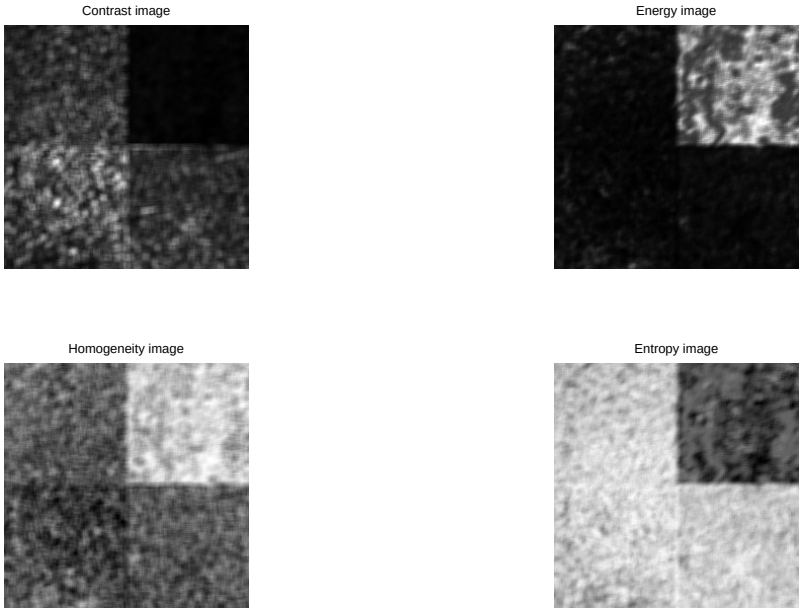


Figure 47. Orientation of 45 degrees for mosaic8.tif with execution time equal to 90.02 seconds

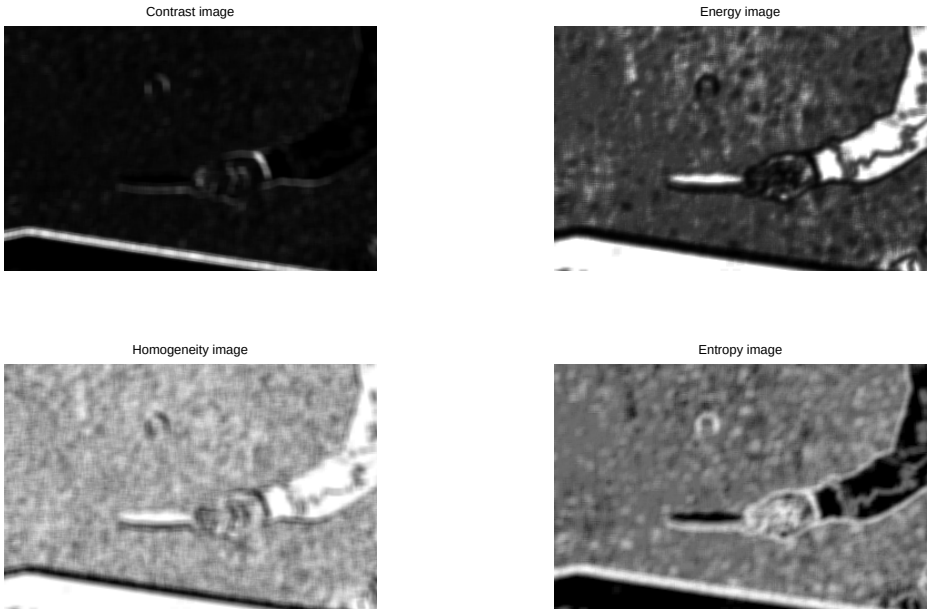


Figure 48. Orientation of 45 degrees for pingpong2.tif with execution time equal to 109.77 seconds

2) Orientation: 90 degrees: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 90 degrees.
- Distance: 1 pixels.

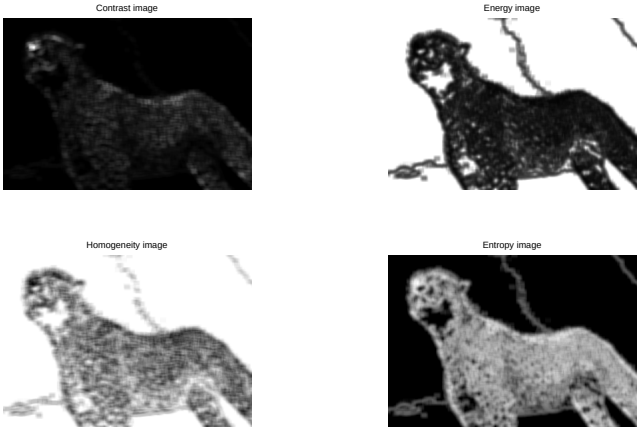


Figure 49. Orientation of 90 degrees for feli.tif with execution time equal to 138.02 seconds

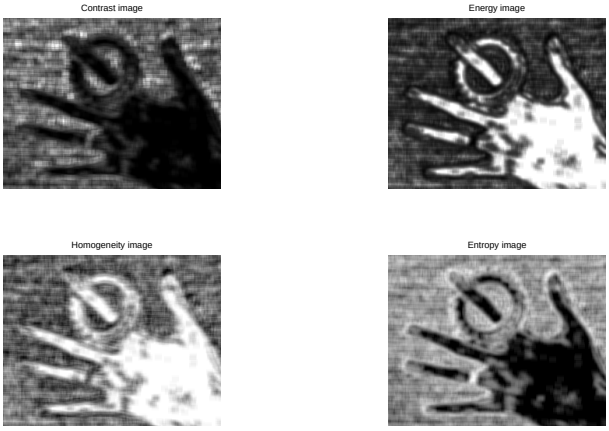


Figure 50. Orientation of 90 degrees for hand2.tif with execution time equal to 90.19 seconds

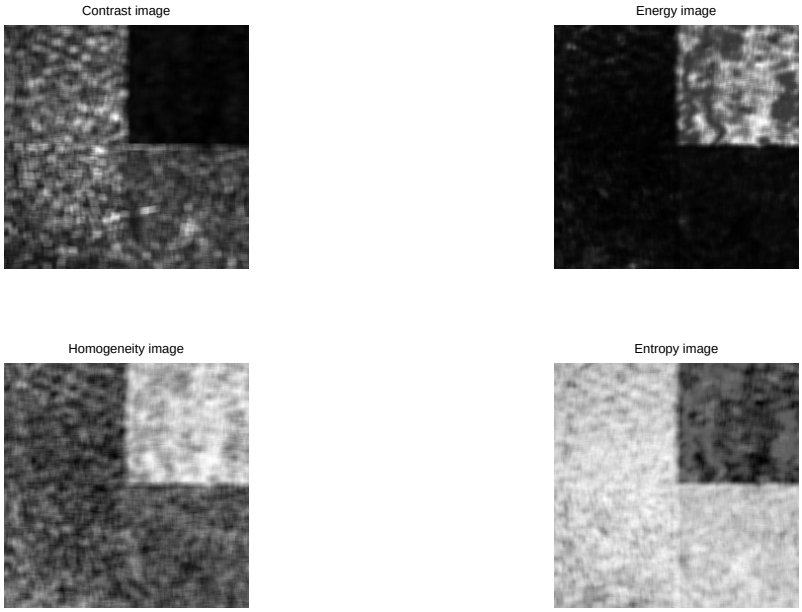


Figure 51. Orientation of 90 degrees for mosaic8.tif with execution time equal to 97.01 seconds

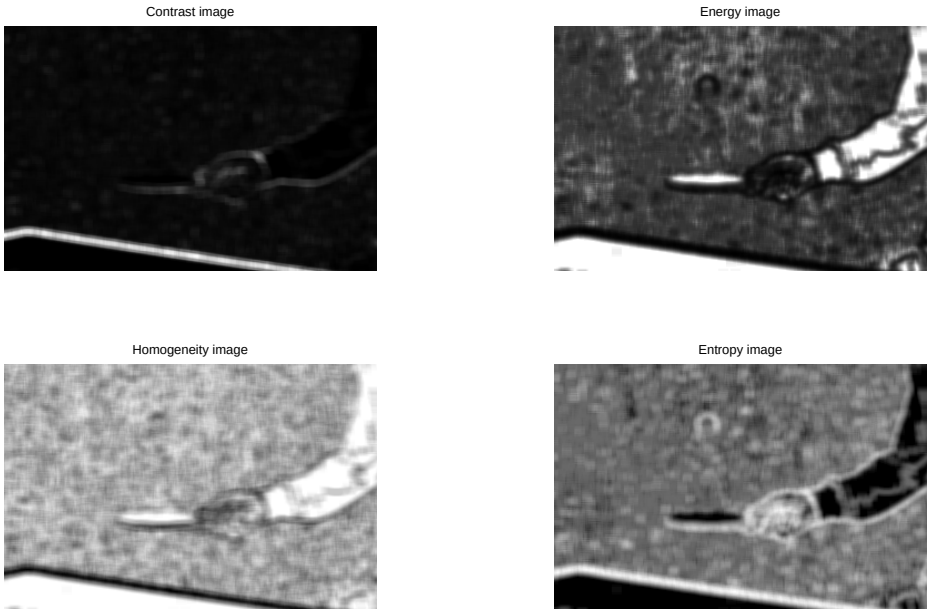


Figure 52. Orientation of 90 degrees for pingpong2.tif with execution time equal to 104.96 seconds

3) Orientation: 135 degrees: In this section, the set of parameters was:

- Window size: 9 pixels
- Number of gray levels: 8 bits.
- Orientation: 135 degrees.
- Distance: 1 pixels.

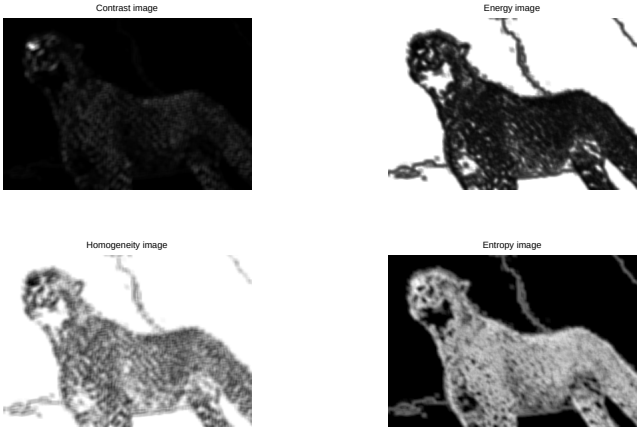


Figure 53. Orientation of 135 degrees for feli.tif with execution time equal to 138.16 seconds

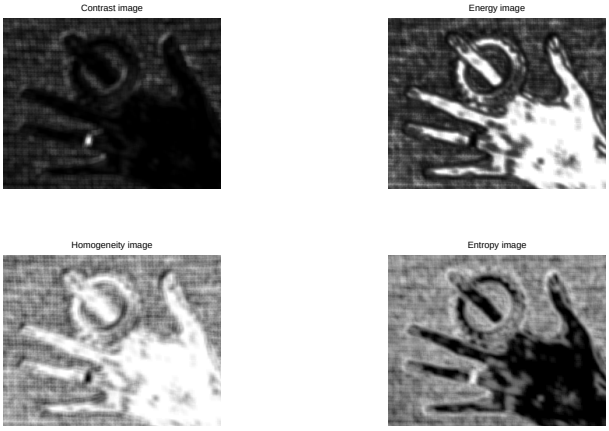


Figure 54. Orientation of 135 degrees for hand2.tif with execution time equal to 98.05 seconds

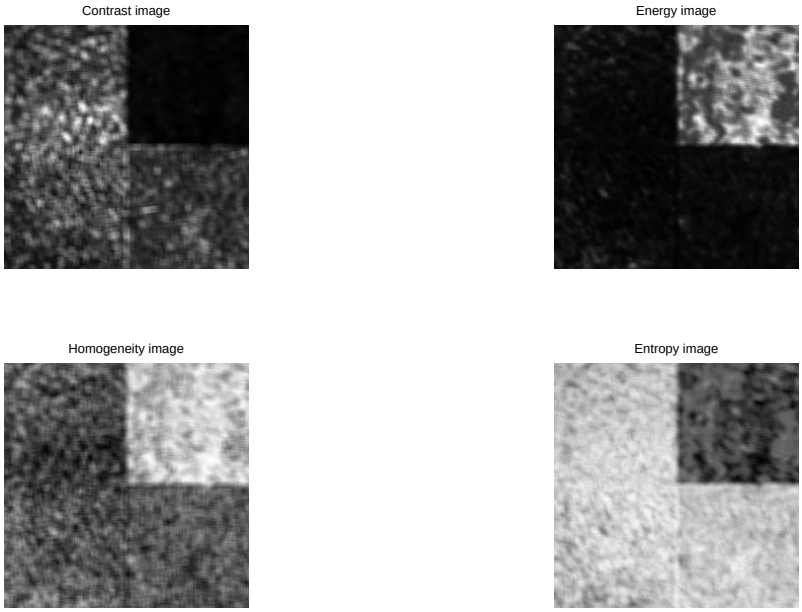


Figure 55. Orientation of 135 degrees for mosaic8.tif with execution time equal to 89.60 seconds

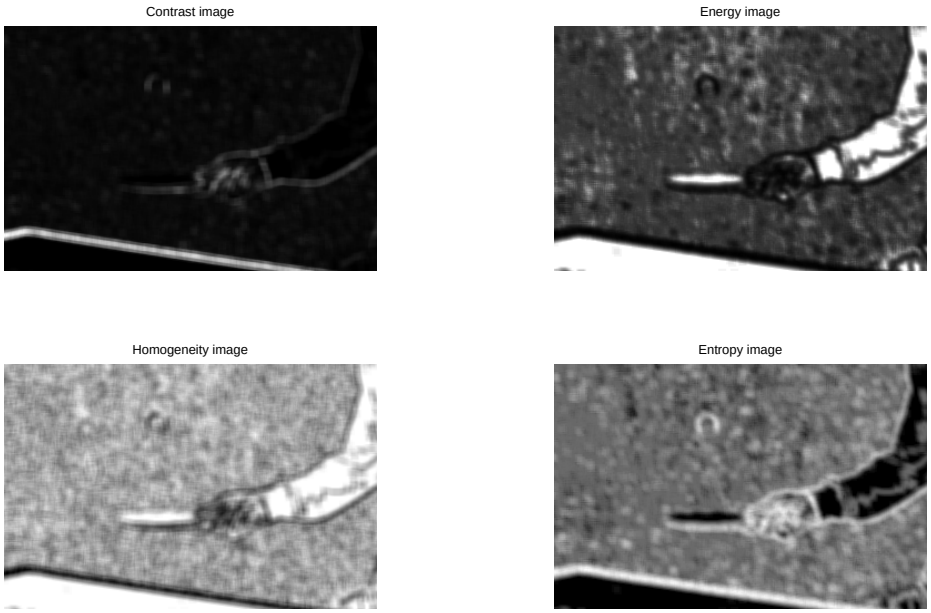


Figure 56. Orientation of 135 degrees for pingpong2.tif with execution time equal to 108.96 seconds

F. Laws' Texture Measures

1) Main function Law_mask():

```
function [image_out] = Law_mask(file_name, filter_type, window_size, statistic_type, normalization_type)
if filter_type(2)~=filter_type(4),
    error('wrong filter type given');
end
if filter_type(2)=='3',
    L3 = [1 2 1]; E3 = [-1 0 1]; S3 = [-1 2 -1]; L3L3mask = L3' * L3;
    switch filter_type
        case 'L3L3'
            filter_mask = L3' * L3;
        case 'E3E3'
            filter_mask = E3' * E3;
        case 'S3S3'
            filter_mask = S3' * S3;
        case 'E3L3'
            filter_mask = E3' * L3;
        case 'L3E3'
            filter_mask = L3' * E3;
        case 'S3L3'
            filter_mask = S3' * L3;
        case 'L3S3'
            filter_mask = L3' * S3;
        case 'S3E3'
            filter_mask = S3' * E3;
        case 'E3S3'
            filter_mask = E3' * S3;
        otherwise
            error('wrong filter type given');
    end
elseif filter_type(2)=='5',
    L5 = [1 4 6 4 1]; E5 = [-1 -2 0 2 1]; S5 = [-1 0 2 0 -1];
    W5 = [-1 2 0 -2 1]; R5 = [1 -4 6 -4 1]; L5L5mask = L5' * L5;
    switch filter_type
        case 'L5L5'
            filter_mask = L5' * L5;
        case 'E5E5'
            filter_mask = E5' * E5;
        case 'S5S5'
            filter_mask = S5' * S5;
        case 'W5W5'
            filter_mask = W5' * W5;
        case 'R5R5'
            filter_mask = R5' * R5;
        case 'L5E5'
            filter_mask = L5' * E5;
        case 'E5L5'
            filter_mask = E5' * L5;
```

```
        case 'L5S5'
            filter_mask = L5' * S5;
        case 'S5L5'
            filter_mask = S5' * L5;
        case 'L5W5'
            filter_mask = L5' * W5;
        case 'W5L5'
            filter_mask = W5' * L5;
        case 'L5R5'
            filter_mask = L5' * R5;
        case 'R5L5'
            filter_mask = R5' * L5;
        case 'E5S5'
            filter_mask = E5' * S5;
        case 'S5E5'
            filter_mask = S5' * E5;
        case 'E5W5'
            filter_mask = E5' * W5;
        case 'W5E5'
            filter_mask = W5' * E5;
        case 'E5R5'
            filter_mask = E5' * R5;
        case 'R5E5'
            filter_mask = R5' * E5;
        case 'S5W5'
            filter_mask = S5' * W5;
        case 'W5S5'
            filter_mask = W5' * S5;
        case 'S5R5'
            filter_mask = S5' * R5;
        case 'R5S5'
            filter_mask = R5' * S5;
        case 'W5R5'
            filter_mask = W5' * R5;
        case 'R5W5'
            filter_mask = R5' * W5;
        otherwise
            error('wrong filter type given');
    end
else
    error('wrong filter type given');
end
image = im2double(rgb2gray(imread(file_name)));
% % STEP 1 mask convolution
 disp(file_name);
 disp('Mask convolution -> in progress...');
 tic
 image_conv = conv2(image,filter_mask,'valid');
 disp('Mask convolution -> done.');
% % STEP 2 statistic computation
av_filter = fspecial('average', [window_size window_size]);
switch statistic_type
    case 'MEAN'
        disp('Statistic computation (mean) -> in progress...');
        image_conv_TEM = conv2(image_conv,av_filter,'valid');
        disp('Statistic computation (mean) -> done.');
    case 'ABSM'
        disp('Statistic computation (abs mean) -> in progress...');
        image_conv_TEM = conv2(abs(image_conv),av_filter,'valid');
        disp('Statistic computation (abs mean) -> done.');
    case 'STDD'
        disp('Statistic computation (st deviation) -> in progress...');
        image_conv_TEM = image_stat_stand_dev(image_conv, window_size);
        disp('Statistic computation (st deviation) -> done...');
    otherwise
        error('wrong statistic type given');
end
switch normalization_type
    case 'MINMAX'
        image_out = normalize(image_conv_TEM);
    case 'FORCON'
        if filter_type(2)=='3',
            image_conv_norm = conv2(image,L3L3mask,'valid');
        else
            image_conv_norm = conv2(image,L5L5mask,'valid');
        end
        switch statistic_type
            case 'MEAN'
                image_norm = conv2(image_conv_norm,av_filter,'valid');
            case 'ABSM'
                image_norm = conv2(abs(image_conv_norm),av_filter,'valid');
            case 'STDD'
                image_norm = image_stat_stand_dev(image_conv_norm, window_size);
        end
        image_out = normalize(image_conv_TEM ./ image_norm);
end
t = toc
 % % present results, normalize before plotting
 figure,
 imshow(image_out); title(['filter type: ' filter_type '  statistic type: ' statistic_type '  norm type: ' normalization_type '  elapsed:' num2str(t)]);
% %%%%%% FUNCTIONS USED %%%%%%%
    function output_image = normalize(input_image)
    MIN = min(min(input_image)); MAX = max(max(input_image));
    output_image = (input_image - MIN) / (MAX - MIN);
    end

    function stand_dev = image_stat_stand_dev(im_in, window_size)
    [im_height im_width] = size(im_in);
    i = 1;
    for row = (window_size+1)/2 : im_height-(window_size-1)/2,
        j = 1;
        for column = (window_size+1)/2 : im_width-((window_size-1)/2),
```

```
            shift = (window_size-1)/2;
            buffer = im_in(row-shift : row+shift, column-shift : column+shift);
            stand_dev(i, j) = std(buffer(1:end));
            j = j + 1;
        end
        i = i + 1;
    end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

2) Program for filtering one particular image:

```
image = 'feli.tif';
image_out = Law_mask(image, 'L3S3', 5, 'ABSM', 'MINMAX');
```

3) Main program for complete filter analysis for an example image:

```
close all; clear all; clc;
%complete filter analysis for an image
load feli.mat
image = 'feli.tif';  window_size = 15;
filter_types_3x3 = ['L3L3'; 'L3E3'; 'L3S3';...
                    'E3L3'; 'E3E3'; 'E3S3';...
                    'S3L3'; 'S3E3'; 'S3S3'];
filter_types_5x5 = ['L5L5'; 'L5E5'; 'L5S5'; 'L5W5'; 'L5R5';...
                    'E5L5'; 'E5E5'; 'E5S5'; 'E5W5'; 'E5R5';...
                    'S5L5'; 'S5E5'; 'S5S5'; 'S5W5'; 'S5R5';...
                    'W5L5'; 'W5E5'; 'W5S5'; 'W5W5'; 'W5R5';...
                    'R5L5'; 'R5E5'; 'R5S5'; 'R5W5'; 'R5R5'];
statistic_types = ['MEAN'; 'ABSM'; 'STDD'];
normalizing_types = ['MINMAX'; 'FORCON'];
normalizing_type = normalizing_types(1, :);
statistic_type   = statistic_types(1, :); % MEAN
for i = 1 : 9,
    temporary = Law_mask(image, filter_types_3x3(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_3x3(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(1); subplot(3,3,i); imshow(temporary); title([filter_types_3x3(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f1 -depsc feli_filter3_w5_mean
for i = 1 : 25,
    temporary = Law_mask(image, filter_types_5x5(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_5x5(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(2); subplot(5,5,i); imshow(temporary); title([filter_types_5x5(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f2 -depsc feli_filter5_w5_mean
statistic_type   = statistic_types(2, :); % ABSM
for i = 1 : 9,
    temporary = Law_mask(image, filter_types_3x3(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_3x3(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(3); subplot(3,3,i); imshow(temporary); title([filter_types_3x3(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f3 -depsc feli_filter3_w5_absm
for i = 1 : 25,
    temporary = Law_mask(image, filter_types_5x5(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_5x5(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(4); subplot(5,5,i); imshow(temporary); title([filter_types_5x5(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f4 -depsc feli_filter5_w5_absm
statistic_type   = statistic_types(3, :); % STDD
for i = 1 : 9,
    temporary = Law_mask(image, filter_types_3x3(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_3x3(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(5); subplot(3,3,i); imshow(temporary); title([filter_types_3x3(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f5 -depsc feli_filter3_w5_stdd
for i = 1 : 25,
    temporary = Law_mask(image, filter_types_5x5(i,:), window_size, statistic_type, normalizing_type); close;
    feli.(genvarname([filter_types_5x5(i, :) '_' num2str(window_size) '_' statistic_type '_' normalizing_type])) = temporary;
    figure(6); subplot(5,5,i); imshow(temporary); title([filter_types_5x5(i,:) ' , ' statistic_type ' , ' normalizing_type]);
end
print -f6 -depsc feli_filter5_w5_stdd
save('feli.mat', 'feli');
```

REFERENCES

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[2] K. Laws, "Textured image segmentation," Ph.D. dissertation, University of Southern California, 1980.

[3] ——, "Rapid texture identification," *SPIE*, vol. 238, Image Processing for Missile Guidance, pp. 376–380, 1980.