



# B31SE Image Processing: Assignment 3

Hugo Millet / Timothe Petitjean
16/03/2023
B31SE: Image Processing
MSc. Robotics
Electrical Electronic and Computer Engineering

## ---Problem 1: skeletonizing procedure ---

The skeletonization process is a method used to reduce the body of an object, in order to keep only its “skeleton”: which simplifies the outline of the object.

For this assignment the method is achieved using the following process:

We only focus on pixels with a value of 1 and with at least one neighbour valued 0. However as one of the conditions for deletion is:

$$\text{a) } 2 \leq N(p_1) \leq 6$$

We can directly decide only to focus on pixels satisfying this

We then check the other conditions:

$$\text{b) } T(p_1) = 1$$

$$\text{c) } p_2 \cdot p_4 \cdot p_6 = 0$$

$$\text{d) } p_4 \cdot p_6 \cdot p_8 = 0$$

If They are satisfied, we flag the pixel for deletion, and delete it once the whole picture has been flagged

We then repeat the process and change only conditions c) and d) by:

$$\text{c') } p_2 \cdot p_4 \cdot p_8 = 0$$

$$\text{d') } p_2 \cdot p_6 \cdot p_8 = 0$$

This is repeated until no point is flagged (Hence no point satisfies the conditions) in the image

### Code:

```
f = imread('Fig1108(a)(mapleleaf).tif'); %read the original image
%if Necessary apply these binarization to a coloured image, to obtain a binarized one
%h = fspecial('gaussian',25,15);
%g = imfilter(f,h,'replicate');
%f = imbinarize(g,0.5);
%f = 1-f;
F=f;
[rows, cols] = size(f);
c = 1;
flag = ones(rows,cols); % Create our flag matrice with the same size as the original picture
%flag = zeros(rows,cols); %Antother way

while c~=0
    c=0; % Counter set to know if the flagged matrix is empty or not

    %---STEP 1---
    for x = 2 : rows-1
        for y = 2 : cols-1 % For each inner pixel
            Tp=0; %Reset T(p)
            P(1) = f(x-1,y); %creation of our vector following the 8-neighborhood notation
            P(2) = f(x-1,y+1);
            P(3) = f(x,y+1);
            P(4) = f(x+1,y+1);
            P(5) = f(x+1,y);
            P(6) = f(x+1,y-1);
            P(7) = f(x,y-1);
            P(8) = f(x-1,y-1);
            NP = sum(P); %Calculate the number of neighbors, and N(p)
            if f(x,y)==1 && NP<=6 && NP >=2 %We focus only on flagged point that fulfill 1st condition
                for i=1:7
                    if P(i)==0 && P(i+1)==1
                        Tp=Tp+1; %Calculate T(p)
                    end
                end
                if P(8)==0 && P(1) ==1
                    Tp=Tp+1;
                end
                Pa = P(1)*P(3)*P(5); %We calculate our last conditions
                Pb = P(3)*P(5)*P(7);
                if Tp ==1 && Pa ==0 && Pb ==0 %We verify if the flagged point fulfill all left conditions
                    flag(x,y) = 0; %If so we flag the point
                    c=1; %We say that the Flag matrix is not empty
                    %flag(x,y) = 1;
                end
            end
        end
    end

    %---DELETION---
    f=f.*flag; % After generation of the flag Matrix, we delete all the points flagged
```

```

%---STEP 2---
for x = 2 : rows-1
    for y = 2 : cols-1 % For each inner pixel
        Tp=0; %Reset T(p)
        P(1) = f(x-1,y); %creation of our vector following the 8-neighborhood notation
        P(2) = f(x-1,y+1);
        P(3) = f(x,y+1);
        P(4) = f(x+1,y+1);
        P(5) = f(x+1,y);
        P(6) = f(x+1,y-1);
        P(7) = f(x,y-1);
        P(8) = f(x-1,y-1);
        NP = sum(P); %Same as Step1
        if f(x,y)==1 && NP<=6 && NP >=2 %Same as Step1
            for i=1:7
                if P(i)==0 && P(i+1)==1 %Same as Step1
                    Tp=Tp+1;
                end
            end
            if P(8)==0 && P(1) ==1 %Same as Step1
                Tp=Tp+1;
            end
            Pa = P(1)*P(3)*P(7); %Calculate the 2 new condition for the 2nd Step
            Pb = P(1)*P(5)*P(7);
            if Tp ==1 && Pa ==0 && Pb ==0 %We verify if the flagged point fulfill all left conditions
                flag(x,y) = 0; %If so we flag the point
                c=1; %We say that the Flag matrix is not empty
                %flag(x,y) =1;
            end
        end
    end
end

%---DELETION---
f=f.*flag;
%f=f-flag;
%figure, imshow(f)
%flag = zeros(rows,cols);
end

figure,
subplot(1,2,1)
imshow(F),title("Original Image")
subplot(1,2,2)
imshow(f),title("skeletonized image")

```

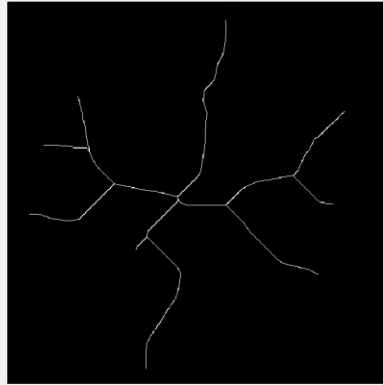
## Results

As expected, we can clearly see the skeletons of our product. When the shape is complex with varieties outlines, the skeleton is clear and precise.

Original Image



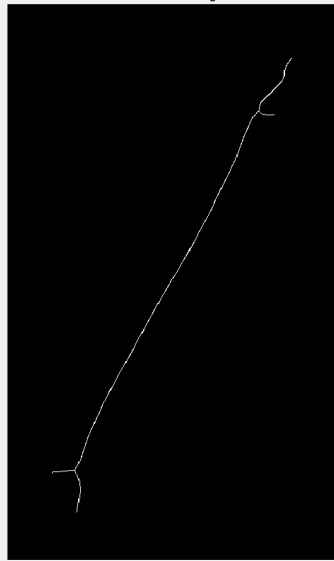
skeletonized image



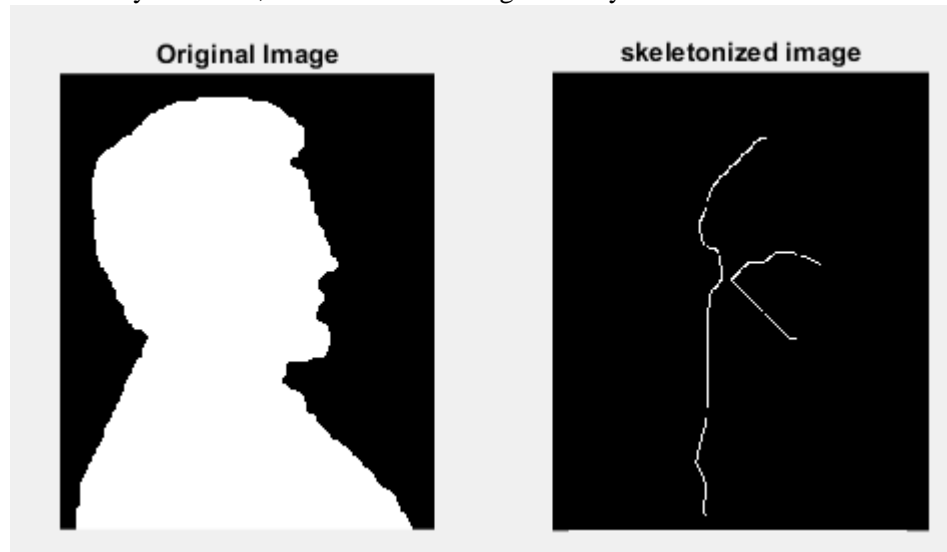
Original Image



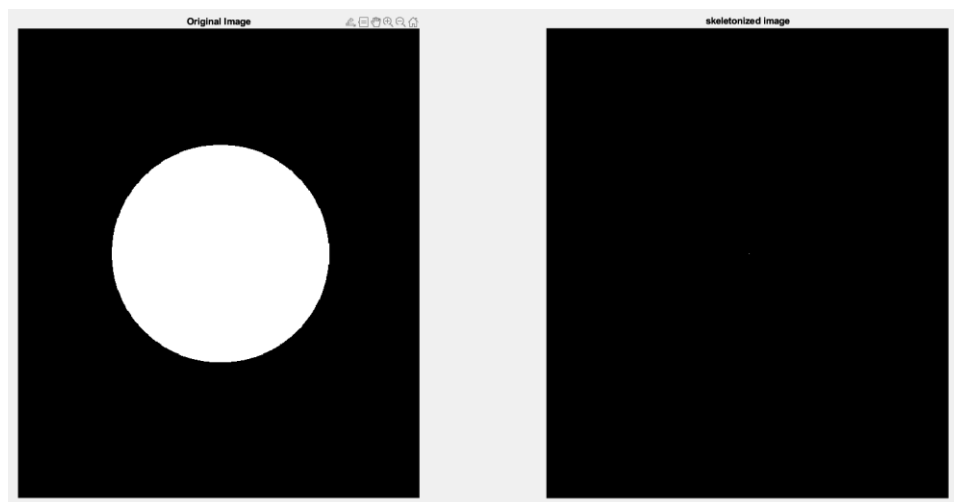
skeletonized image

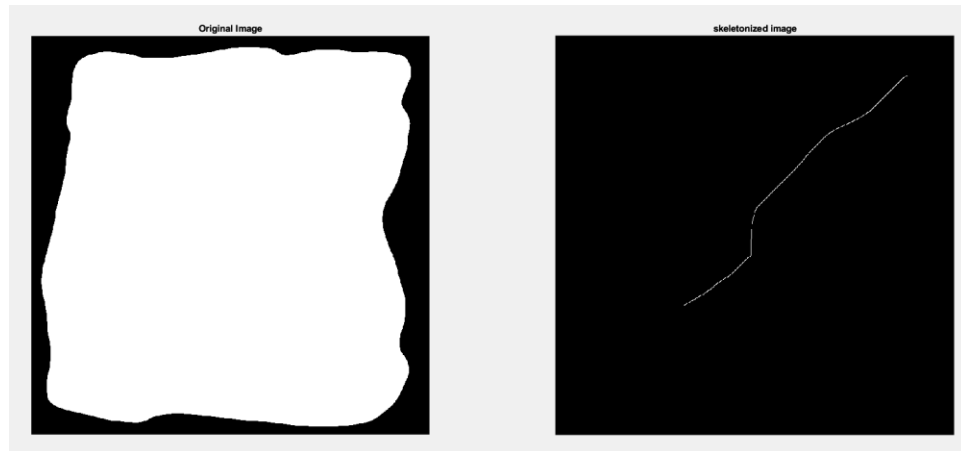


Now when applied to a face from the side for example, as there are not a lot of details, the skeleton is not very effective, and we cannot recognize it by its thinned structure.



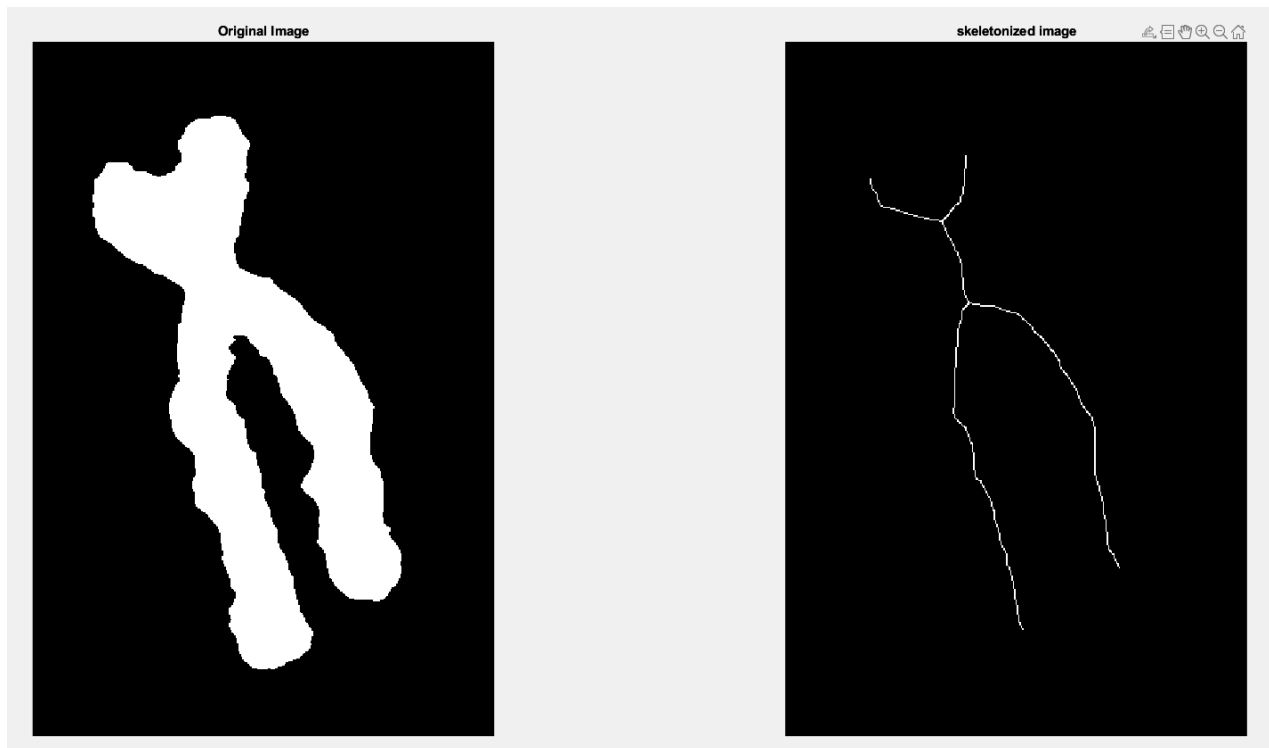
Another example with pictures with no edge or angle outlined, the skeleton is not effective and so we cannot recognize the image only from its skeleton. This method is more valuable for a body, a geographic landscape rather than the recognition of a face.





### *Application of skeletonizing procedure for generating human chromosome*

As chromosomes have various shapes, with complex outlines it might be hard to build a precise skeleton of it. However, depending on the binarization parameters, a good skeleton can be created in order to have a good overview of the shape of the chromosome



### *Time consummation for implementation of skeleton procedure*

There is a little bit of delay before receiving the results of the skeletonization. This delay could be a problem if the method was used for live event, for example recognizing if the skeleton of a gun, knife or a weapon was out, having an instantaneous answer and taking the necessary actions could prevent any unwanted events from happening.





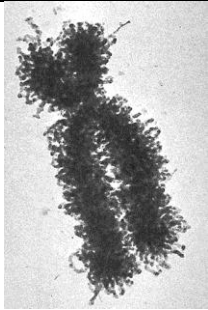

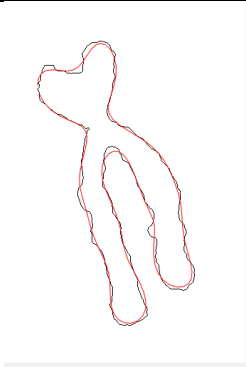





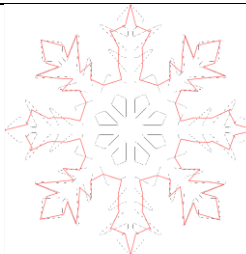
## ---Problem 2: Fourier descriptors---

First, we need to state how we will adapt the original code for Fourier descriptors (fd.m), for our purposes.

The coefficient of the threshold value to binarize our original image will be different for each image, since they all have different luminosity and contrast settings. We will create a parameter for this value, which will make its variation easier. This parameter will be called `bw_param`.

For a given image, we will vary the number of descriptors, which is directly dependant from the value of our parameter `u_max`.

By using the given code for Fourier descriptors adapted to our purposes, we can build this table of the subjective number of descriptors needed to keep the image recognizable:

Original image	Bw_param	Black and White image	Minimal u_max for image recognition and justification	Results
	1.7		13 For chromosomes, we chose the minimal <code>u_max</code> allowing to visualize the X shape that makes chromosome recognizable. Since the little curve at the top making the X appears only at <code>u_max = 13</code> , we chose that number	
	0.4		8 To recognize a hand, the definition of the five fingers are important. <code>U_max = 8</code> is the first where all five are visible.	
	0.6		52 For a valid snowflake, all little shapes in each branch need to be represented.	

### *Analysis of results*

We can assess for the results than the minimal number of Fourier Descriptors needed to recognize an image is correlated with the number and complexity of details needed that make an image recognizable. For example, a snowflake is recognizable from a simple cross shape because of its small details on all branches : therefore a high number of descriptors are needed comparing to a hand or a chromosome.

We can also note that its not necessarily the number of details that defines the minimal number of descriptors but also the curvature of these details. We can argue that a hand has more features than a chromosome (5 fingers versus 4 “branches”), but in this chromosome image, the top X curvature was note really distinct from the other curves, and therefore took more descriptors to appear, whereas in the hand, each finger is a clear curvature, so it is more easily represented by Fourier descriptors.

### ***Application of Fourier descriptors***

Fourier descriptors can be used to compress images by reducing its contour to a minimal of coefficients, while still keeping the shape recognizable.

On the opposite way it is also useful for Image Reconstruction: Fourier descriptors can be used to reconstruct an image from its Fourier coefficients. This approach can be useful when the original image is lost or damaged, and only its Fourier coefficients are available.

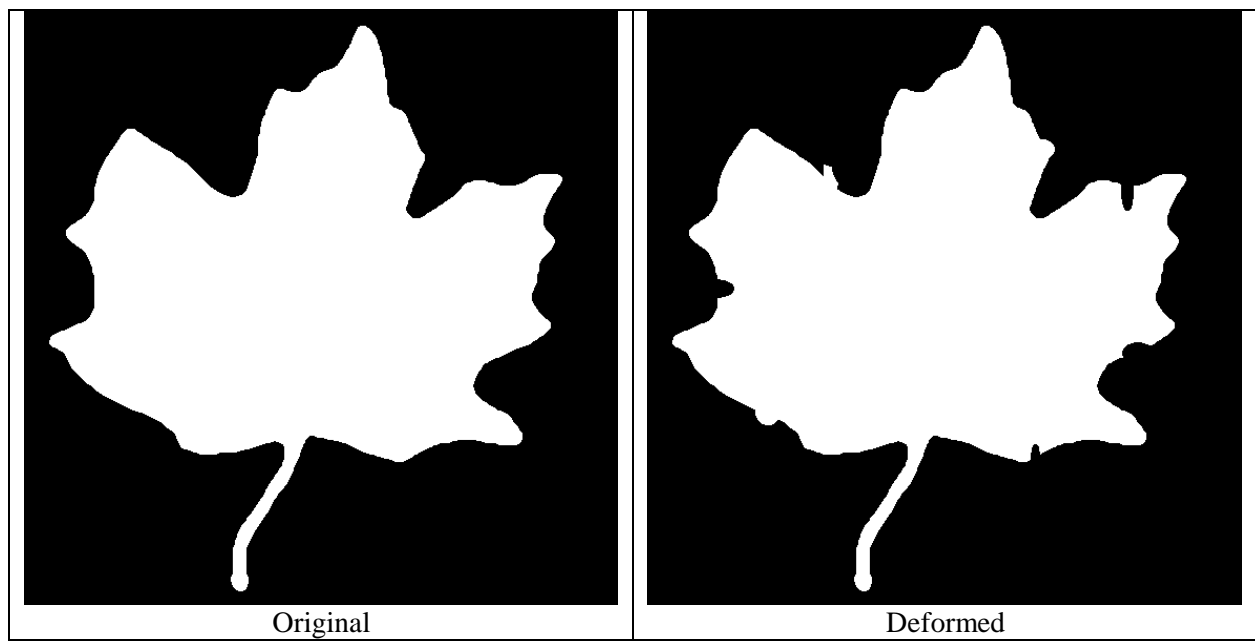
It can also be used for image filtering, where noise in the background can be removed with frequency domain filtering without affecting the Fourier descriptors used to define the image.

### ***Analysis of transformations/invariances***

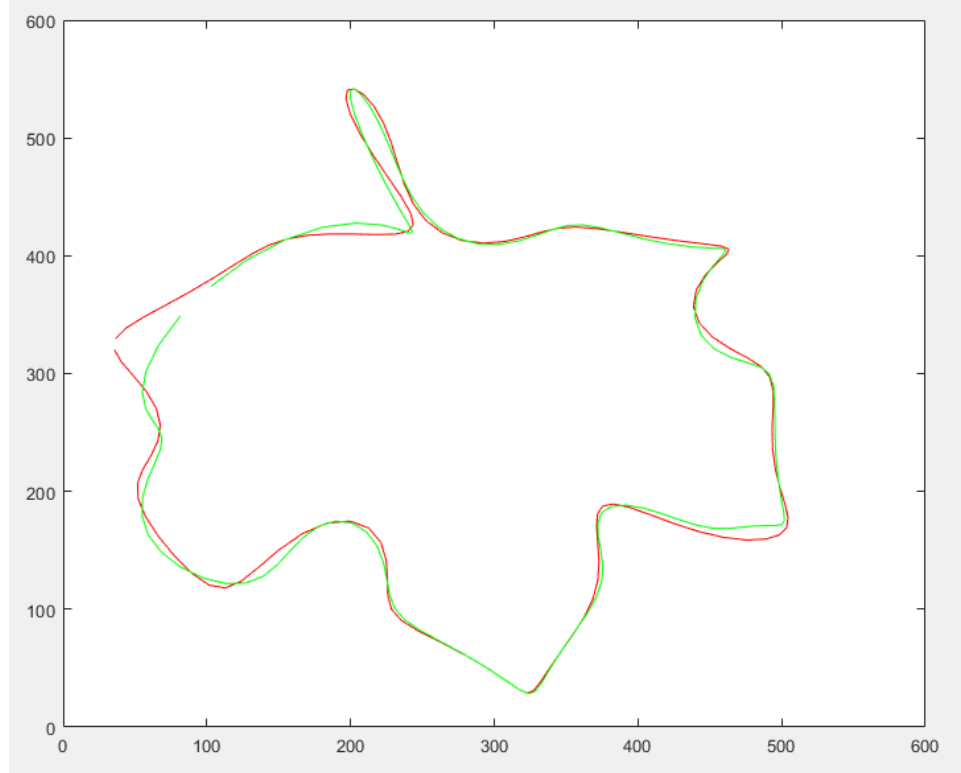
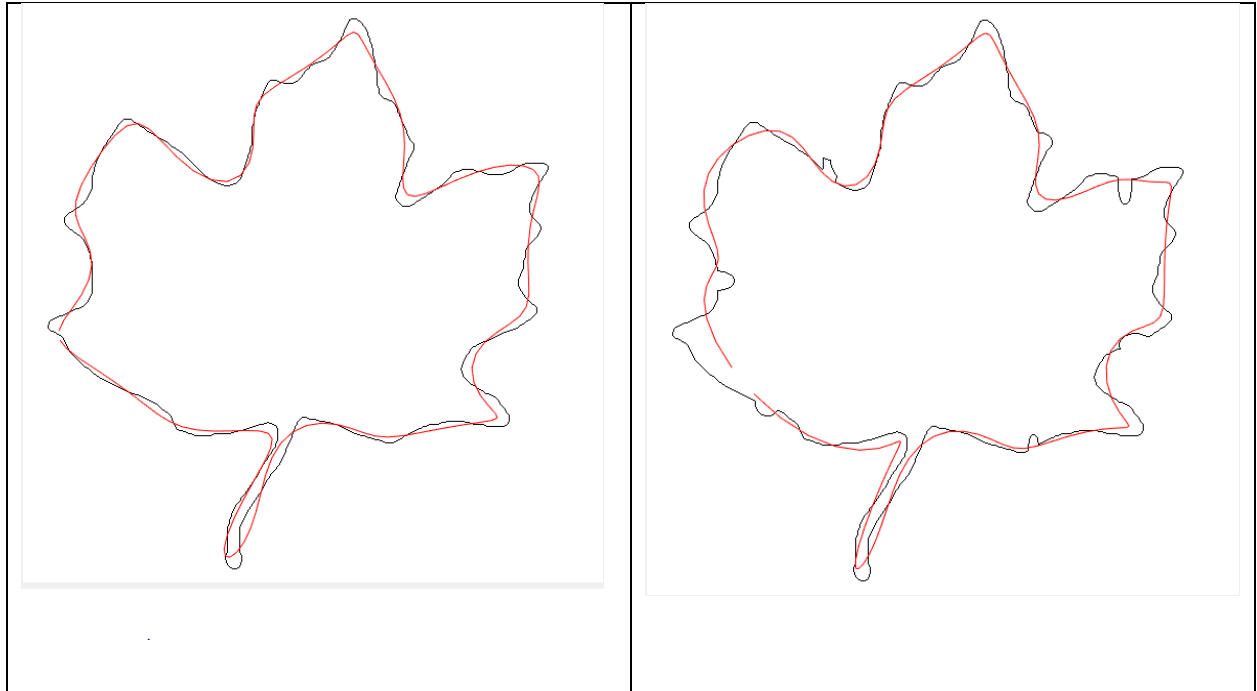
We will test the invariance of Fourier descriptors on two aspects: Deformation and noise

#### **Deformation:**

Let’s compare the results of Fourier descriptors on two versions of a black and white image, the original, and the deformed one:



We then compare the contour of 13 Fourier descriptors for the images:

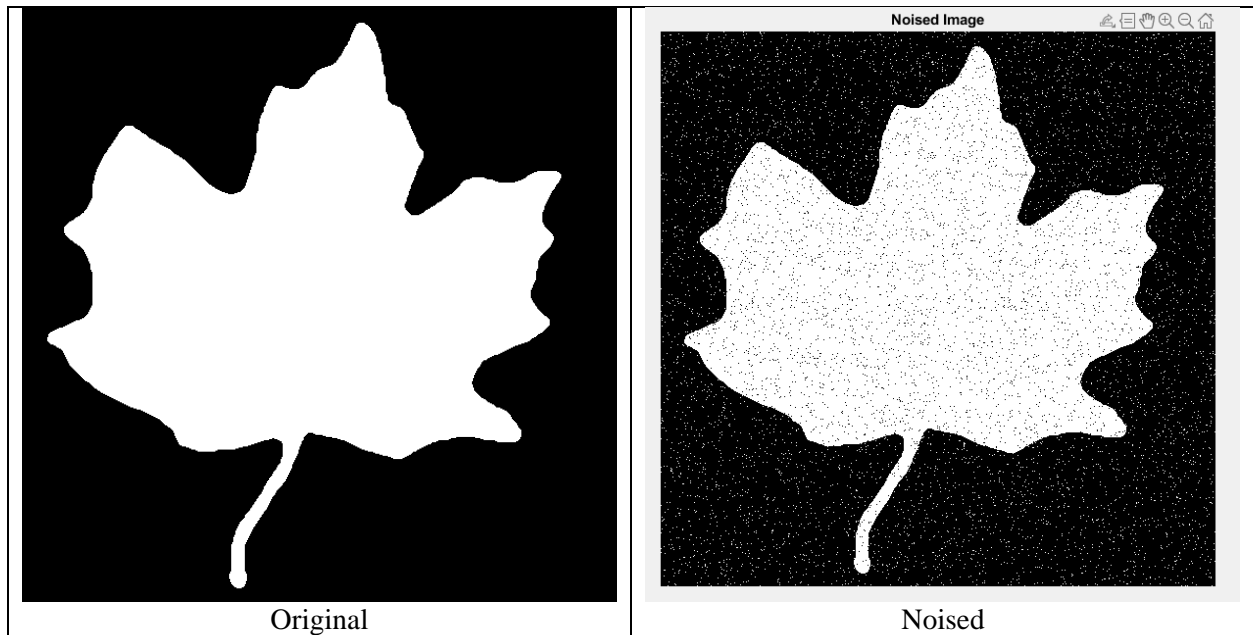


*Comparison between Fourier Descriptors for Original and Deformed Image (Red : Original, Green : Deformed)*

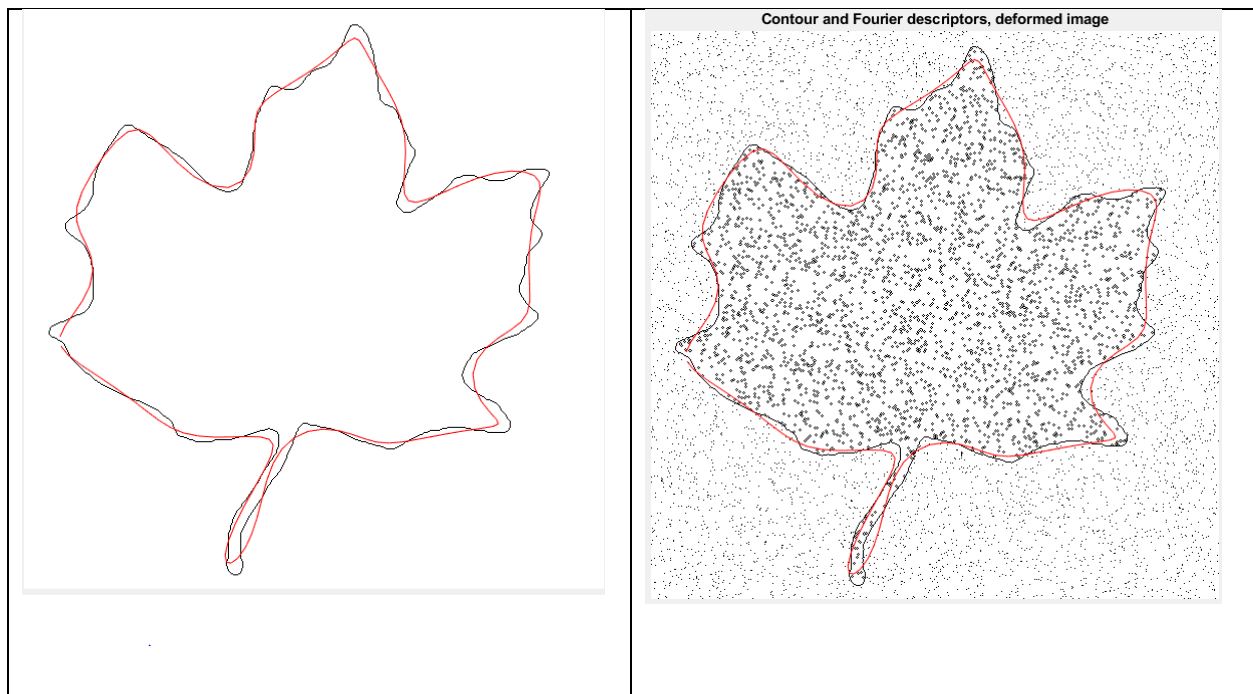
We can see that it is only minimally impacted by the deformations, therefore Fourier descriptors are pretty robust to small deformations.

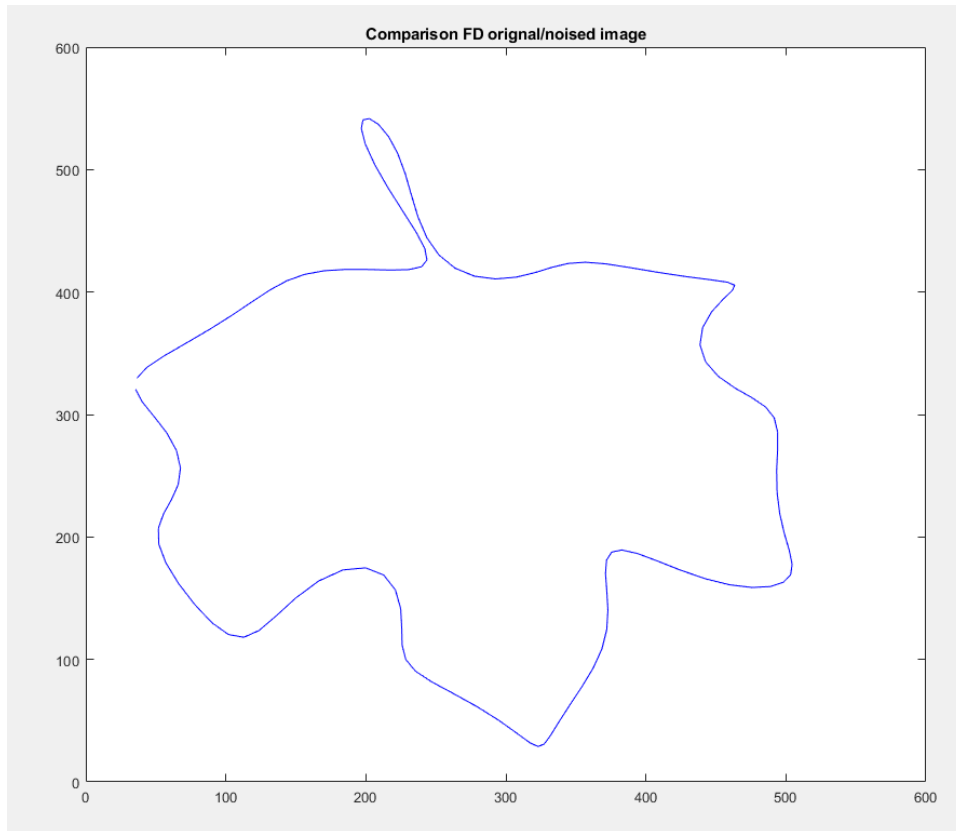
Black and Pepper Noise :

Let's compare the results of Fourier descriptors on two versions of a black and white image, the original, and the deformed one :



We then compare the contour of 13 Fourier descriptors for the images :





*Comparison between Fourier Descriptors for Original and Noised Image (Red : Original, Blue : Deformed)*

We can see that Fourier descriptors are totally invariant to black & pepper noise, the two curves overlays each other perfectly.