IN5520/IN9520 Mandatory term project 2018 – Part I
Segmentation of textured regions in an image

Francisco Jose Nava Lujan
User name: francijn

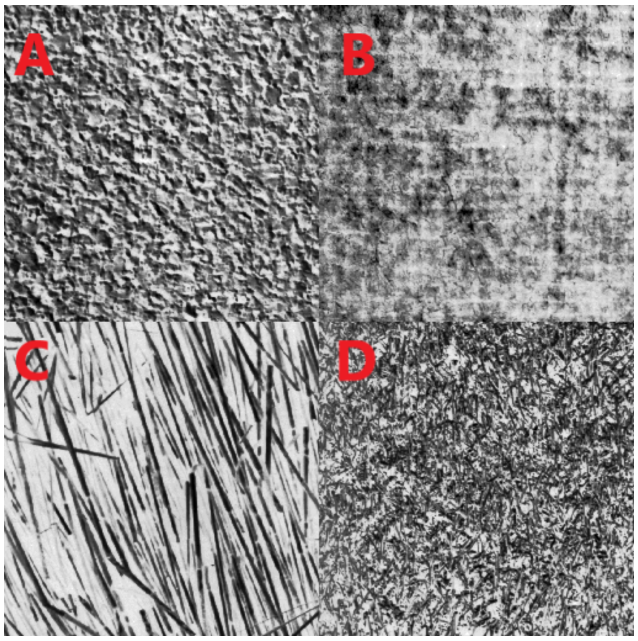
8th October 2018

Abstract

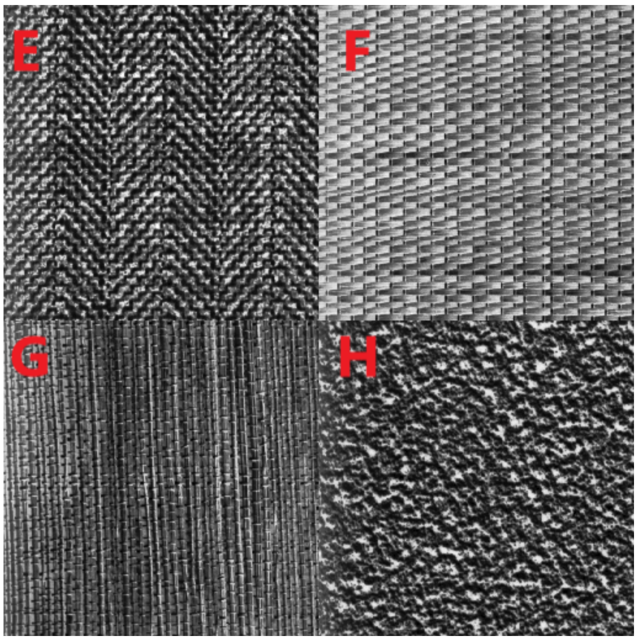
In this mandatory exercise, we will describe textured regions in an image, compute and visualize GLCMs from each texture, extract GLCM feature images, and segment these images.

1 A. Analyzing the textures.

For this step, the mandatory assignment requests for a description of each texture; what can be seen that characterizes each texture visually. In this step, it is helpful to look at the textures we have to work with, so further analysis can be done in a more tailored manner: for this case, when choosing the parameters for the GLCM. The two mosaic pictures are shown in Figure 1.



(a) mosaic1.png



(b) mosaic2.png

Figure 1: Mosaic pictures.

As we can see in the pictures above, textures ***A, D and H*** tend to have random patterns and it is difficult to talk about a direction of texture. These three textures have a relatively equal size of texture elements, but texture ***D*** seems a little more coarse. Also, these textures have almost the same changes in intensity.
Textures ***B and F*** both have darkened parts in them, but an apparent frequency is detected visually (more so on texture ***F***, in which the direction of the texture seems to be horizontal). They do not have a great deal of contrast, and both seem almost grainy. The change in graylevels isn't exactly uniform, so probably an analysis of inertia will give us interesting values.

Textures ***E and G*** both have a way more evident texture frequency and direction. In the case of texture ***E***, we can see that there are lines in 45 ° and 135 ° in a vertical arrangement and, even though the patterns are not completely uniform throughout the image, it is quite evident. In the case of texture ***G***, the most evident characteristic is the vertical lines distributed in an almost uniform frequency, with a very similar width size. There's not that much change in intensity through the image, but this characteristics will be helpful for further classification.
Finally, texture ***C*** has a greater coarseness and contrast to it. It seems like almost vertical lines scattered randomly, without following a pattern or an exact direction.

2 B. Visualizing GLCM matrices.

First of all, it is necessary to divide the four textures in each mosaic picture. The following Matlab code was used to do so:

```
M1 = imread('mosaic1.png');
M2 = imread('mosaic2.png');

M1I1 = M1(1:end/2,1:end/2);
M1I2 = M1(1:end/2,end/2:end);
M1I3 = M1(end/2:end,1:end/2);
M1I4 = M1(end/2:end,end/2:end);

M2I1 = M2(1:end/2,1:end/2);
M2I2 = M2(1:end/2,end/2:end);
M2I3 = M2(end/2:end,1:end/2);
M2I4 = M2(end/2:end,end/2:end);
```

After doing this, the GLCM is computed for each texture. In this step, the GLCM is not computed by using a sliding window yet, but for the whole texture. The way the GLCM matrix is calculated was done from scratch, based on my own understanding of the functionality of GLCM. It is very important to say that, for this particular project, a thorough function (or rather, 2 functions, if the normalization is considered) was coded for different values of ***d*** and θ. The GLCM function is attached along with this document, but for the sake of explanation, the isometric calculation is added in the next frame:

```
if theta == 0
    for row = 1:size(x,1)
        for col = 1:size(x,2)-d
            r1(auxI(row,col),auxI(row,col+d)) = r1(auxI(row,col),auxI(row,col+d)) + 1;
        end
    end
elseif theta == 45
    for row = d+1:size(x,1)
        for col = 1:size(x,2)-d
            r1(auxI(row,col),auxI(row-d,col+d)) = r1(auxI(row,col),auxI(row-d,col+d)) + 1;
        end
    end
elseif theta == 90
    for row = d+1:size(x,1)
        for col = 1:size(x,2)
            r1(auxI(row,col),auxI(row-d,col)) = r1(auxI(row,col),auxI(row-d,col)) + 1;
        end
    end
elseif theta == 135
    for row = d+1:size(x,1)
        for col = d+1:size(x,2)
```

```
                r1(auxI(row,col),auxI(row-d,col-d)) = r1(auxI(row,col),auxI(row-d,col-d)) + 1;
            end
        end
elseif theta == 150 % ISOMETRIC calculation
        for row = d+1:size(x,1)-d
            for col = d+1:size(x,2)-d
                % theta == 0
                r1(auxI(row,col),auxI(row,col+d)) = r1(auxI(row,col),auxI(row,col+d)) + 1;
                % theta == 45
                r1(auxI(row,col),auxI(row-d,col+d)) = r1(auxI(row,col),auxI(row-d,col+d)) + 1;
                % theta == 90
                r1(auxI(row,col),auxI(row-d,col)) = r1(auxI(row,col),auxI(row-d,col)) + 1;
                % theta == 135
                r1(auxI(row,col),auxI(row-d,col-d)) = r1(auxI(row,col),auxI(row-d,col-d)) + 1;
            end
        end
end
% Make symmetric
r1 = r1 + r1.';
% Return quantized image
```

The function *my_glcm* receives 4 parameters, which are: image, gray-levels, d and θ
As we can see, there are 5 possible values of θ: 0, 45, 90, 135 and 150. To calculate the isometric GLCM matrix, the value 150 (ISO) is chosen. The value of ***d*** can be anything less than ceil(windowSize/2), but this is not validated in my code.
As we can see in the code sample above, the only difference (which is important because it determines the way GLCM normalization is taken care of) between the isometric calculation and all other possibilities is the number of co-occurrences that make it into the GLCM matrix, and when it is made symmetric, it doubles the co-occurrences. Isotropic calculation was used mostly in textures with no visual texture frequency, such as the randomly spread textures ***A, D and H***. In the next frame, normalization is shown:

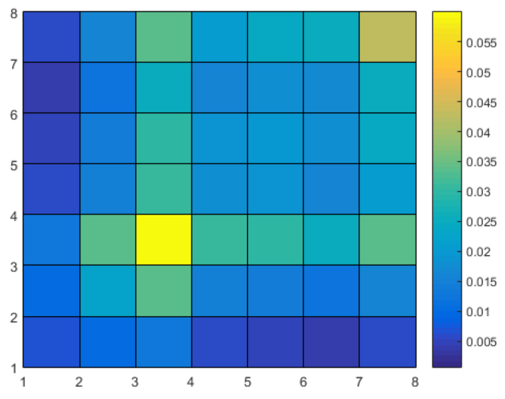
```
function [ newGL ] = my_glcm_norm( GLCM, sow, d, theta )
%   Normalization of GLCM matrix
newGL = GLCM;
if theta == 0 || theta == 90
        for i = 1:numel(GLCM)
            if GLCM(i) ~= 0
                newGL(i) = GLCM(i)/(2*(sow*(sow-d)));
            end
        end
elseif theta == 45 || theta == 135
        for i = 1:numel(GLCM)
            if GLCM(i) ~= 0
                newGL(i) = GLCM(i)/(2*((sow-d)*(sow-d)));
            end
        end
else
        for i = 1:numel(GLCM)
            if GLCM(i) ~= 0
                newGL(i) = GLCM(i)/(8*((sow-d*2)*(sow-d*2)));
            end
        end
end

end
```

The function *my_glcm_norm* receives 4 parameters, which are: GLCM, size of window, d and θ

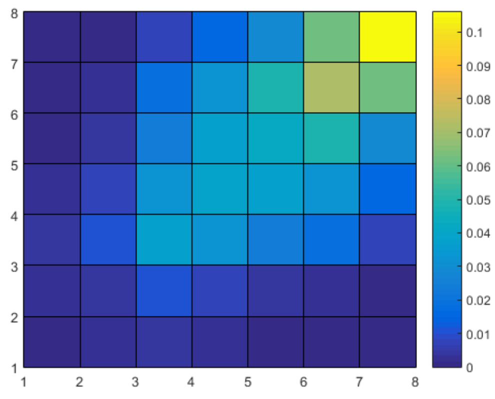
The next step is to calculate the normalized GLCM for the individual textures. Visual analysis was considered in order to choose the right parameters of d and theta for each texture in Figure 1. For example, for texture ***G***, which is composed of mostly vertical arrangements with a certain frequency, the value of θ chosen was 90. This is done using the following code:

```
% GLCM for textures in mosaic1.png
glcmM1I1 = my_glcm(M1I1, 8, 3, 90 );
glcmM1I1norm = my_glcm_norm( glcmM1I1, 256, 3, 90);
glcmM1I2 = my_glcm(M1I2, 8, 3, 150 );
glcmM1I2norm = my_glcm_norm( glcmM1I2, 256, 3, 150);
glcmM1I3 = my_glcm(M1I3, 8, 3, 0 );
glcmM1I3norm = my_glcm_norm( glcmM1I3, 256, 3, 0);
glcmM1I4 = my_glcm(M1I4, 8, 3, 90 );
glcmM1I4norm = my_glcm_norm( glcmM1I4, 256, 3, 90);

% GLCM for textures in mosaic2.png
glcmM2I1 = my_glcm(M2I1, 8, 3, 45 );
glcmM2I1norm = my_glcm_norm( glcmM2I1, 256, 3, 45);
glcmM2I2 = my_glcm(M2I2, 8, 3, 0 );
glcmM2I2norm = my_glcm_norm( glcmM2I2, 256, 3, 0);
glcmM2I3 = my_glcm(M2I3, 8, 3, 90 );
glcmM2I3norm = my_glcm_norm( glcmM2I3, 256, 3, 90);
glcmM2I4 = my_glcm(M2I4, 8, 3, 150 );
glcmM2I4norm = my_glcm_norm( glcmM2I4, 256, 3, 150);
```

In the figures below, each GLCM matrix is shown. Based on the analysis made in point A, and the parameters chosen, it is evident how the choice of both the step size d and orientation theta affect the resulting GLCM. This was done by means of experimentation, using a small set of step sizes d and orientation thetas. We can see, for example, for GLCMs in mosaic1 textures A and D have a very similar GLCM matrix, and the inertia value for both textures is much alike. Texture H, on the other hand, shows high correlation between low gray-levels, therefore, cluster shading may apply to recognize that particular texture. We can use this kind of information to classify each texture.

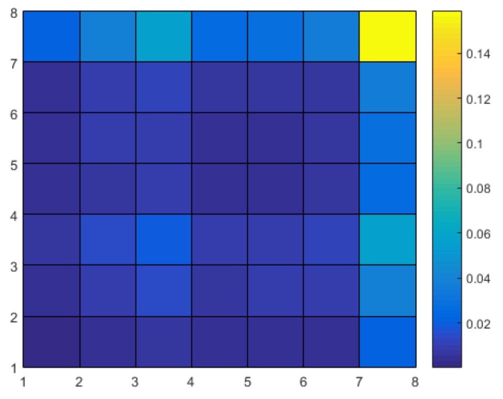


(a) Texture A

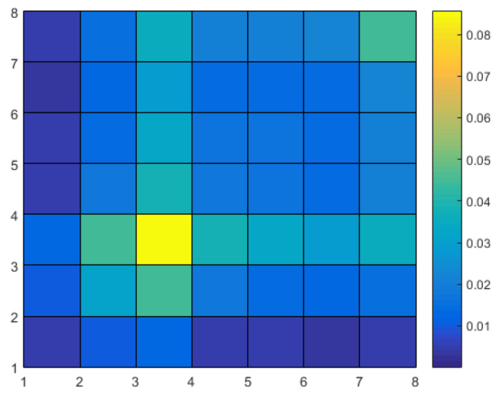


(b) Texture B

Figure 2: GLCM for mosaic1 textures A and B.

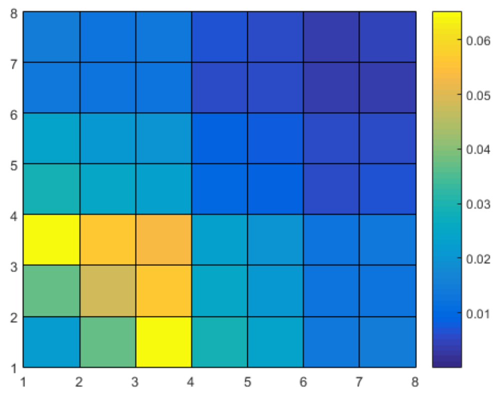


(a) Texture C

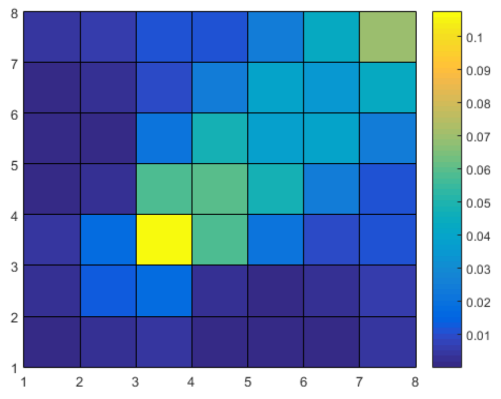


(b) Texture D

Figure 3: GLCM for mosaic1 textures C and D.

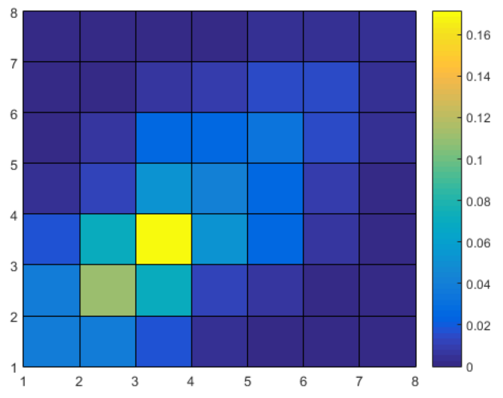


(a) Texture E

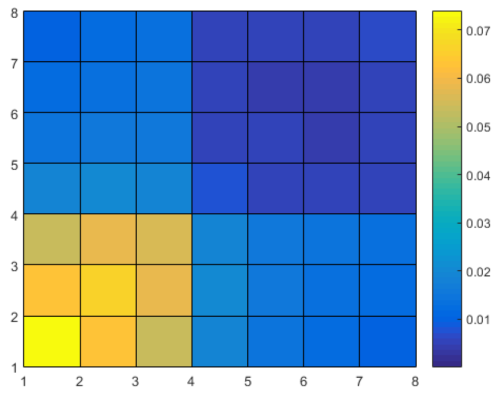


(b) Texture F

Figure 4: GLCM for mosaic2 textures E and F.



(a) Texture G



(b) Texture H

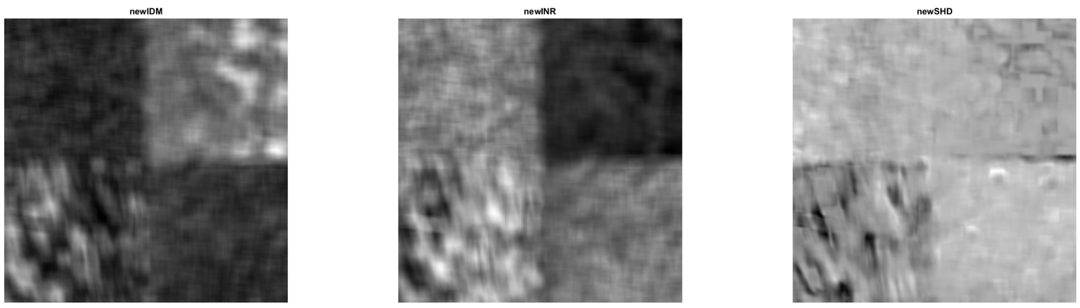
Figure 5: GLCM for mosaic2 textures G and H.

3 C. Computing GLCM feature images in local windows.

For this step, computation of the GLCM of each mosaic image using a sliding window of a fixed size was made. The window size chosen for this particular project was 31x31 pixels. For each computed GLCM within the window, the three features were computed and stored in the corresponding matrices. To do this, a new function called *my_features* was created. This function receives 4 parameters, which are: image, window size, d and theta, and returns three matrices, each containing one of the features [IDM, INR, SHD].
The code implemented in this function is shown in the next frame.

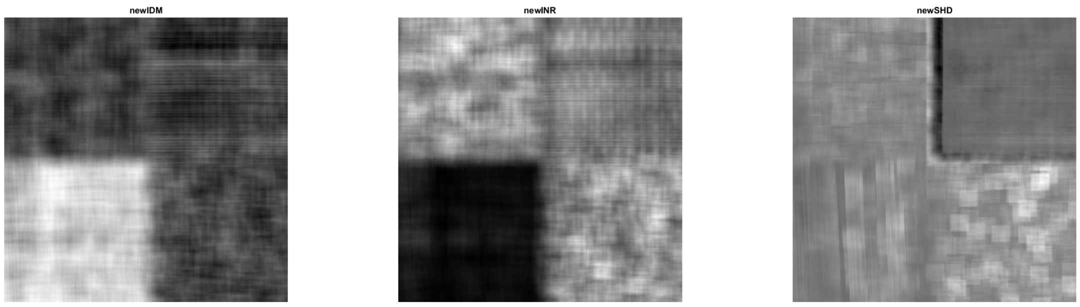
```
function [ IDM, INR, SHD ] = my_features( x, sow, d, theta )

sowC = ceil(sow/2);
sowF = floor(sow/2);
% Initializing feature matrices
IDM = zeros(size(x(sowC:end-sowF, sowC:end-sowF)));
INR = zeros(size(x(sowC:end-sowF, sowC:end-sowF)));
SHD = zeros(size(x(sowC:end-sowF, sowC:end-sowF)));
auxIDM = 0;
auxINR = 0;
auxSHD = 0;

for row = 1:size(IDM,1)
    for col = 1:size(IDM,2)
        newGL = my_glcm(x(row:row+(sow-1),col:col+(sow-1)), 8, d, theta);
        GLnorm = my_glcm_norm(newGL, sow, d, theta);
        [mx, my] = my_means(GLnorm);
        for i = 1:size(GLnorm,2)
            for j = 1:size(GLnorm,1)
                auxIDM = auxIDM + (GLnorm(i,j) / (1 + (i-j)^2));
                auxINR = auxINR + ((i-j)^2 * GLnorm(i,j)); % Contrast
                auxSHD = auxSHD + ((i + j - mx - my)^3 * GLnorm(i,j));
            end
        end
        IDM(row,col) = auxIDM;
        INR(row,col) = auxINR;
        SHD(row,col) = auxSHD;
        auxIDM = 0;
        auxINR = 0;
        auxSHD = 0;
    end
end

end
```

It is shown in the code above that a GLCM matrix is computed for each iteration of a gliding window throughout the whole image, and to take advantage of this cycle, I found it useful to store the three features at the same time. For the cluster shader calculation, a new function called *my_means* was created. This is a generic function and it works for m x n sized matrices, and calculates means in x and y as stated in the textbook (Gonzalez & Woods, 2008, Digital image processing, p. 828). In this particular case, the GLCM matrix will always be a squared symmetric matrix, so the value of mx and my will always be the same.
The calculation of the dimensions of the resulting feature matrices is explained through the simple code above; it was done using the desired size of window and calculating given the size of the whole image with an inner padding. The images below show the result of this feature calculations in images, normalized in an interval of 0, 1 for display purposes.



(a) IDM (b) INR (c) SHD

Figure 6: Features for mosaic1, d = 3, window size=31, isometric direction.



(a) IDM (b) INR (c) SHD

Figure 7: Features for mosaic2, d = 3, window size=31, theta=90.

4 D. Segment the GLCM feature images and describe how they separate the textures

In this final step, a global threshold is computed and used to threshold the mosaic image. To segment each texture (or rather, a specific characteristic of any given texture), all previous analysis was useful. Hence, for this step, the texture orientation, frequency and other features were taken into consideration. The code used for segmentation and classification for one sample texture is shown in the next frame.

```
currFeat = my_threshold(newSHD, 0.33);
imshow(currFeat);
newI = I1(floor(windowSize/2):end-ceil(windowSize/2),
        floor(windowSize/2):end-ceil(windowSize/2));
result = double(newI) .* currFeat;
imshow(result/256);
```

The function *my_threshold* receives two parameters: the feature to threshold and the thresholding level in an interval from 0 to 1. To give an example, for mosaic2.png, we calculate the IDM feature using theta=90 and step size 3, with a window size 31. Since theta=90, the feature will include interesting data assuming a vertical texture direction is being analized. If we then threshold the resulting IDM image, and map it into the original image, we will see that it correctly classifies the texture given said parameters.
In the next few pages, the result for various segmentation results is being shown. The parameters used to calculate and segment the images is described in the image caption.

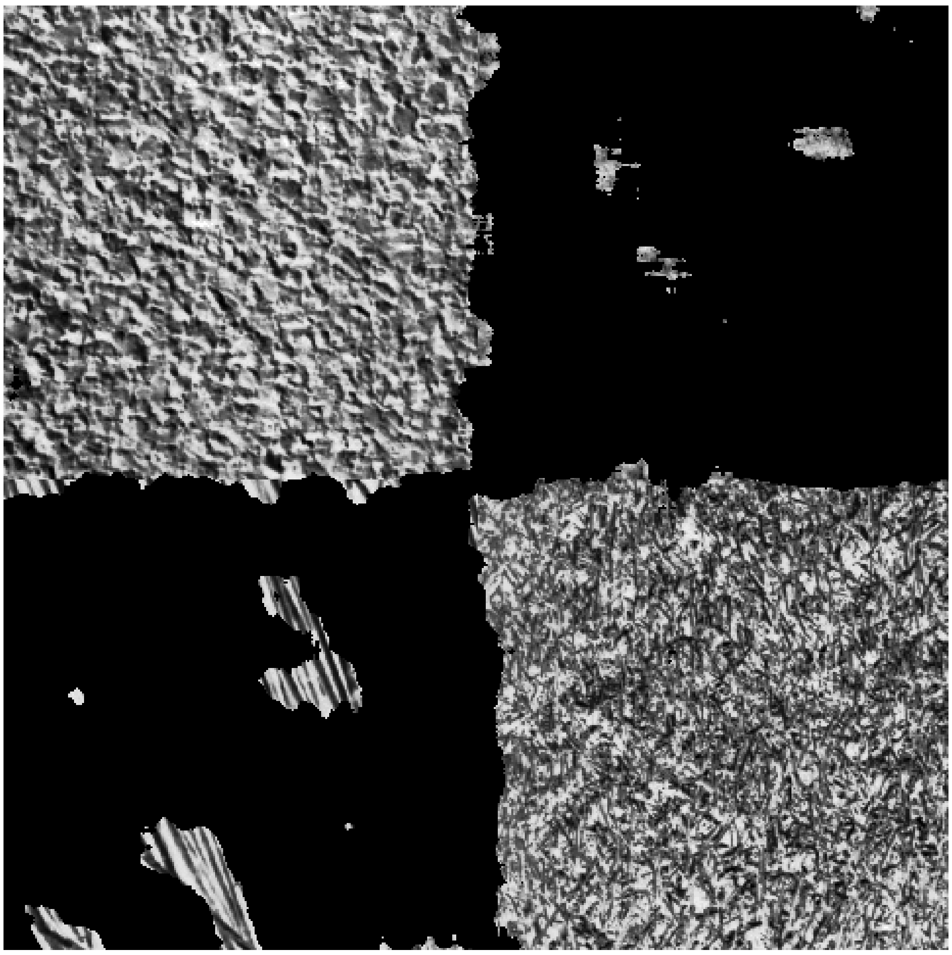


Figure 8: For mosaic1, textures A and D : d = 3, θ=90. Value of threshold=0.3 using IDM.

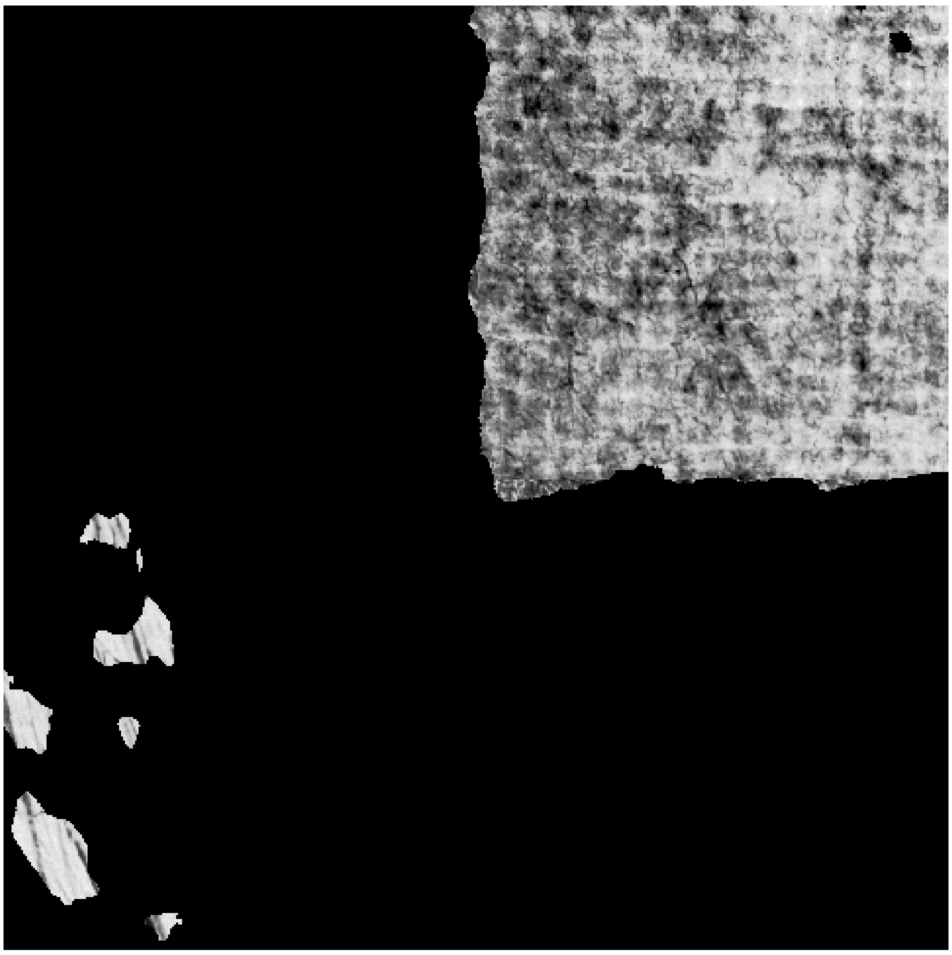


Figure 9: For mosaic1, texture B : d = 3, θ=150 (isometric). Value of threshold=0.27 using INR.

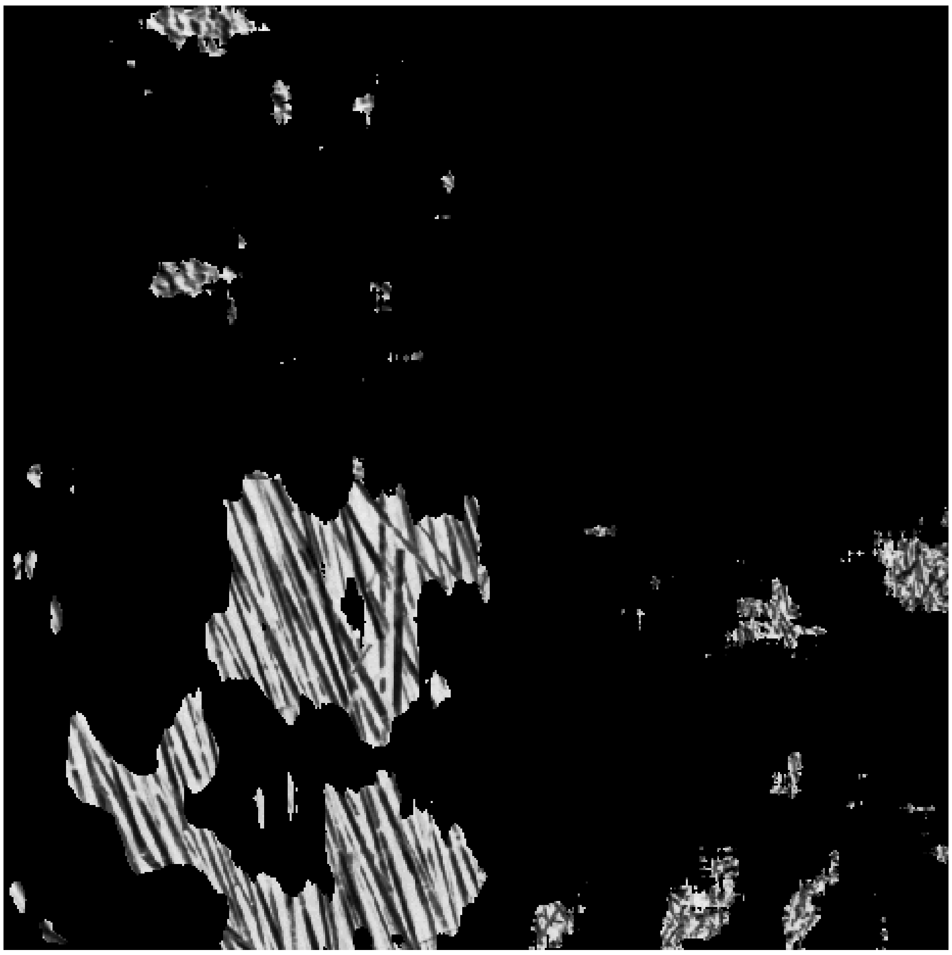
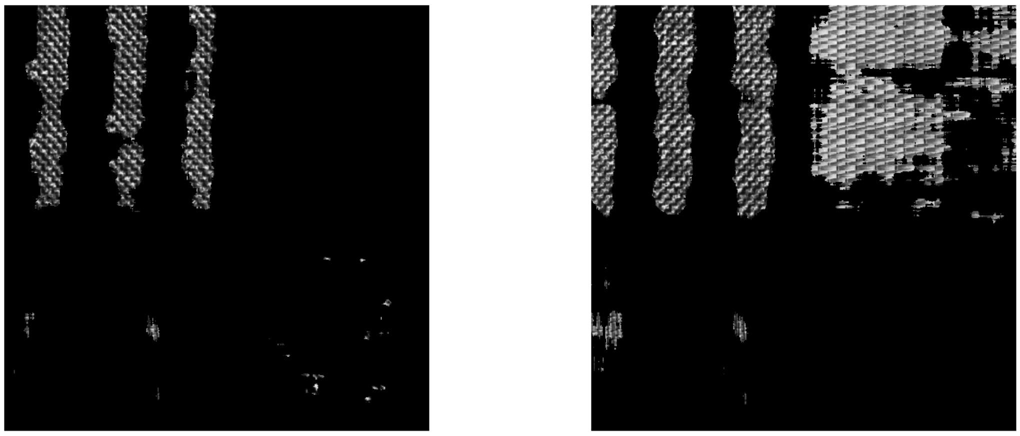


Figure 10: For mosaic1, texture C : d = 3, θ=0. Value of threshold=0.25 using IDM.



(a) θ=45

(b) θ=45

Figure 11: For mosaic2, texture E : d = 3, θ=45 and 135. Value of threshold=0.25 and 0.35 using IDM.

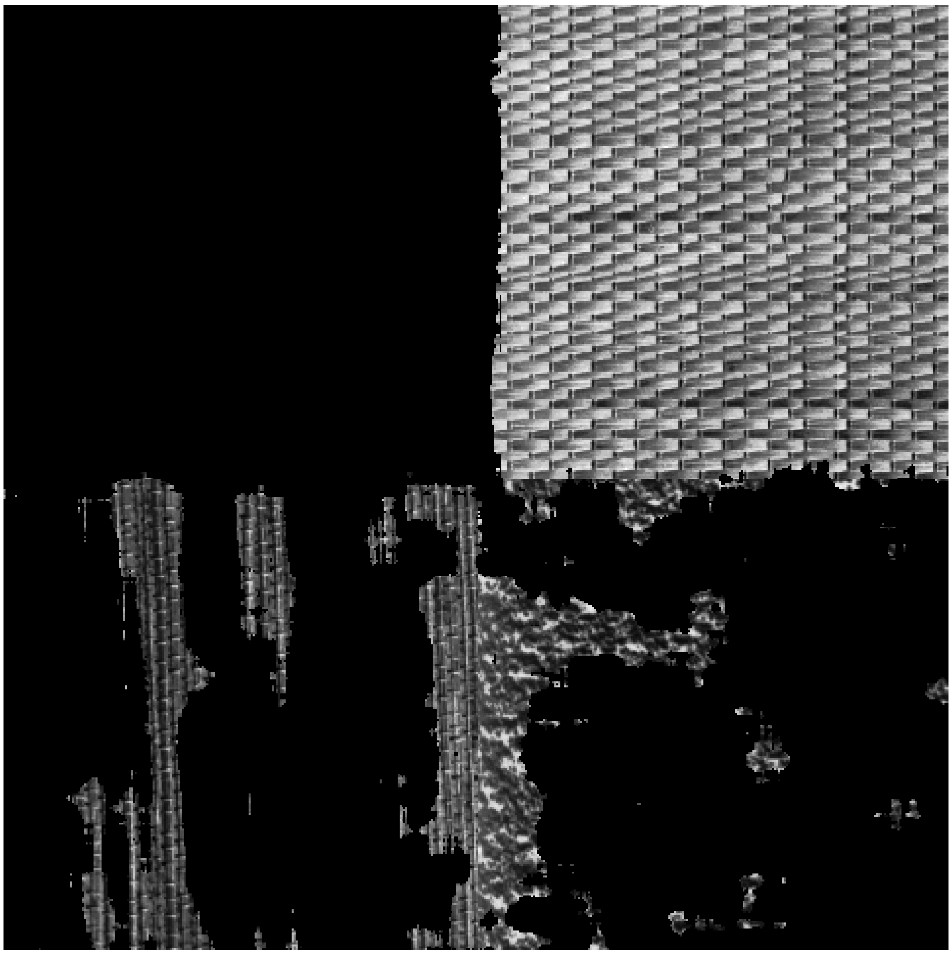


Figure 12: For mosaic2, texture F : d = 3, θ=0. Value of threshold=0.25 using INR.

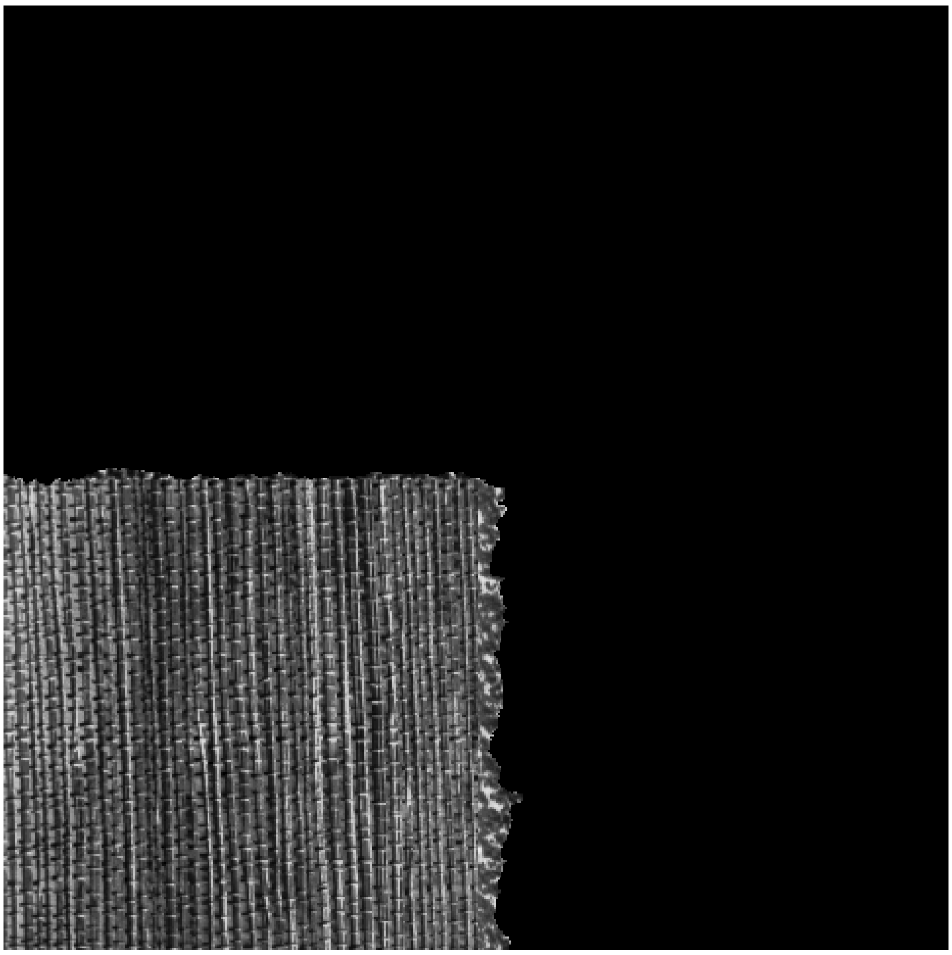


Figure 13: For mosaic2, texture G : d = 3, θ=90. Value of threshold=0.5 using IDM.

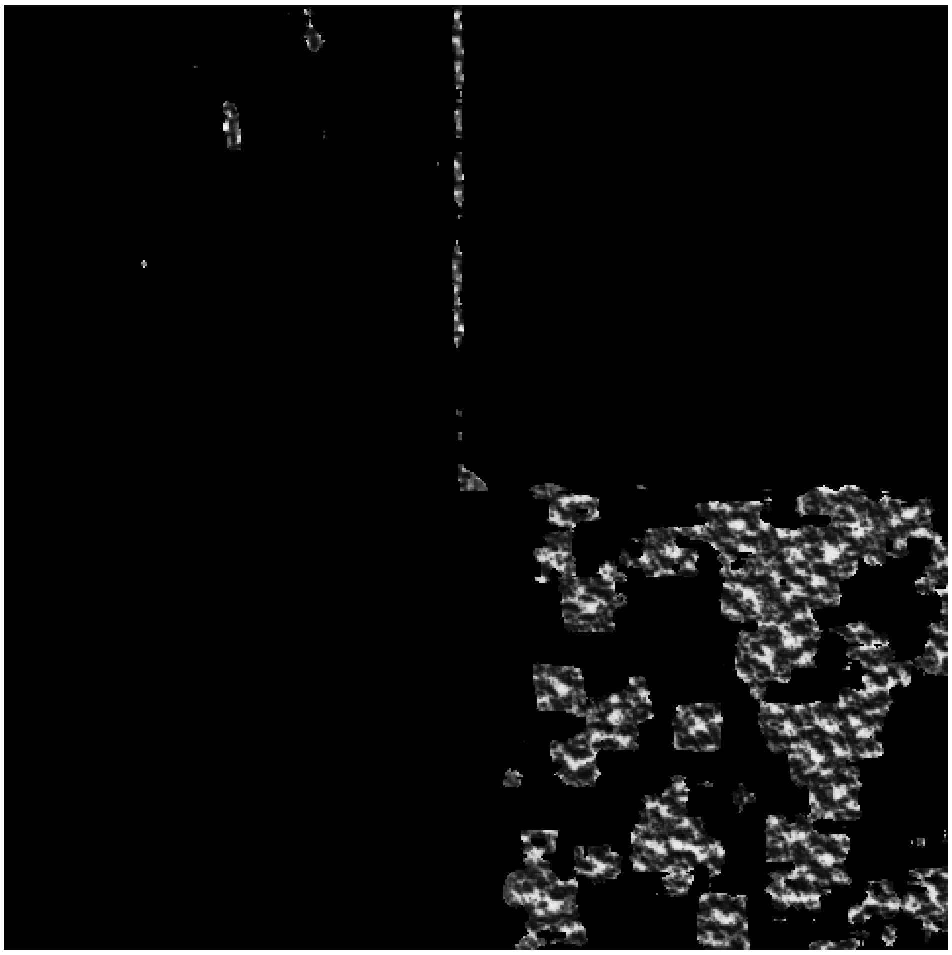


Figure 14: For mosaic2, texture H : d = 3, θ=150 (isometric). Value of threshold=0.6 using SHD.