Wiener filtering

Image Analysis

Subject: Image Analysis
Instructor: O. Laligant
Student: GARCIA Frederic
Postgraduate in: VIBOT MSc
E-mail: fgarciab@eia.udg.es
Student number: 26006629

Objective: restoration of a burred and noisy image (program in Matlab)

The Wiener filter purpose is to reduce the amount of noise present in a signal by comparison with an estimation of the desired noiseless signal. It is based on a statistical approach.

Typical filters are designed for a desired frequency response. The Wiener filter approaches filtering from a different angle. One is assumed to have knowledge of the spectral properties of the original signal and the noise, and one seeks the LTI filter whose output would come as close to the original signal as possible. Wiener filters are characterized by the following:

1. Assumption: signal and (additive) noise are stationary linear stochastic processes with known spectral characteristics or known autocorrelation and cross-correlation.

2. Requirement: the filter must be physically realizable, i.e. causal (this requirement can be dropped, resulting in a non-causal solution).

3. Performance criteria: minimum mean-square error.

1. **Generate a 128x128p image:**
   a. **background = 0**
   b. **a centred 80x80p square of level 100**
   c. **a centred disk (radius = 20) of level = 200**

Matlab does not have a specific function to draw a square or a circle. Then it is necessary to do it manually. A grid of 128x128 cells has been done in order to define a mask for the square and another mask for the disk. The square mask is easy to be computed while for the disk mask has been necessary to apply the circumference equation (Eq. 1).

$$x^2 + y^2 < r$$

Eq. 1

Once both masks has been computed, there are multiply by the specific colour and merged using the *mergeImages(img1,img2)* function, which merge both images taking into account the maximum colour of each pixel. Hence, it maintains the square and disk colour after being merged in a single image.

The Matlab code to perform this step is presented in Table 1:

```
% 1. Generate a 128x128p image:
%   a. background = 0
n = 128; ind = -n/2:1:(n/2)-1;
[Y,X] = meshgrid(ind,ind);
cc = [0,0];  % center
%   b. a centred 80x80p square of level 100
r = 40; % radius of the square
sqMask = max(abs(X-cc(1)),abs(Y-cc(2))) < r; % mask containing the square
square = sqMask*100; % fill with color = 100
%   c. a centred disk (radius = 20) of level 200
r = 20; % radius of the disk
dkMask = (X-cc(1)).^2 + (Y-cc(2)).^2 < r^2; % mask containing the disk
disk = dkMask*200; % fill with color = 200
iSrc = mergeImages(square,disk);
```

Table 1. Step 1 MATLAB

The following figures present both masks and the resulting image after being merged.

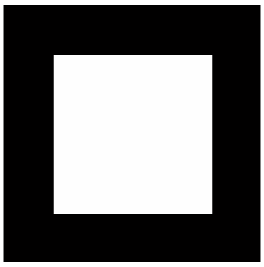
Figure 1a. Mask for the square image.

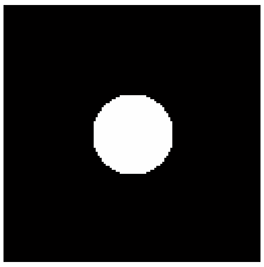
Figure 1b. Mask for the disk image.

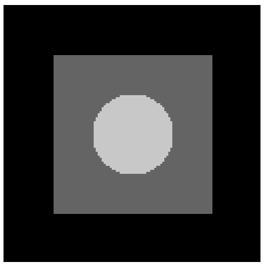
Figure 1c. Initial image with the specific colours for the square and for the disk.

2. **Perform a blurring step of the image with the following filter (choose s <1):**

$$h(x,y) = \frac{2}{2\pi} e^{-s\sqrt{x^2+y^2}}$$

The implementation of this step is based on the generation of the filter 'h', done by the function *generateFilter(s,img)* but once the filter is computed it is necessary to be applied on the image in order to obtain a blur image.

To apply this filters there are two possible ways:

1. By convolution of the filter and the image in the temporal domain. The problem of this way is that the Matlab function *conv2()* performs the 2D convolution of matrices but if [ma,na] = size(A) and [mb,nb] = size(B), then size(C) = [ma+mb-1,na+nb-1]. However, the resulting image size expected is the same as the input size.

   There are three flags in order to determine the output of conv2. These flags are:

   - 'full' which returns the full 2D convolution (default and presented in figure 2a).
   - 'same' which returns only the central part of the convolution that is the same size as A (Figure 2b).
   - 'valid' which returns only those parts of the convolution that are computed without the zero-padded edges size(C) = [ma-mb+1,na-nb+1] when all(size(A) >= size(B)), otherwise C is an empty matrix [].

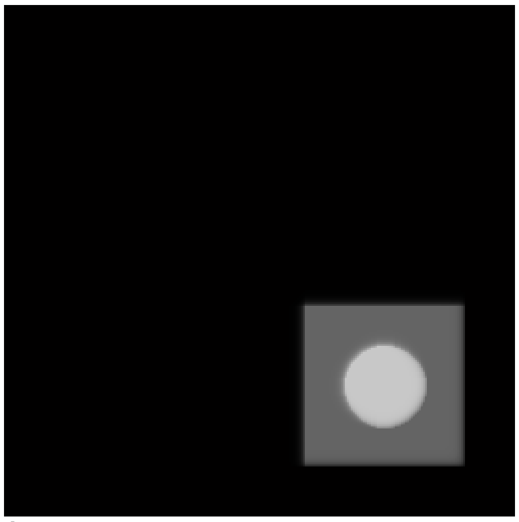
Figure 2a. Obtained image after the convolution using flag: ‘full’.

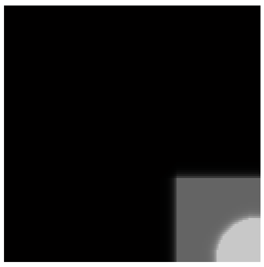
Figure 2b. Obtained image after the convolution using flag: ‘same’.

As can be observed the convolution does not returns the expected image size or the desired convolution part which implies to work in the frequency domain.

2. By computing the product between the filter and the image in the frequency domain. In order to do not spend much time trying to select the valid range of the convolution the computation has been done in the Fourier domain as is presented in Table 2.

```
% 2. Perform a blurring step of the image with the following filter:
s = 0.5;
[iBlur,h] = applyBlurFilter(s,iSrc);

function [iRes,h] = applyBlurFilter(s,img);
    iRes = [];

    h = generateFilter(s,img);
    H = fft2(h);
    IMG = fft2(double(img));
    I_BLUR = IMG .* H;
    iRes = ifft2(I_BLUR);
end

function h = generateFilter(s,img)
    h = [];

    [width,height] = size(img);
    for y = 1:height
        for x = 1:width
            h(x,y) = (s/(2*pi))^(-s*sqrt(x^2+y^2));
        end
    end
    % 'h' is the bluring filter
    h = h / sum(sum(h)); % normalize
end
```

Table 2. Step 2 MATLAB

The resulting image after the blur step is presented in figure 3:

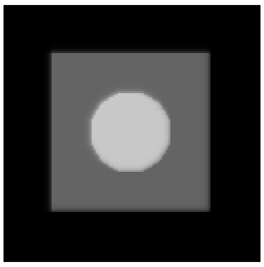

Figure 3. Blurred image.

3. Add a Gaussian (white) noise: variance v>10 (mean=0).

Table 3 presents the Matlab code to add white noise to an image. The noise is simulated by random values in the standard deviation area.

```
% 3. Add a gaussian (white) noise: variance v>10 (mean=0)
variance = 11; % Must be bigger than 10
std_dev = sqrt(variance);
noise = std_dev.*randn(size(iBlur));
iBlurNoise = iBlur + noise;
```

Table 3. Step 3 MATLAB

The resulting image after the addition of noise is presented in figure 4:

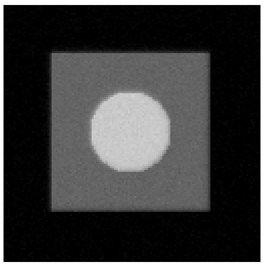

Figure 4. Noisy and blurred image.

4. Perform a restoration of this built image by using the simplified Wiener restoration filtering:

$$W = \frac{H^*}{|H|^2 + K}$$

where the restored image $\hat{F}$ is obtained from the observed (degraded) image G in the Fourier space by:

$$\hat{F} = WG$$

Give the K value leading to the best visual restoration.

In order to compute the K value leading to the best visual restoration an array from 0.001 to 0.1 with 100 intervals has been computed. Each value has been assigned to a specific K and the restoration has been performed.

The selected K value corresponds with the minimum error obtained between the original image and the restored one.

The Matlab code is presented in Table 4:

```
% 4. Perform a restoration of this built image by using the
simplifier
H = fft2(h); % Transform the filter into Fourier domain
% Give the K value leading to the best visual restoration, calculated
% by minimazing the error btw the initial image and the restored
image
K = linspace(0.001,0.1,100);
errorVect = zeros(1,100);
for i=1:length(K)
    % Generate restored Wiener filter
    W = conj(H)./(abs(H).^2 + K(i));
    % Apply filter
    G = fft2(iBlurNoise);
    F = W.*G;
    iRestored = uint8(ifft2(F));
    % Calculate error
    error = uint8(iSrc) - iRestored;
    errorVect(i) = mean(error(:))^2;
end
% Retrieve minimum error
[minErrorValue minErrorPos] = min(errorVect);
idealK = K(minErrorPos);
W = conj(H)./(abs(H).^2 + idealK);
G = fft2(iBlurNoise);
F = W.*G;
iRestored = ifft2(F);
```

Table 4. Step 4 MATLAB

And the restored image is presented in figure 5:

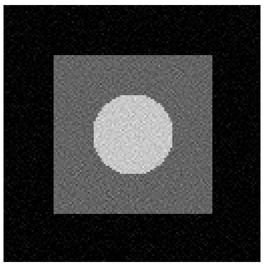

Figure 5. Restored image.

Figure 6 presents the results of all the computations until this step:

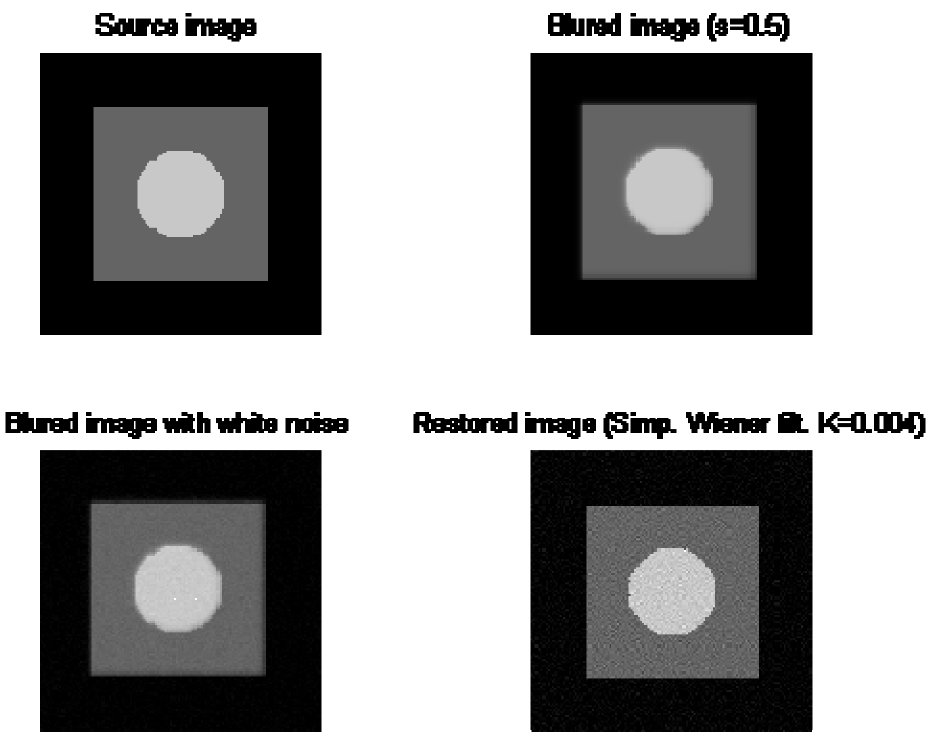
Source image
Blured image (s=0.5)
Blured image with white noise
Restored image (Simp. Wiener filt. K=0.004)

Figure 6. Results of the steps 1-4.

Although the information about the spectrum of the undistorted image and also of the noise is known, it is not possible to achieve a perfect restoration. The main reason is the random nature of noise

5. **Try to introduce two different blurs: one (s_1) for the square and another ($s_2<s_1/2$) for the disk. Add noise and restore the image. Conclusion?**

In order to perform this step, two different filters ‘h’ has been computed for the square and for the disk, respectively, with different ‘s’ values. After compute separately both images, there are added in order to obtain a single image. Then, the noise is added and the restoration process is performed.

In that case, it is not possible to find a priori a fit value of ‘s’ to make a correct restoration. Then, an array of 100 elements between s_1 and s_2 has been use to compute the blur filter and besides, as in the step 4, the best K value has been computed for each ‘s’ value.

The Matlab code for this step is presented in Table 5:

```
% 5. Try to introduce different blurs: one (s1) for the square and another
% (s2<s1/2) for the disk. Add noise and restore the image
s1 = 0.5; % Square
s2 = (s1/2) - 0.05; % Disk
% Perform the bluring step
[iBlurSq,hSq] = applyBlurFilter(s1,square); % Apply the filter to Square
[iBlurDk,hDk] = applyBlurFilter(s2,disk); % Apply the filter to Disk
iBlurNew = mergeImages(iBlurSq,iBlurDk);
% Add the gaussian noise
iBlurNoiseNew = iBlurNew + noise;

% Find the ideal H by trial and error
sVect = linspace(s1,s2,100);
```

```
kVal = []; % Best K value for each value of s
kVect = linspace(0.001,0.1,100);
errorVect = zeros(1,100);
for i=1:length(sVect)
    % Generate the filter
    h = generateFilter(sVect(i),iBlurNew);
    H = fft2(h);
    errorKVect = zeros(1,100);
    for j=1:length(kVect)
        % Generate restored Wiener filter
        W = conj(H)./(abs(H).^2 + kVect(j));
        % Apply filter
        G = fft2(iBlurNoiseNew);
        F = W.*G;
        iRestoredNew = uint8(ifft2(F));
        % Calculate error
        error = uint8(iSrc) - iRestoredNew;
        errorKVect(j) = mean(error(:))^2;
    end
    % Retrieve minimum error
    [minErrorValue minErrorPos] = min(errorKVect);
    kVal(i) = kVect(minErrorPos);
    % Generate restored Wiener filter
    W = conj(H)./(abs(H).^2 + kVal(i));
    % Apply filter
    G = fft2(iBlurNoiseNew);
    F = W.*G;
    iRestoredNew = uint8(ifft2(F));
    % Calculate error
    error = uint8(iSrc) - iRestoredNew;
    errorVect(i) = mean(error(:))^2;
end
% Retrieve minimum error
[minErrorValue minErrorPos] = min(errorVect);
% Generate the best filter 'h'
h = generateFilter(sVect(minErrorPos),iBlurNew);
H = fft2(h);
W = conj(H)./(abs(H).^2 + kVal(minErrorPos));
G = fft2(iBlurNoiseNew);
F = W.*G;
iRestoredNew = ifft2(F);
```

Table 5. Step 5 MATLAB

The results are presented in the following figure:

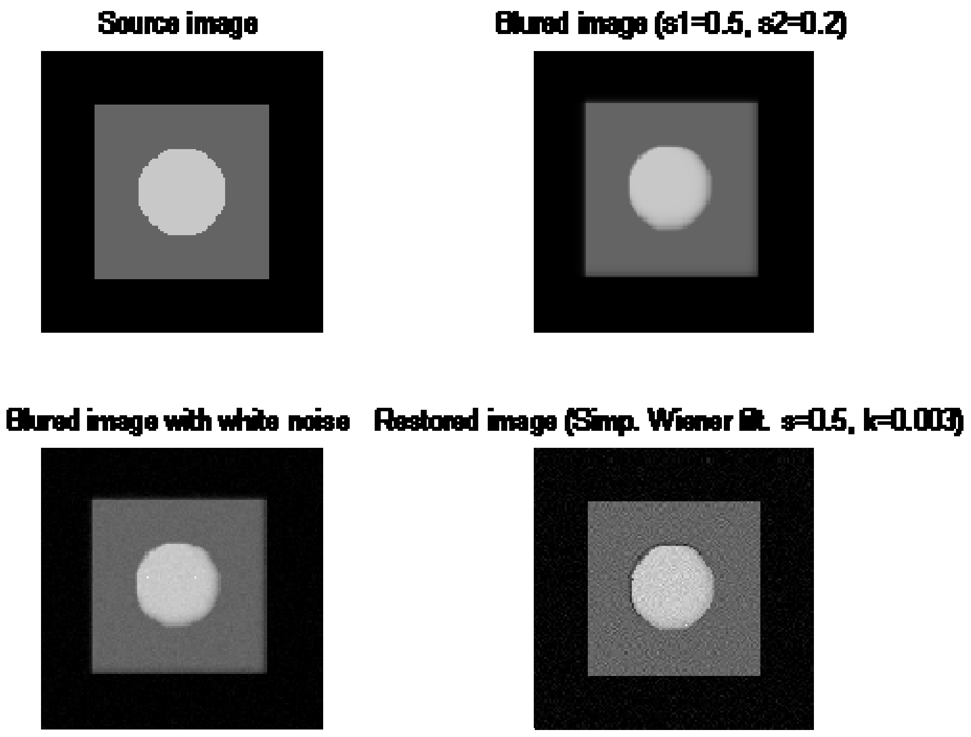
Source image
Blured image (s1=0.5, s2=0.2)
Blured image with white noise
Restored image (Simp. Wiener filt. s=0.5, k=0.003)

Figure 6. Results of the step 5.

In that case the restoration is slightly worse than in the previous step but the invested time to perform the best restoration is considerably higher.

Conclusions:

As is said in the introduction, the Wiener filter is used to reduce the amount of noise presented in a signal by comparison with an estimation of the desired noiseless signal. As can be observed in the results, the image restoration is not absolutely perfect but it achieves a very close image to the original one.

However, if it is necessary to find a value of ‘s’ or ‘K’ to perform the closest restoration, the invested time is very high, which can be a problem in many cases.