Ψηφιακή Επεξεργασία και Ανάλυση Εικόνας

Ορέστης Νικόλας, ΑΜ 228438, 5ο Έτος, ΗΜΤΥ

Εργαστηριακή Άσκηση, μέρος Α

Υλοποίηση οδηγιών υλοποίησης των ασκήσεων:

Μετατόπιση FFT, ώστε το χωρικό σημείο (0, 0) να βρεθεί στο κέντρο:

```
A = [0 1 2 3 1; 4 3 2 1 0; 5 1 2 3 1]
matlab_fftshift = fftshift(A)
my_fftshift = FFTshift(A)
```

```
function func = FFTshift(F)
        dimentions = size(F);
        d1 = dimentions(1); %Rows
        d2 = dimentions(2); %Columns
        med = floor(d2 / 2);
        func = zeros(d1, d2);

        func(:,1:med) = F(:,d2 - med + 1:end);
        func(:,med + 1:end) = F(:, 1:d2 - med);

        tfunc = transpose(func);
        dimentions = size(tfunc);
        d1 = dimentions(1); %Rows
        d2 = dimentions(2); %Columns
        med = floor(d2 / 2);

        func = zeros(d1, d2);

        func( : , 1:med) = tfunc( : , d2 - med + 1:end);
        func( : , med + 1:end) = tfunc( : , 1:d2 - med);

        func = transpose(func);
    end
```

```
A =

     0     1     2     3     1
     4     3     2     1     0
     5     1     2     3     1


matlab_fftshift =

     3     1     5     1     2
     3     1     0     1     2
     1     0     4     3     2


my_fftshift =

     3     1     5     1     2
     3     1     0     1     2
     1     0     4     3     2
```

Απεικόνιση του πλάτους του FFT:

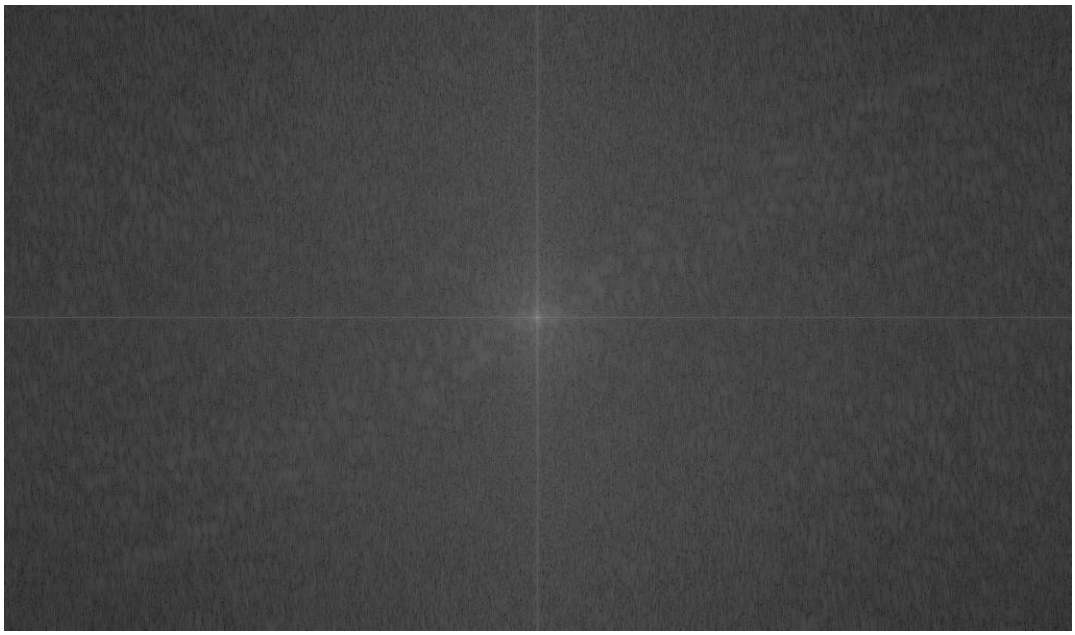
Γενικότερα έχω χρησιμοποιήσει την grayscaled Eikona4.jpg που υπάρχει στο eclass για να δοκιμάσω αν δουλεύουν οι κώδικες μου.

```
f = imread('Eikona4.jpg');
f = rgb2gray(f);
F = FFT2(f);
pwr = PowerSpectrum(F);
psd = 10*log10(PowerSpectrum((FFTshift(F))) + 0.01);
imshow(psd, [0 255]);
```

```
function func = FFT2(f)
    N1 = size(f,1);
    temp = zeros(N1, size(f,2));
    for n1 = 1:N1
        temp(n1,:) = fft(f(n1,:));
    end
    temp = transpose(temp);
    N2 = size(temp,1);
    for n2 = 1:N2
        temp(n2,:) = fft(temp(n2,:));
    end
    func = transpose(temp);
end
```

```
function func = PowerSpectrum(F)
        func = real(F).^2 + imag(F).^2;
end
```

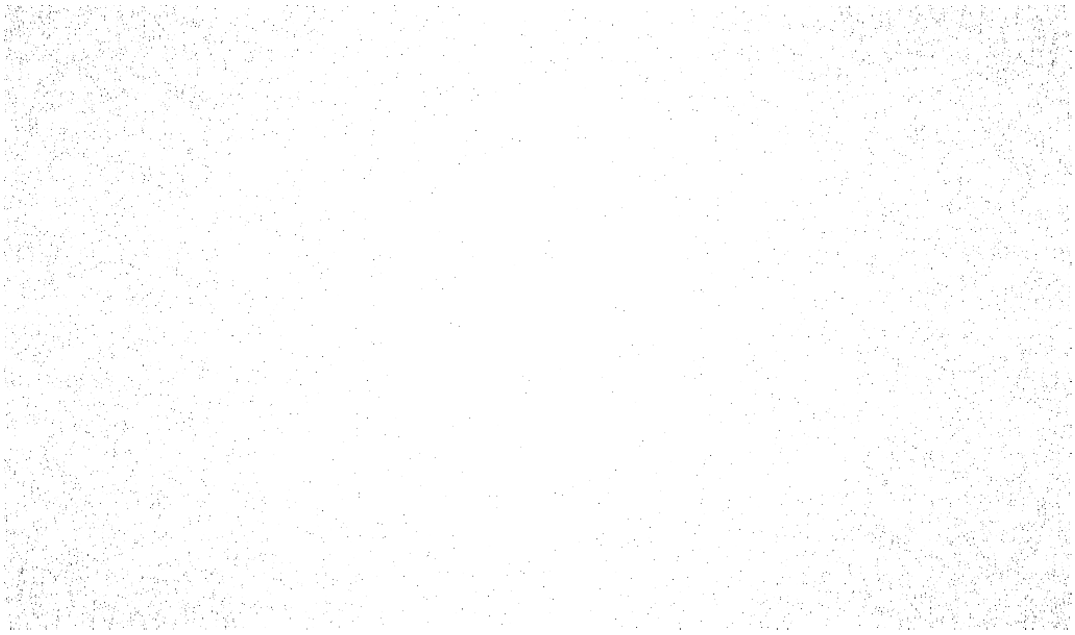
Αποτελέσματα όπως τα περιμέναμε(χρησιμοποιόντας τη λογαριθμική κλίμακα):



Χωρίς λογαριθμική κλίμακα και με τετραγωνική ρίζα, δηλαδή:

```
psd1 = PowerSpectrum((FFTshift(F)));
imshow(sqrt(psd1), [0 255]);
```

Αποτέλεσμα:



Όπου είναι λογικό γιατί οι τιμές του abs(F) είναι πολύ μεγάλες και ειδικά μετά το quantization σε τιμές [0 255] οι περισσότερες τιμές είναι άσπρες.

Φίλτρο Μεσαίου (Median) και Πρόσθεση κρουστικού θορύβου (Salt & Pepper):

```
function func = Med3Filter(f)
    dimentions = size(f);
    d1 = dimentions(1); %Rows
    d2 = dimentions(2); %Columns
    shadowImg = zeros(d1, d2);

    %Applying mask
    for row = 2:d1 - 1
        for col = 2:d2 - 1

            mask = f(row - 1:row + 1, col - 1: col + 1);
            mask = reshape(mask, 1, 9);
            mask = sort(mask);
            shadowImg(row, col) = mask(5);

        end
    end
    %By default edges of the picture will not be masked
    shadowImg(1, :) = f(1, :);
    shadowImg(d1, :) = f(d1, :);
    shadowImg(:, 1) = f(: , 1);
    shadowImg(:, d2) = f(: , d2);
    func = shadowImg;
end
```

Η **Med3Filter** συνάρτηση παίρνει μια μάσκα 3x3 και σαρώνει την εικόνα βρίσκοντας και αντικάθιστά τα στοιχεία με τα κοντινότερα στο μέσο όρο στοιχεία της μάσκας. Επίσης θα χρησιμοποιήσω την **AddSaltNoise** για να δείξω ότι το φίλτρο αφαιρεί ικανοποιητικά θόρυβο τύπου salt & pepper:

```
function func = AddSaltNoise(f, p)

        %if Pa == Pb;
        percen=p;
        Prob_den_f=255*percen/100;
        func = f;

        Rmatrix = randi( [0, 255], [size(f,1), size(f,2)]);

        func(Rmatrix <= Prob_den_f/2) = 0;
        func(Rmatrix > Prob_den_f/2 & Rmatrix < Prob_den_f) = 255;

    end
```

Original Picture

```
f = imread('Eikona4.jpg');
f = rgb2gray(f);
imshow(f);
```

Salt & pepper picture

```
f = imread('Eikona4.jpg');
f = rgb2gray(f);
f_ = AddSaltNoise(f,0.5);
imshow(f_);
```

Retained picture after applying median mask

```
f = imread('Eikona4.jpg');
f = rgb2gray(f);
f_ = AddSaltNoise(f,0.5);
f_ = Med3Filter(f_);
imshow(f_,[0 255]);
```

Original picture(greyscaled)

Picture after salt & pepper noise

Retained picture after applying median mask

Πρόσθεση λευκού Gaussian θορύβου

```
f = imread('Eikona4.jpg');
f = rgb2gray(f);
f_ = AddGaussianNoise(f,10,400);
imshow(f_,[0 255]);
```

```
function func = AddGaussianNoise(f, const, variance)
        gNoise = randn([size(f,1), size(f,2)]) * sqrt(variance) + const;
        func = gNoise + double(f);
    end
```

Αποτέλεσμα gaussian noise

Υλοποίηση 1-D DCT, 2-D DCT, 2-D FFT και τα inverse αυτών χρησιμοποιώντας 1-D FFT και 1-D IFFT του matlab.

```
function func = FDCT1(f)
    N = length(f);
    y = zeros(1,2*N);
    y(1:N) = f;
    y(N+1:2*N) = fliplr(f);
    Y = fft(y);
    k=0:N-1;
    temp = real(exp(-1i.* pi.*k./(2*N)).*Y(1:N));
    temp(1) = temp(1) * sqrt(1/N)/2;
    temp(2:N) = temp(2:N) * sqrt(2/N)/2;
    func = temp;

end
```

```
function func = FFT2(f)
    N1 = size(f,1);
    temp = zeros(N1, size(f,2));
    for n1 = 1:N1
        temp(n1,:) = fft(f(n1,:));
    end
    temp = transpose(temp);
    N2 = size(temp,1);
    for n2 = 1:N2
        temp(n2,:) = fft(temp(n2,:));
    end
    func = transpose(temp);
end
```

```
function func = FDCT2(f)
    N1 = size(f,1);
    N2 = size(f,2);
    temp = zeros(N1,N2);

    for n1 = 1:N1
        temp(n1,:) = FDCT1(f(n1,:));
    end

    temp = transpose(temp);

    for n2 = 1:N2
        temp(n2,:) = FDCT1(temp(n2,:));
    end
    func = transpose(temp);
 end
```

Όλες οι συναρτήσεις βγάζουν ακριβώς τα ίδια αποτελέσματα με τις built-in αντίστοιχες συναρτήσεις του matlab. Στις ασκήσεις θα χρησιμοποιώ τις δικιές μου συναρτήσεις.

```
function func = IFFT2(f)
        N1 = size(f,1);
        temp = zeros(N1, size(f,2));
        for n1 = 1:N1
            temp(n1,:) = ifft(f(n1,:));
        end
        temp = transpose(temp);
        N2 = size(temp,1);
        for n2 = 1:N2
            temp(n2,:) = ifft(temp(n2,:));
        end
        func = transpose(temp);
    end
```

Άσκηση 1: Φιλτράρισμα στο πεδίο συχνοτήτων

Image normalization:

```
%% Normalizing image to values 0->255
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\moon.tiff');
normalised_f = uint8(255*mat2gray(f));
```

Results:

```
>> f(f>253)

ans =

  0×1 empty uint8 column vector

>> normalised_f(normalised_f>253)

ans =

  2×1 uint8 column vector

   255
   254
```

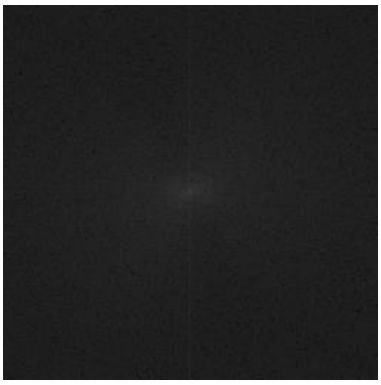
Βλέπουμε ότι έχει γίνει σωστά το normalization και τώρα έχουμε τιμές σε όλο το εύρος.

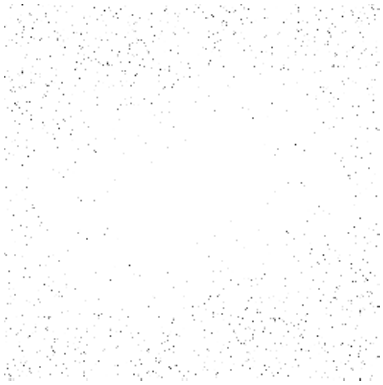
2D-FFT and FFT magnitude plots:

```
%% Applying 2-D FFT with row-column logic plus shifting zero frequency element to center
F = FFT2(normalised_f);
F = FFTshift(F);
```

```
%% Linear and logarithmic display of image fourier magnitude
Fmag = FFTmagnitude(F);
FmagLog = 10*log10(Fmag);
FmagLin = Fmag;
subplot(1,2,1);
imshow(FmagLog,[0 255]);
subplot(1,2,2);
imshow(FmagLin,[0 255]);
```

Results:





Η FFTmagnitude και FFTshift είναι δική μου συνάρτηση και υπολογίζει το magnitude του FFT. Όπως έχω αναφέρει και πρίν, οι συναρτήσεις για τους μετασχηματισμούς είναι δικιές μου και όλες χρησιμοποιούν fft 1-D και την λογική γραμμών – στηλών. Τα αποτελέσματα είναι αναμενόμενα αφού η διαφορά τιμών του μέγιστου με το ελάχιστο σημείο της Fmag είναι τεράστια, συγκεκριμένα:

```
>> max(max(Fmag))

ans =

     8575112

>> min(min(Fmag))

ans =

    7.5466
```

```
>> numel(Fmag(Fmag<max(max(Fmag)/100)))

ans =

        65467

>> numel(Fmag(Fmag>max(max(Fmag)/100)))

ans =

    69
```

Το οποίο υποδεικνύει ότι μακράν οι περισσότερες τιμές είναι κοντά στο min άρα το αποτέλεσμα είναι λογικό για το linear plot.

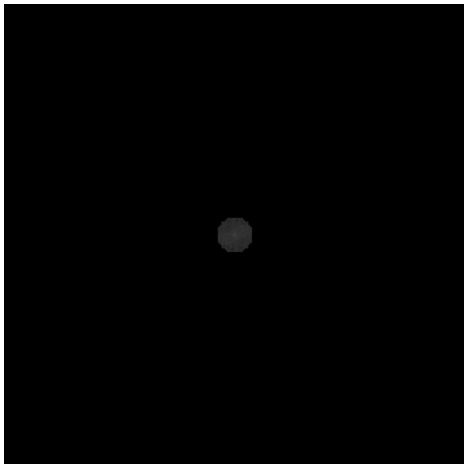
Low-Pass image filtering in frequency domain:

```
%% Low-Pass filtering in Frequency Domain
[x, y] = meshgrid( -128:127, -128:127);
z = sqrt(x.^2 + y.^2);
c = z<10;
filtered_F = F.*c;
imshow(10*log10(abs(filtered_F)),[0 255]);
```

```
%% Retrieving Filtered image
filtered_f = IFFT2(fftshift(filtered_F));
filtered_f = uint8(filtered_f);
imshow(filtered_f,[0 255]);
```

Results:

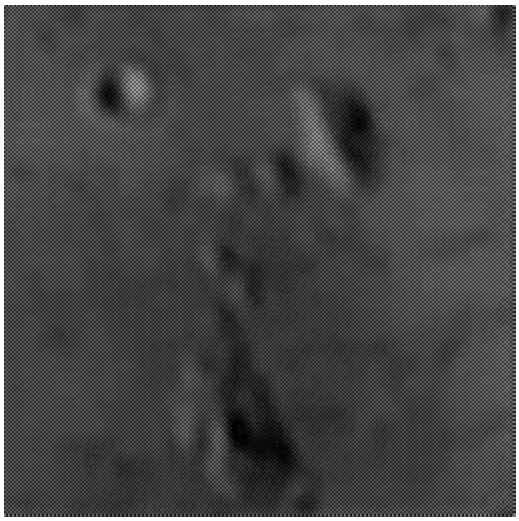
Multiplication on frequency result:



Original picture:

After low-pass filtering with re-shifting:

After low-pass filtering without re-shifting:



Όπως βλέπουμε μετά το filtering στο frequency domain ξανακάνω fftshift ώστε να επαναφέρω το σημείο (0,0), μετά κάνω IFFT2 για να γυρίσω στο χώρο. Αμα δεν κάνω το shift τότε έχουμε το φαινόμενο που φαίνεται στην τελευταία εικόνα.

Άσκηση 2: Συμπίεση εικόνας με χρήση DCT

Image segmentation:

```
%% Segmenting the original 256x256 image into 64 32x32 images
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\clock.tiff');
F = FDCT2(f);
A = SegmentImage32(f);
```

Έχω δημιουργήσει την συνάρτηση **SegmentImage32** η οποία απλά γυρνάει ένα A = 64x32x32 matrix όπου πρακτικά έχω τους 64 32x32 υποπίνακες της original 256x256 εικόνας.

DCT of the segmented images:

```
%% DCT on segmented 32x32 images
C = zeros(64,32,32);    %DCT's of the segmented sub-images
%B(:,:) = A(1,:,:);
for i = 1:64
    temp(:,:) = A(i,:,:);
    C(i,:,:) = FDCT2(temp);
end
```

Εδώ παρομοίως για κάθε υποπίνακα κάνω DCT και τα αποθηκεύω στη 64x32x32 μεταβλητή C.

Compressing using threshold:

Εδώ πιστεύω είναι αναγκαίο να περιγράψω λίγο τί είχα στο μυαλό μου για την **αρχική** μου υλοποίηση εφόσον πιστεύω δεν είναι αρκετά κατατοπιστική η εκφώνηση σε κάποιες λεπτομέρειες. Αυτό που κάνω είναι ότι αρχικά διαλέγω ένα ποσοστό τιμών που θα κρατήσω και μετά διαλέγω ένα Threshold. Από το ποσοστό αυτό βγαίνει ένα νούμερο(`numOfValues`) με το πόσες τιμές θα κρατήσω για κάθε 32x32 υποπίνακα. Πρέπει να σημειώσω ότι είναι λογικό για κάποιο συγκεκριμένο Threshold να μην υπάρχουν όσες τιμές θέλω που να είναι πάνω από αυτό το Threshold οπότε θα κρατήσω ακόμα λιγότερες απ'ότι είχα σχεδιάσει αρχικά. π.χ άμα πρέπει να κρατήσω 100 τιμές για κάθε υποπίνακα και πάνω από το Threshold που έχω διαλέξει υπάρχουν μόνο 20 τότε απλά θα κρατήσω αυτές τις 20 τιμές, πράγμα το οποίο σημαίνει ότι εν τέλει θα κρατήσω ακόμα λιγότερη πληροφορία απ' ότι είχα σχεδιάσει.

Ο τρόπος με τον οποίο κάνω το compression είναι με τη λογική iloc, jloc, val. Για τις τιμές που είναι >= του Threshold κρατάω μέχρι και `numOfValues` elements, . Την ίδια λογική χρησιμοποιώ και στη συμπίεση με μέθοδο ζώνης. Με αυτή τη λογική δεν θα ισχύει το ότι άμα κρατήσω το 50% των στοιχείων τότε θα έχω συμπίεση 50% αλλά <=50%. Θα είναι 50% μόνο εάν σε όλους τους υποπίνακες θα καταφέρει να βρεί `numOfValues` elements τα οποία θα είναι πάνω από το κατώφλι ή αν το κατώφλι μου είναι η ελάχιστη τιμή που μπορεί να βρεθεί στους συντελεστές DCT.

Το reconstruct του πίνακα συντελεστών DCT γίνεται προφανώς χρησιμοποιώντας τα iloc, jloc, val. Σε αυτά θεωρώ ότι είναι η "συμπιεσμένη" πληροφορία. Το κατώφλι που έχω πάρει είναι (max-min)/1000.

Δεν θα αναθέσω τον κώδικα που έκανα για την πρώτη προσπάθεια παρόλο που υπάρχει στο παραδοτέο(Askhsh2_test.m), απλά ήθελα να το αναφέρω γιατί εκεί υλοποίησα αλγόριθμο συμπίεσης και αποσυμπίεσης ώστε πραγματικά να κρατάει ακριβώς όση πληροφορία χρειάζεται και όχι απλά να μηδενίζει όσα στοιχεία είναι κάτω του Threshold.

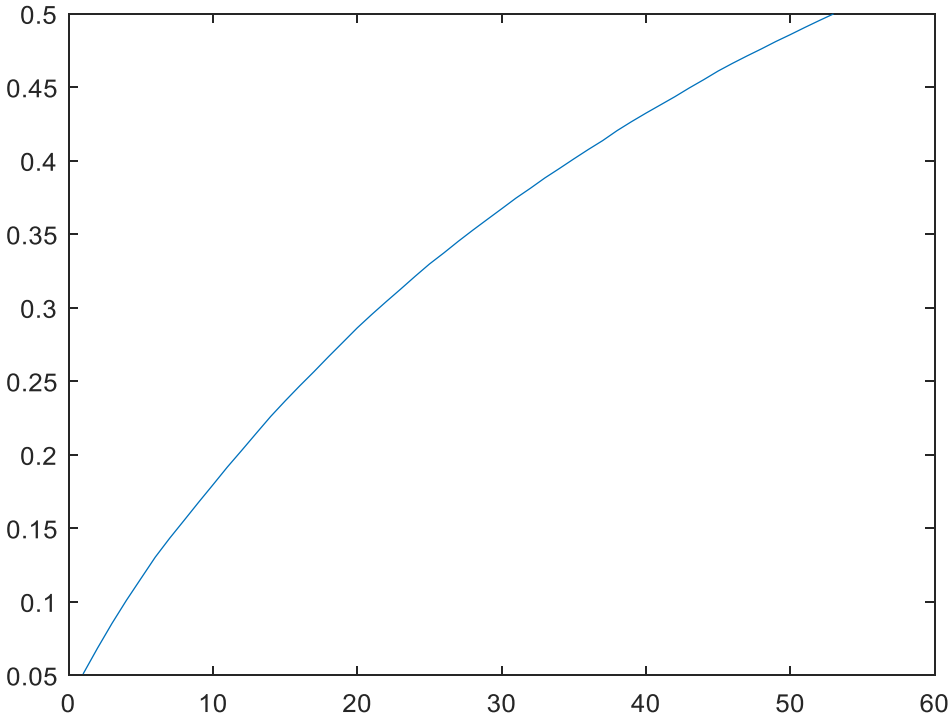
Η **τελική** υλοποίηση είναι η παρακάτω:

```
%% Compression using Threshold method
cntr = 1;
for l = 300:100:5500
    Thres = (max(max(max(C)))-min(min(min(C))))/l;
    C1 = zeros(64,32,32);

    % Compressing...
    for i = 1:64
        temp(:,:) = C(i,:,:);
        indices = find(abs(temp) < Thres);
        temp(indices) = 0;
        C1(i,:,:) = temp(:,:);
    end

    % IDCT on the compressed dct matrix
    C2 = zeros(64,32,32);
    for i = 1:64
        temp(:,:) = C1(i,:,:);
        C2(i,:,:) = idct2(temp(:,:));
    end

    % Reforming the sub-images into the whole
    newImage = ReconstructImage32(C2);
    %imshow(newImage,[0 255]);


    % Median square error & Compression percentage
    medSumErr = sum(sum(abs(newImage.^2 - double(f).^2)))/256/256;
    p = nnz(C1)/256/256;
    P(cntr) = p;
    Err(cntr) = medSumErr;
    cntr = cntr + 1;
end
```

Εδώ πρακτικά βάζω διάφορα Threshold ώστε να πηγαίνει το p από 5 εώς 50%. Επίσης υπολογίζω το median sum error του αρχικού σήματος από το reconstructed. Η συνάρτηση ReconstructImage32 αυτό που κάνει είναι παίρνει τον 64x32x32 πίνακα και τον διαμορφώνει σε 256x256.

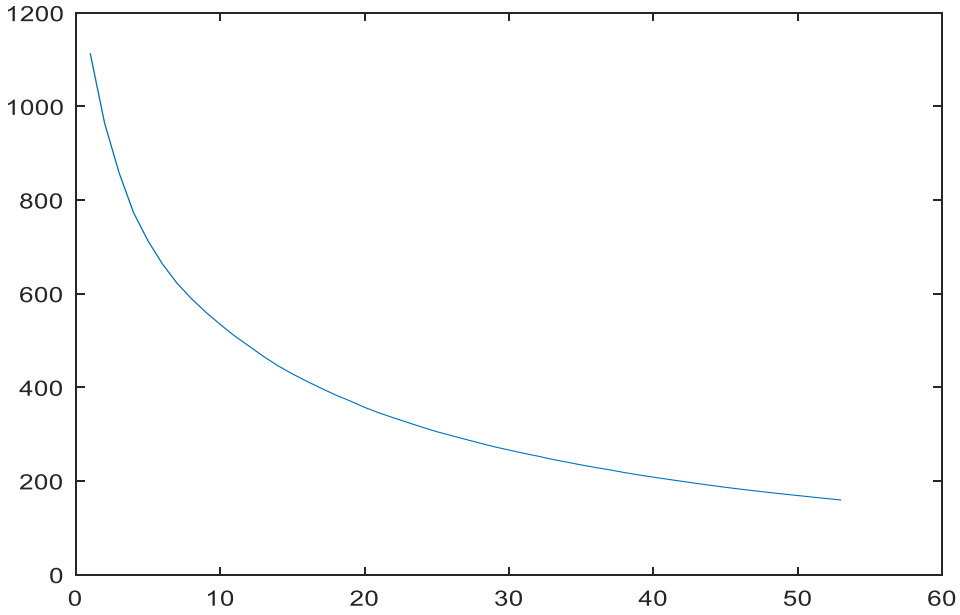
Επίσης να σημειώσω ότι δεν υλοποιώ κάποιον compression αλγόριθμο, απλά μηδενίζω τα στοιχεία που είναι κάτω του κατωφλίου , έτσι κ αλλιώς θεωρητικά τα απότελέσματα στο MedSumErr και στο p θα είναι τα ίδια.

Για τα συγκεκριμένα Threshold οι τιμές του p φαίνονται στο παρακάτω διάγραμμα:



Φαίνεται ότι το p πάει από 5% εώς 50%.

Επίσης οι τιμές του Error φαίνονται παρακάτω:



Εδώ αξίζει να σημειώσουμε ότι εφόσον πέφτει το Err εκθετικά όσο μεγαλύτερο το p τότε θεωρητικά μπορούμε να κρατάμε τους μισούς συντελεστές και να έχουμε μικρή απόκλιση από την αρχική εικόνα και compression ration 2:1. Η διαφορά στην εικόνα στο μάτι είναι ελάχιστή.

<u>50% compression</u> - <u>Original</u>

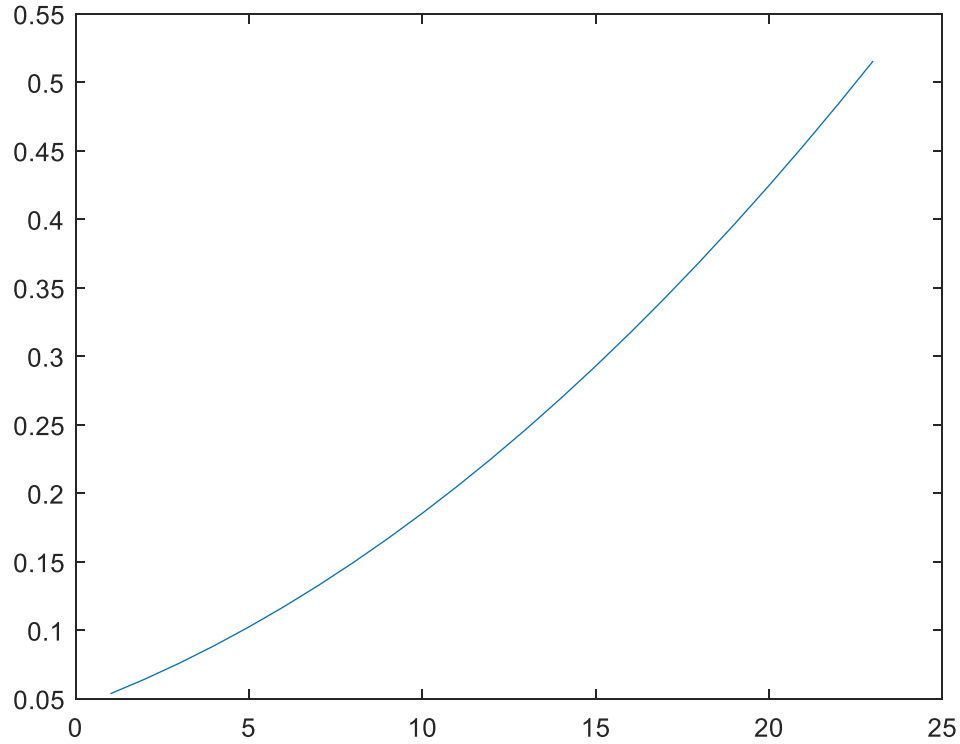
Compressing using zone:

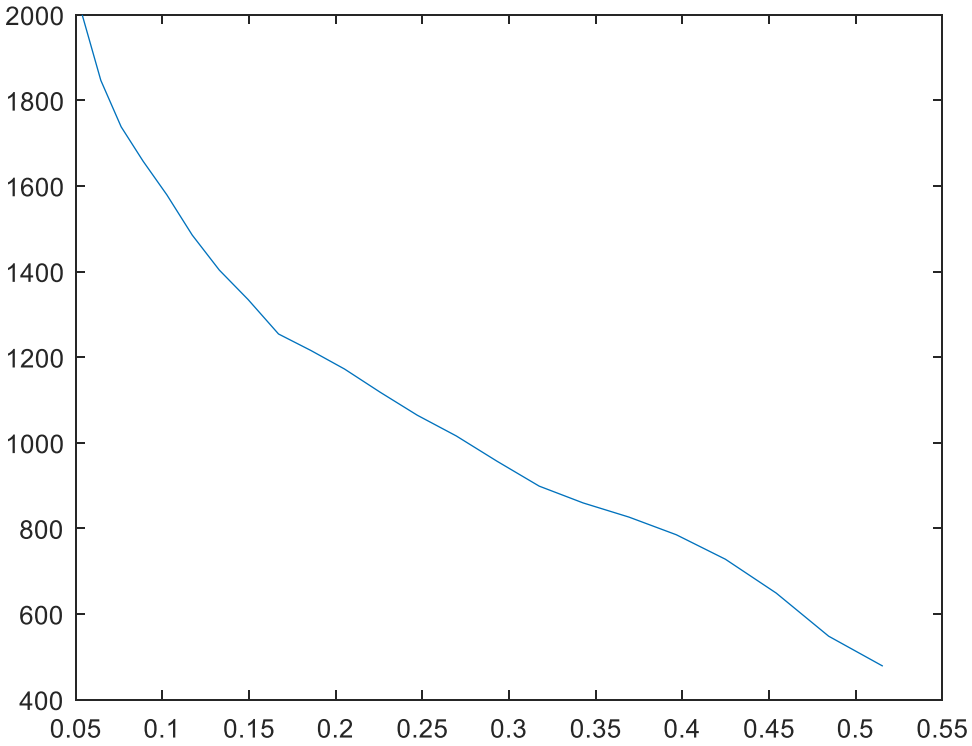
Εδώ επίσης έχω διαλέξει ανάλογες ζώνες ώστε να κρατάει από 5-50% των συντελεστών. Ο κώδικας φαίνεται παρακάτω:

```
%% Compression using zone method
C1 = zeros(64,32,32);
cntr = 1;
for l = 22:-1:0
    for i = 1:64
        temp(:,:) = C(i,:,:);
        temp = triu(rot90(temp,3),l);
        temp = rot90(temp);
        C1(i,:,:) = temp(:,:);
    end

    C2 = zeros(64,32,32);
        for i = 1:64
            temp(:,:) = C1(i,:,:);
            C2(i,:,:) = idct2(temp(:,:));
        end

    % Reforming the sub-images into the whole
    newImage = ReconstructImage32(C2);
    %imshow(newImage,[0 255]);


    medSumErr = sum(sum(abs(newImage.^2 - double(f).^2)))/256/256;
    p = nnz(C1)/256/256;
    P(cntr) = p;
    Err(cntr) = medSumErr;
    cntr = cntr + 1;
end
```

Παρακάτω φαίνεται το plot(P) για να δούμε αν όντως οι τιμές των p πάνε από 5 -50%. Τα l πρακτικά είναι οι ζώνες του τριγωνικού πίνακα που θα πάρω κάθε φορά και οι τιμές 22:-1:0 είναι ώστε να έχω p 5-50%.

Compressing using zone:


0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
5
10
15
20
25


2000
1800
1600
1400
1200
1000
800
600
400
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55

Το πρώτο είναι το plot(P). Το δεύτερο είναι το plot(P,Err). Φαίνεται να είναι και εδώ το Err περίπου εκθετικό.

Notes on results:

Γενικότερα φαίνεται το Threshold method να είναι πιο αποδοτικό από το Zone method και το σφάλμα σταθεροποιείται πολύ πιο γρήγορα όσο ανεβαίνει το p. Ακόμα και στα μικρά p το Threshold method δίνει μικρότερο Error. Στην 50% συμπίεση δίνει περίπου 200 medSumErr ενώ αντίστοιχα το Zone method δίνει περίπου 400. Άρα υποθέτω ότι στη συγκεκριμένη περίπτωση τουλάχιστον το Threshold method αποδίδει καλύτερο ποσοστό Error/Compression rate απότι το zone method.

Άσκηση 3: Βελτίωση εικόνας - Φιλτράρισμα Θορύβου

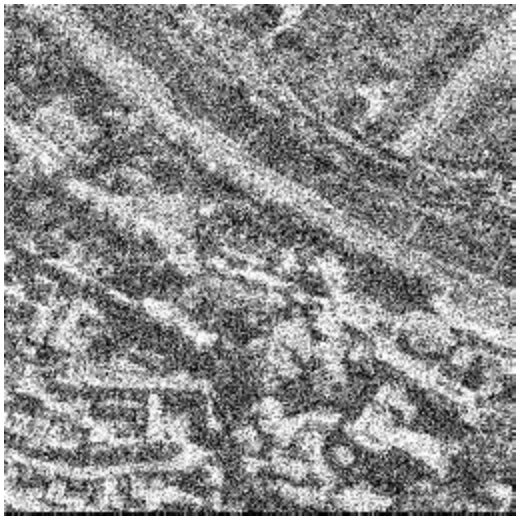
Gaussian noise:

```
%% Gauss noise
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\aerial.tiff');
f1 = imnoise(f,'gaussian', 0, 0.04);
%f1 = MovAvg3Filter(f1);
%f1 = Med3Filter(f1);
imshow(f1);
noise = double(f1) - double(f);
SNR = snr(double(f1), noise)
```

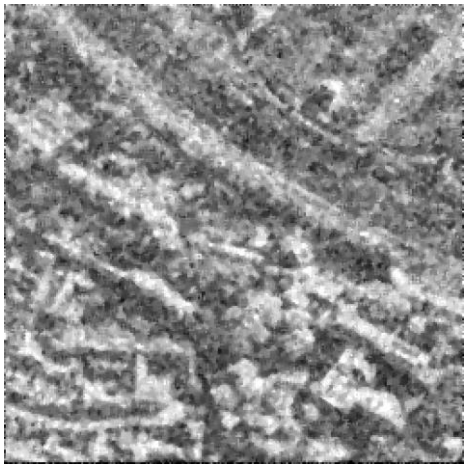
Γενικότερα με variance 0.04 βγάζει snr περίπου 10dB η τελική εικόνα. Από τα παρακάτω αποτελέσματα βλέπουμε ότι για white gaussian noise τα medium filter και moving average filter δεν είναι και ιδιαίτερα αποδοτικά. Ξέρουμε ότι το mean filter(moving average) είναι ένα linear filter και approximation του gaussian filter, το οποίο χρησιμοποιείται για να προκαλέσει blurring στα edges και να μειώσει το contrast. Σε αντίθεση με το median filter που είναι non-linear και χρησιμοποιείται για να μειώσει το θόρυβο κρατώντας τα edges σχετικά ικανοποιητικά. Το white noise που έχουμε εδώ θα μπορούσε να αντιμετωπισθεί με ένα weiner filter ιδανικά. Πιστεύω ότι στη συγκεκριμένη περίπτωση επειδή η εικόνα δεν είναι φυσικό τοπίο και έχει αρκετά edges άρα και μεγάλες συχνότητες τότε μας συμφέρει περισσότερο να χρησιμοποιήσουμε median filter ώστε να διατηρήσουμε τα edges καλύτερα. Αντιθέτως, άμα η εικόνα ήταν φυσικό τοπίο με πιο ομαλή εναλλαγή χρωμάτων ίσως να άξιζε περισσότερο το mean filter.

Original picture:

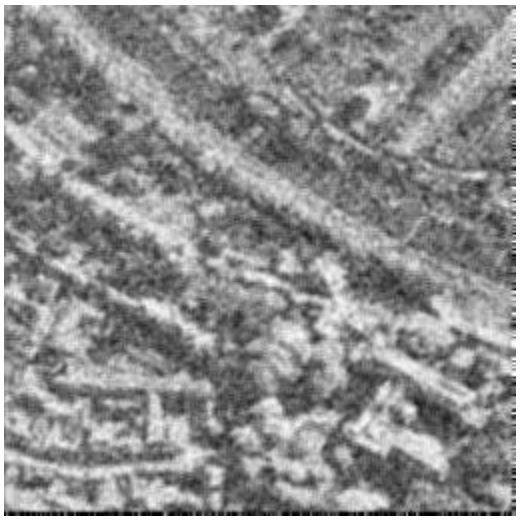
Gaussian noised picture (mean = 0, variance = 0.04, SNR = 10dB):



Gaussian noised picture after Median filter:



Gaussian noised picture after Moving Average filter:



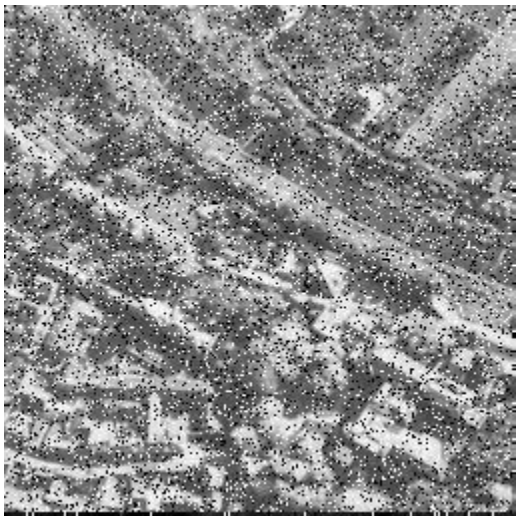
<3 <3

Salt & Pepper noise:

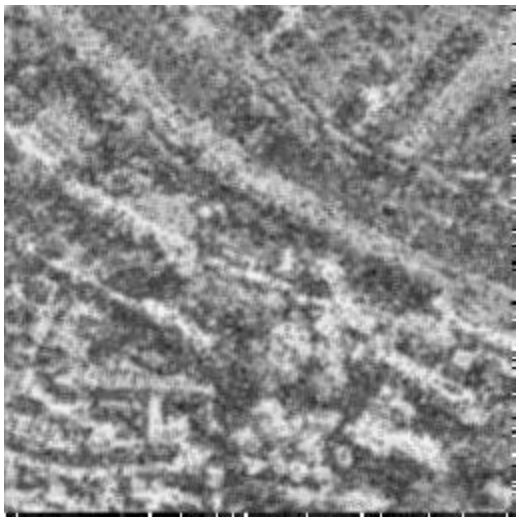
```
%% Salt noise
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\aerial.tiff');
f1 = AddSaltNoise(f,15);
%f1 = MovAvg3Filter(f1);
%f1 = Med3Filter(f1);
imshow(f1, [0 255]);
```

Από τα παρακάτω αποτελέσματα φαίνεται ότι για κρουστικό θόρυβο το median filter είναι πολύ καλύτερο απότι το moving average filter γιατί δίνει πολύ πιο καθαρή εικόνα και δεν είναι τόσο blurry.

Salt & Pepper noised picture (prob = 15%):



Noised picture after moving average filter:



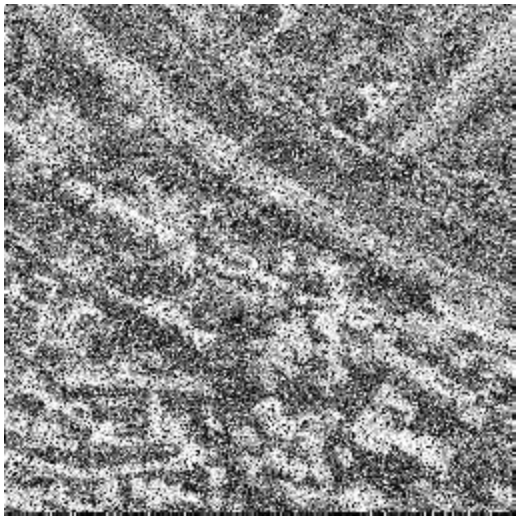
Noised picture after medium filter:

Applying both noises and trying to remove them:

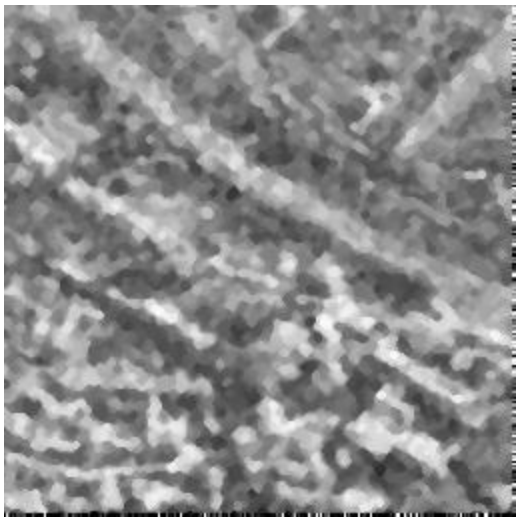
```
%% Both noises
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\aerial.tiff');
f1 = imnoise(f,'gaussian', 0, 0.04);
f1 = AddSaltNoise(f1,15);
%f1 = Med3Filter(f1);
%f1 = MovAvg3Filter(f1);
imshow(f1, [0 255]);
```

Πιστεύω η καλύτερη προσέγγιση θα ήταν να χρησιμοποιήσουμε κυρίως το median filter ενώ το mean filter να το χρησιμοποιήσουμε μόνο μία φορά γιατί προσθέτει ένα blurring στην εικόνα και όσο παραπάνω το βάζουμε τόσο χειρότερα. Η προσέγγισή που κάνω είναι 1 φορά median filter, 1 φορά mean και μετά 2 φορές median. Από διάφορους συνδυασμούς που δοκίμασα αυτός φαίνεται να δίνει το μεγαλύτερο SNR(~= 15.5dB vs ~= 7dB της noised image) και επίσης τα edges φαίνονται καλύτερα άρα οι ενδείξεις λένε ότι είναι ικανοποιητικό το αποτέλεσμα.

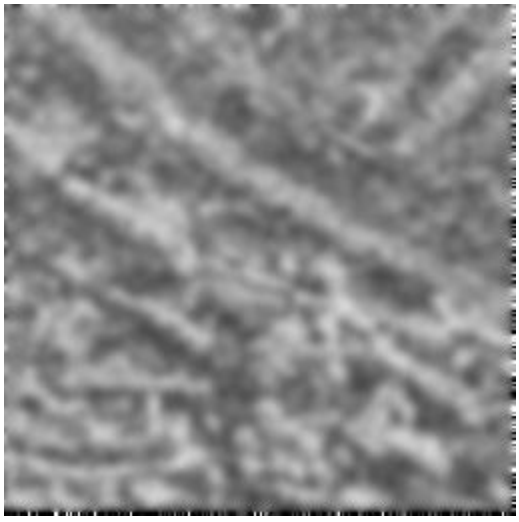
Double noised picture:



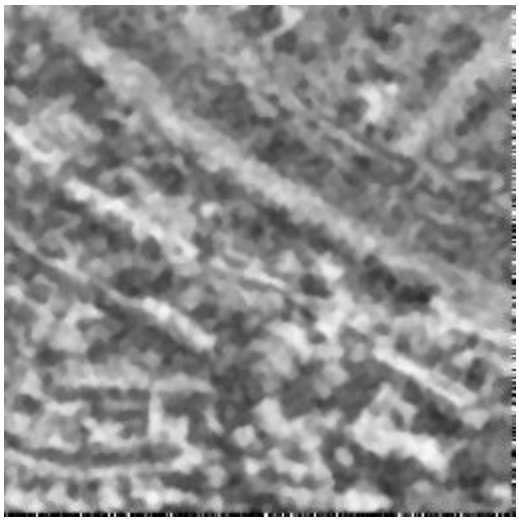
Noised picture after applying 10 times median filter:



Noised picture after applying 10 times moving average filter:

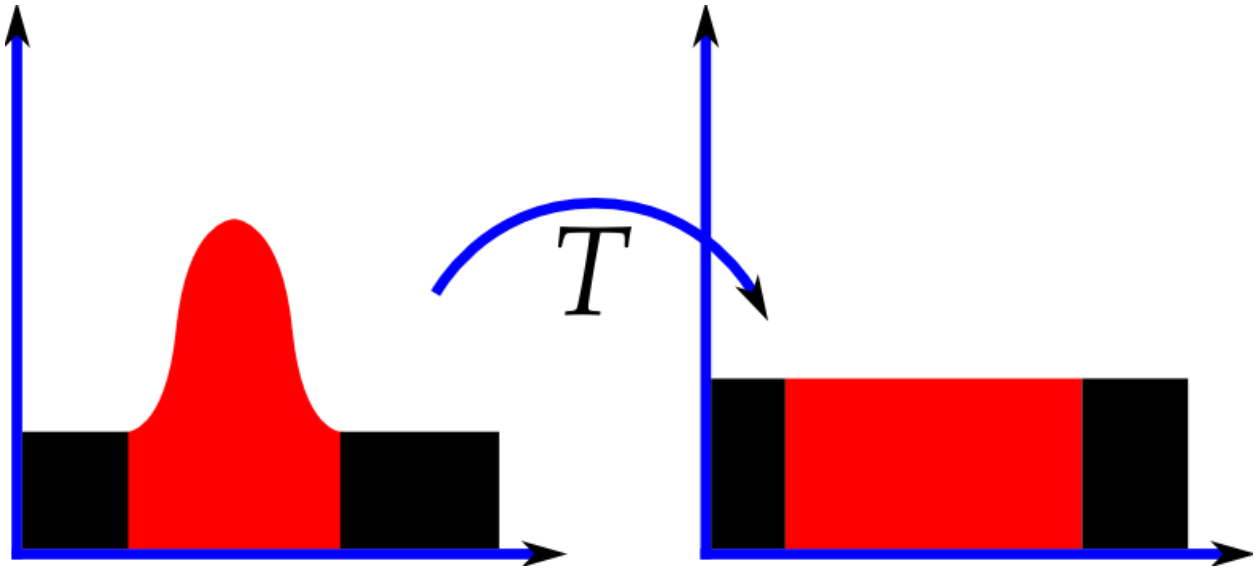


Noised picture after 1med-1mean-2med filters:



Άσκηση 4: Βελτίωση εικόνας – Εξίσωση ιστογράμματος

Some info about histogram equalization:



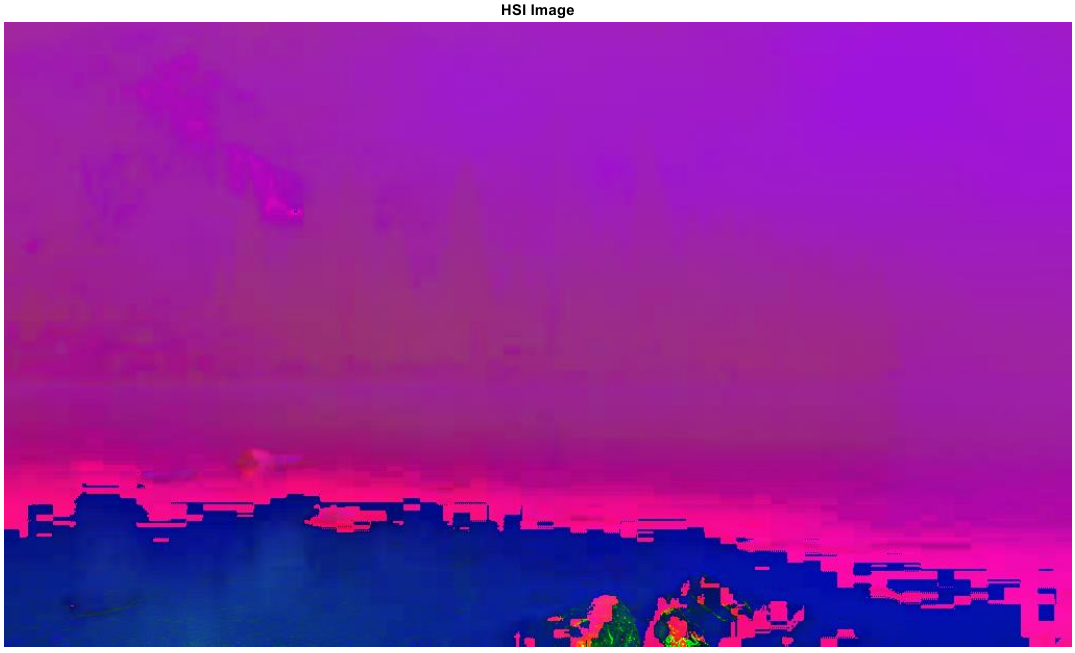
Γενικότερα το histogram equalization είναι μία μέθοδος με την οποία μπορούμε να αυξήσουμε το contrast μιας εικόνας, ειδικά όταν τα intensity της εικόνας αυτής έχουν αρκετά κοντινές τιμές. Αυτό κάνει περιοχές της εικόνας οι οποίες έχουν σχετικά μικρό τοπικό contrast να έχουν μεγαλύτερο. Ο τρόπος με τον οποίο καταφέρνει αυτή η μέθοδος να αυξήσει το contrast είναι με το να διαμοιράζει τις πιο συχνές τιμές του ιστογράμματος ώστε να μετασχηματιστεί σε ένα ιστόγραμμα όπως φαίνεται στην παραπάνω εικόνα(Ιδανικά). Επειδή οι κώδικες είναι αρκετά μεγάλοι δεν θα τους αναθέσω εδώ απλά θα αναθέσω τα αποτελέσματα. Οι κώδικες αντίστοιχα για τα HSI και RGB βρίσκονται στο παραδοτέο στα αρχεία Askhsh4HSI.m και Askhsh4RGB.m και έχω βάλει αρκετά comments για να είναι εύκολοι στην ανάγνωση.

Η μετατροπή από RGB σύστημα σε HSI(Hue-Saturation-Intensity) προκύπτει από διάφορες συναρτήσεις των red, green, blue πινάκων της RGB εικόνας. Ομοίως η μετατροπή HSI σε RGB προκύπτει από συναρτήσεις των hue, saturation, intensity της HIS εικόνας. Οι συναρτήσεις αυτές υλοποιούνται και περιγράφονται ακριβώς στους κώδικες που έχω αναθέσει.

Results:

RGB Image

HSI Image



Hue Image

Saturation Image

Intensity Image

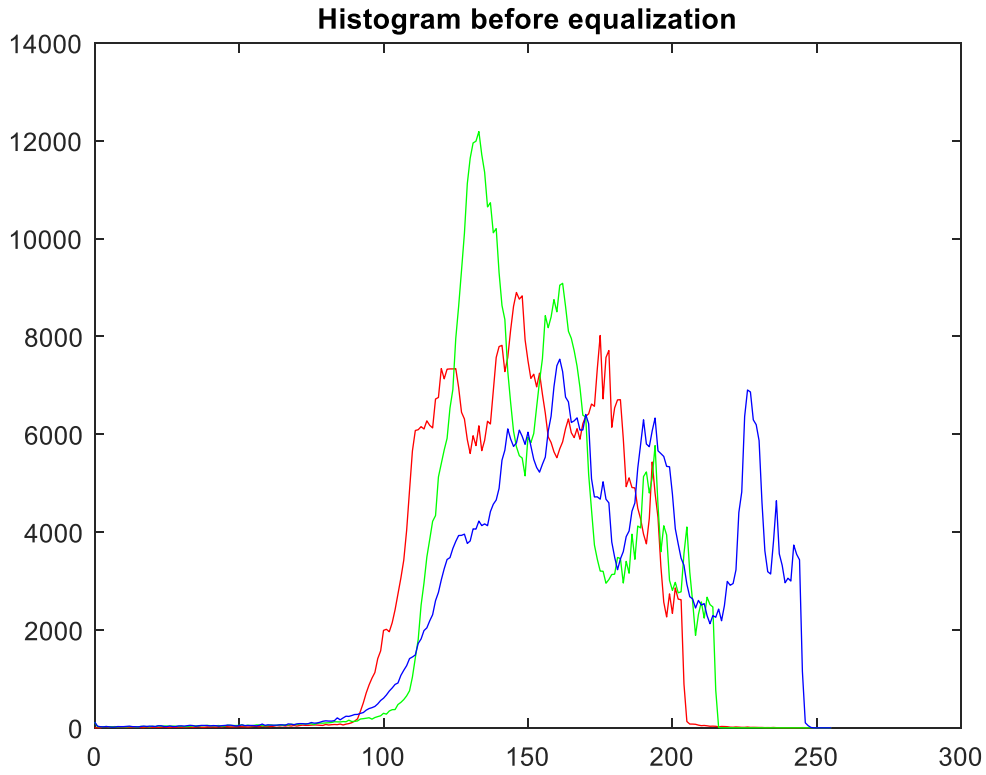
RGB Image-Hue Equalized

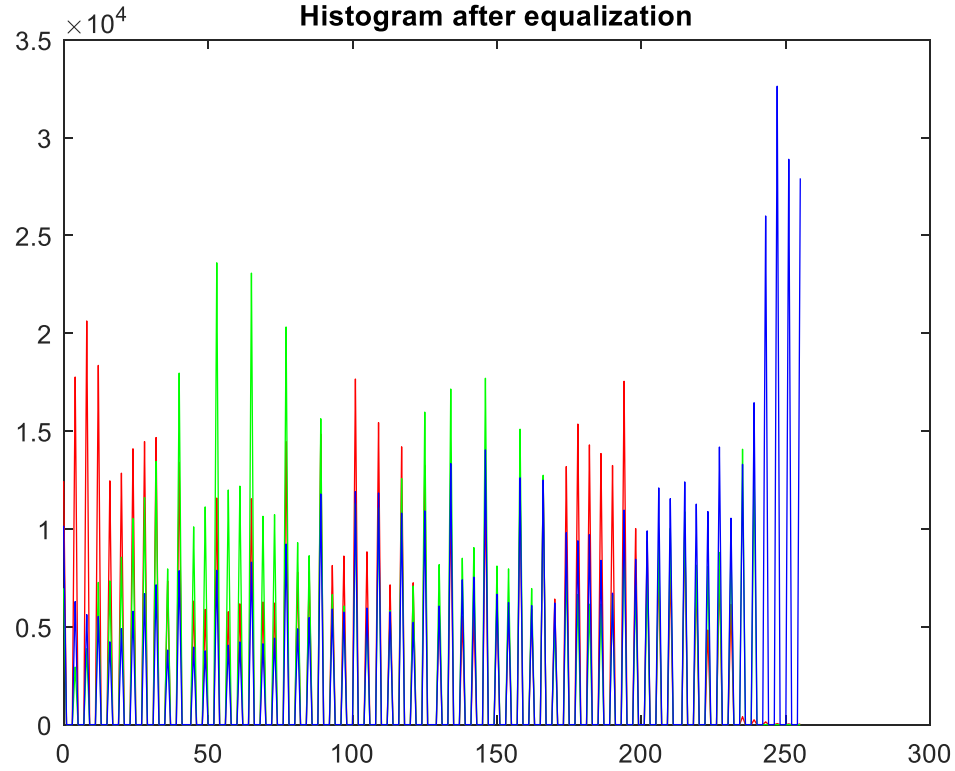
RGB Image-Saturation Equalized

RGB Image-Intensity Equalized

RGB Image-HSI Equalized

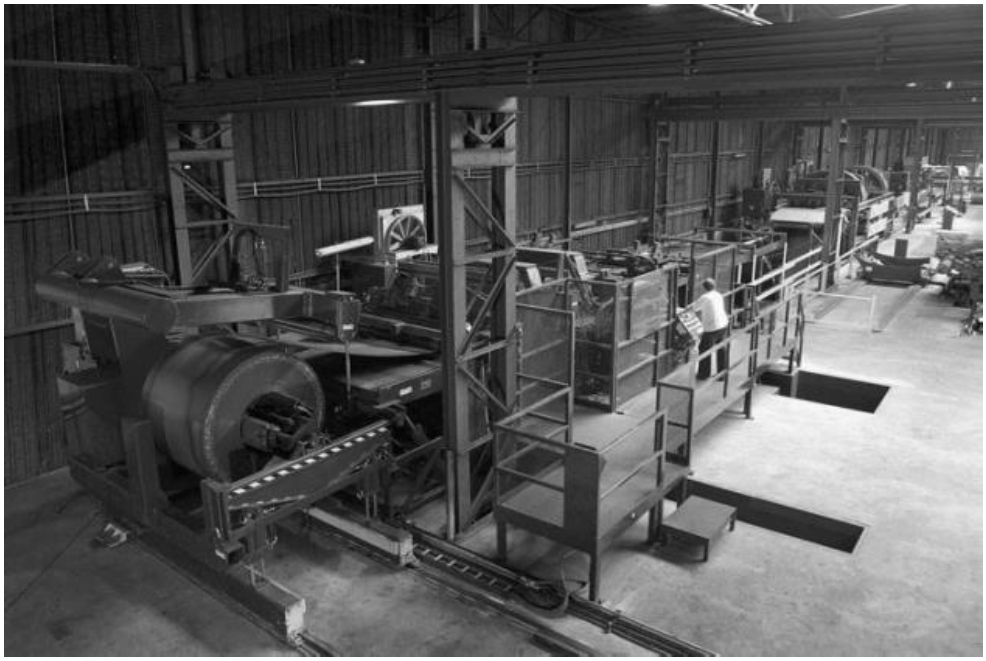
RGB histograms before and after histogram equalization:


Histogram before equalization
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250
300


Histogram after equalization
×10^4
3.5
3
2.5
2
1.5
1
0.5
0
0
50
100
150
200
250
300

Άσκηση 5: Αποκατάσταση Εικόνας

Original picture



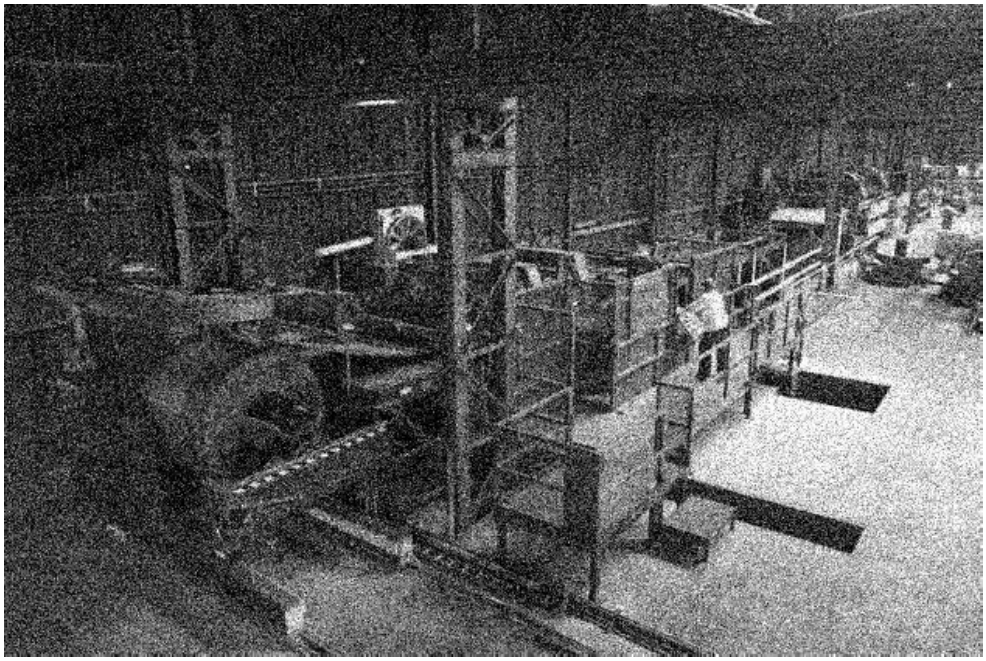
Κάνοντας μια διαδικασία σαν την παρακάτω βρήκα ότι για να έχω SNR 10dB πρέπει το variance του white gaussian noise να είναι περίπου 0.02. Με αυτό το variance το SNR είναι 10.08XXX, όπου είναι ικανοποιητικά κοντά στο 10.

```
%% Finding required variance for 10db noised image
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\factory.jpg');
f = rgb2gray(f);
f1 = imnoise(f,'gaussian', 0, 0.02);
noise = double(f1) - double(f);
SNR = snr(double(f1), noise);
```

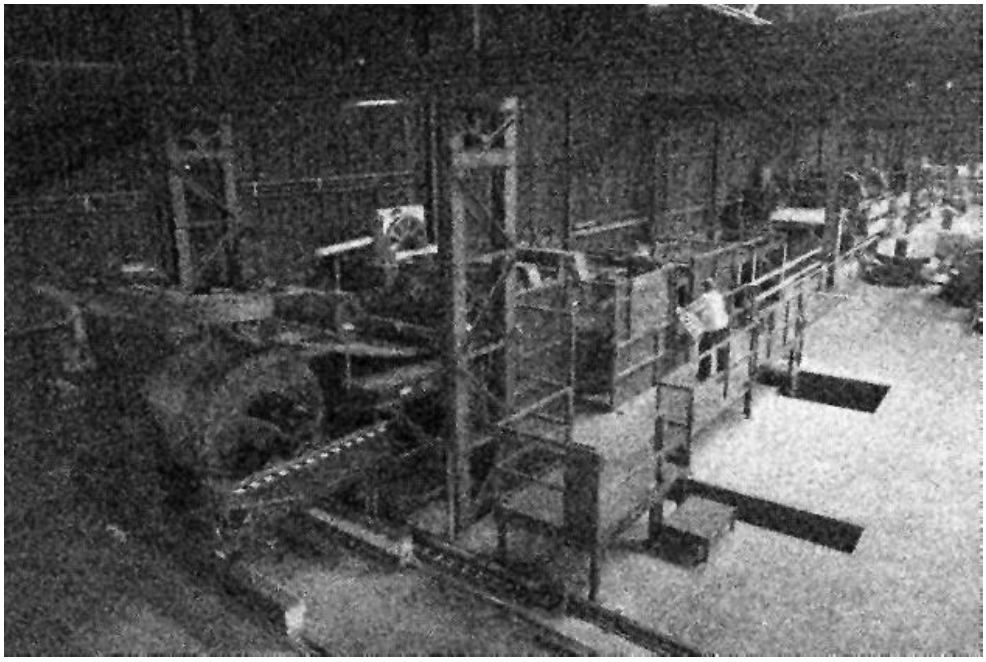
Removing white gaussian noise:

Εδώ έκανα μια προσπάθεια να φτιάξω το δικό μου weiner filter(WeinerFilter.m αρχείο) αλλά επειδή έβγαζε πολύ μεγάλο MSE συγκριτικά με το weiner2 του matlab συνέχισα με την weiner2 συνάρτηση του matlab.

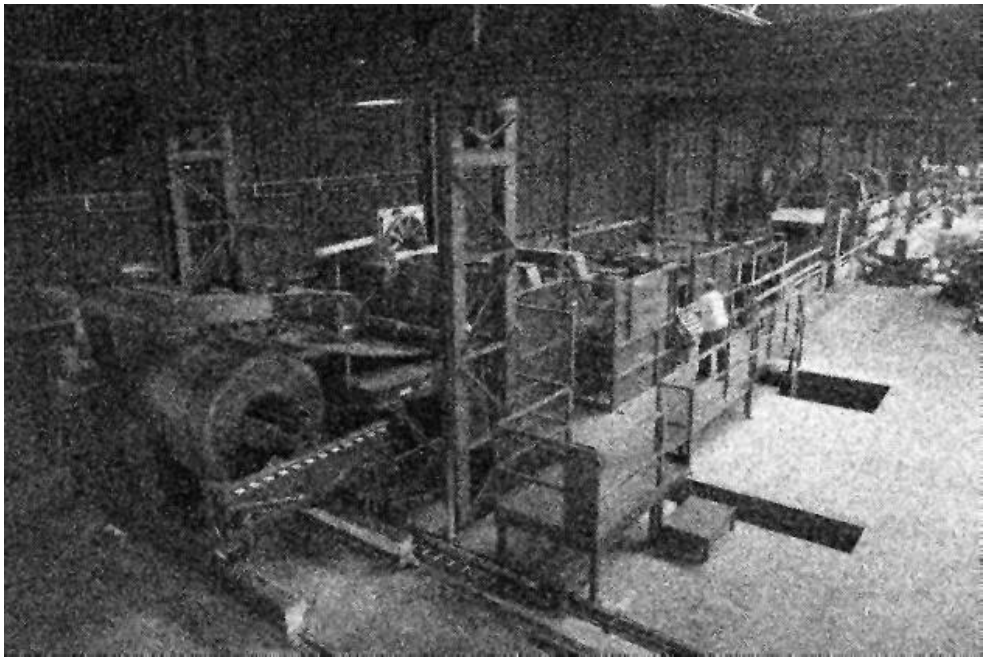
White gaussian noiseed image with σ = 0.02



Removal of white gaussian noise knowing σ = 0.02



Removal of white gaussian noise WITHOUT knowledge of σ



Συγκεκριμένα για την αφαίρεση white gaussian noise χωρίς να ξέρουμε το σ αυτό που θα έκανα για να ένα estimation είναι να βρώ το mean(vr), όπου vr είναι ο πίνακας με τα variance που υπολογίζω στην WeinerFilter.m . Επειδή βέβαια δεν βγάζει ικανοποιητικά αποτελέσματα χρησιμοποιώ `[f3 n] = wiener2(f1,[3 3]);`.

Το n είναι ένα estimation του θορύβου που υπάρχει στην f1 και υπολογίζεται με ακριβώς τον ίδιο τρόπο, δηλαδή βρίσκοντας το mean των variance. Για το συγκεκριμένο παράδειγμα το estimation είναι 0.018 ενώ το πραγματικό είναι 0.02 άρα έχω ένα αρκετά ικανοποιητικό estimation.

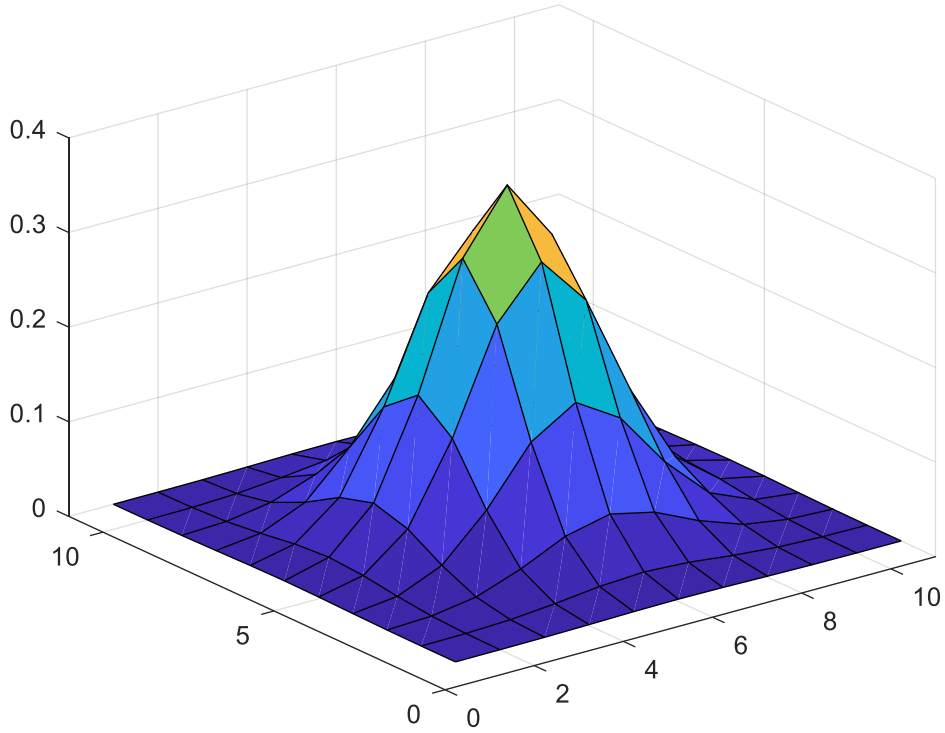
Τα αποτελέσματα είναι σχεδόν ίδια στο μάτι και πολύ ικανοποιητικά στο χαρτί. Συγκεκριμένα αν ξέρω το σ τότε MSE = 234.7052 ενώ με το estimation MSE = 247.9645.

PSF:

PSF show

```
Y = psf(f);
Z = zeros(11);
Z(6,6) = 1;
Y1 = psf(Z);     %Kroustikh apokrish
X = conv2(Y,Z)
```

Impulse response:



Άσκηση 6: Ανίχνευση Ακμών

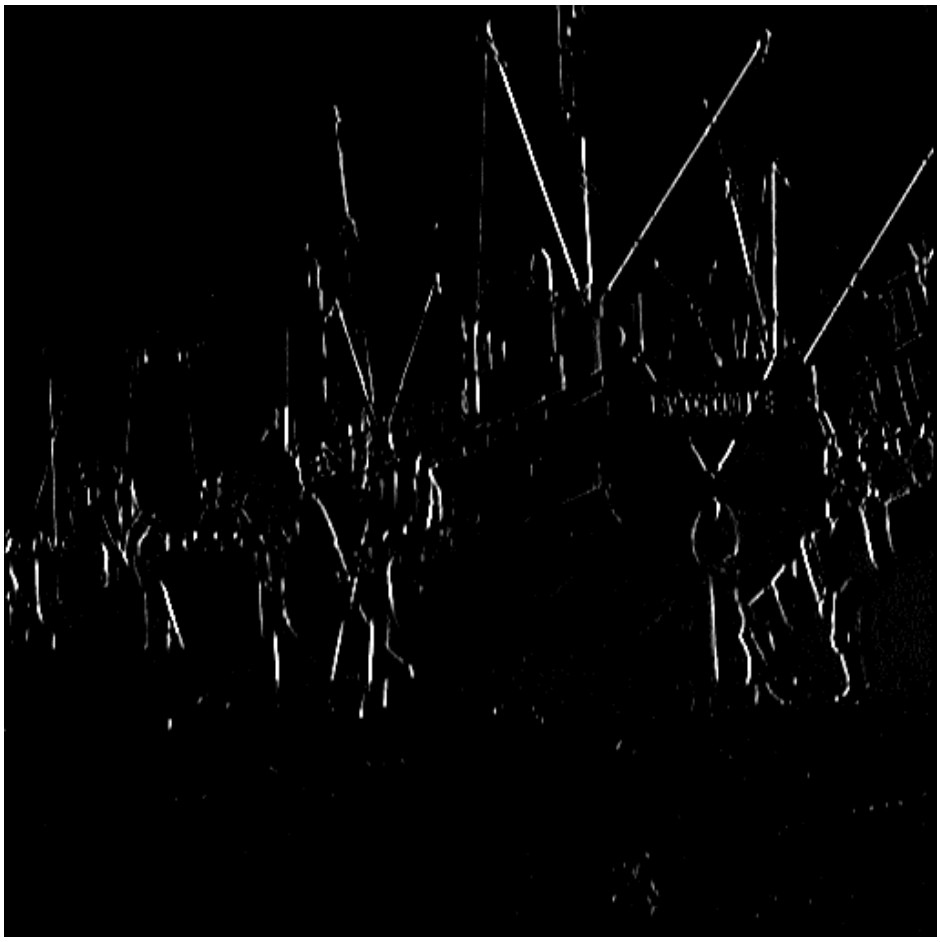
Using Sobel and Robert masks for edge detection:

Έχω δημιουργήσει τις συναρτήσεις SobelVer, SobelHor, SobelUL, SobelDL, RobertDL, RobertDR όπου κάνουν apply τις αντίστοιχες μάσκες που περιγράφει το όνομα τους. Θα ήθελα να σημειώσω εδώ ότι όταν κάνω imread το boat.tiff για κάποιο λόγο γυρνάει 512x512x4 όπου είναι 4 φορές η ίδια εικόνα και γ ’αυτό κάνω το `f = f(:,:,1);`

Sample code:

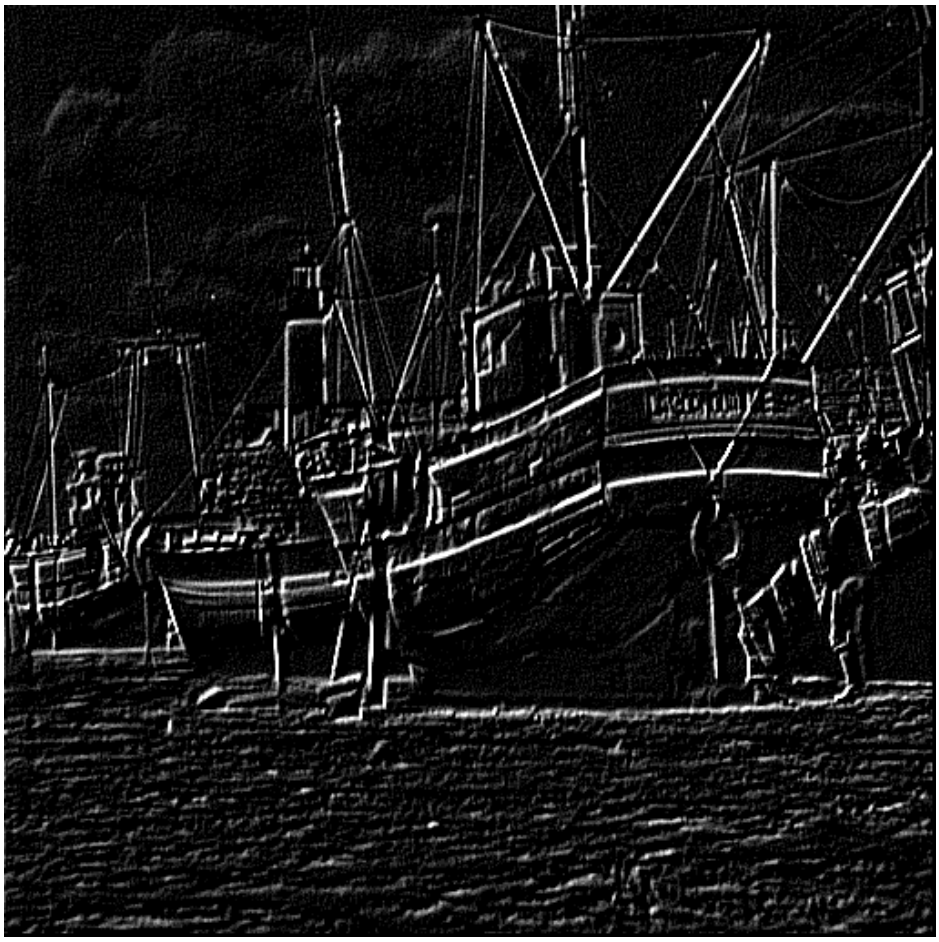
```
clear;
f =
imread('C:\Users\OrestisDrow\Desktop\I
mageProccessing\DIP_MEROS_A\Images\boa
t.tiff');
f = f(:,:,1);
f = SobelVer(f);
imshow(f,[0 255]);
```

Horizontal Sobel:

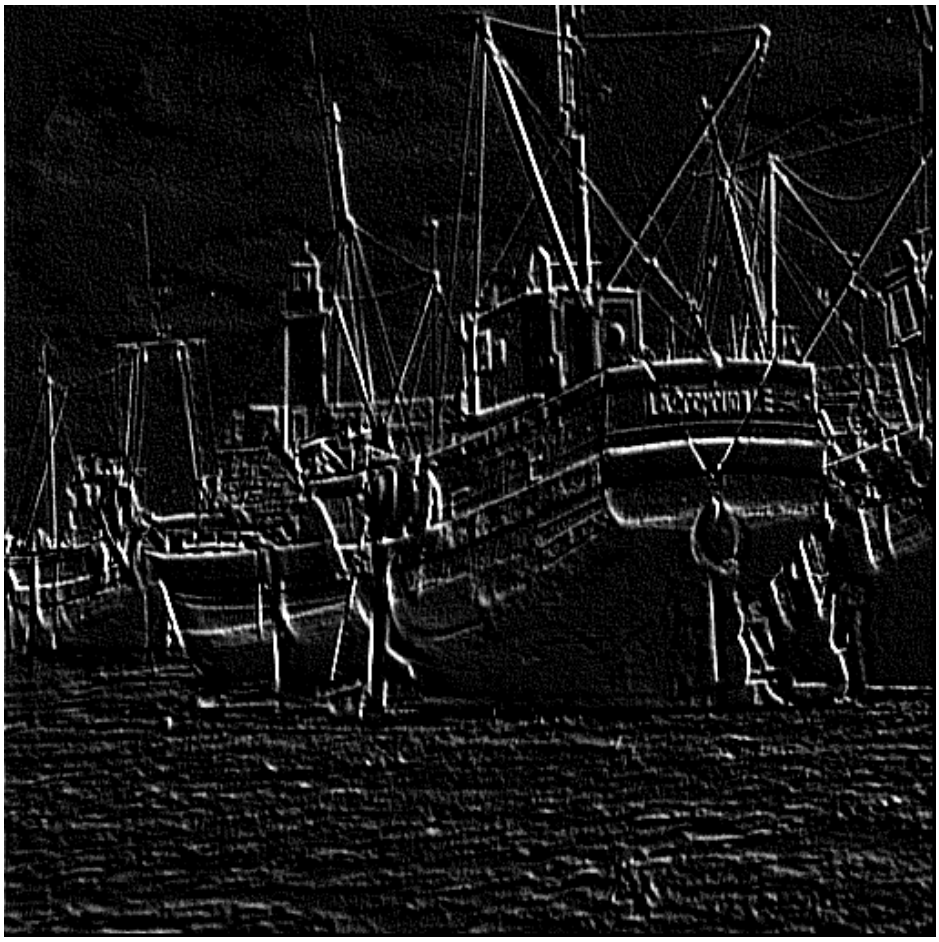


Vertical Sobel:

SobelDL:



SobelUL:

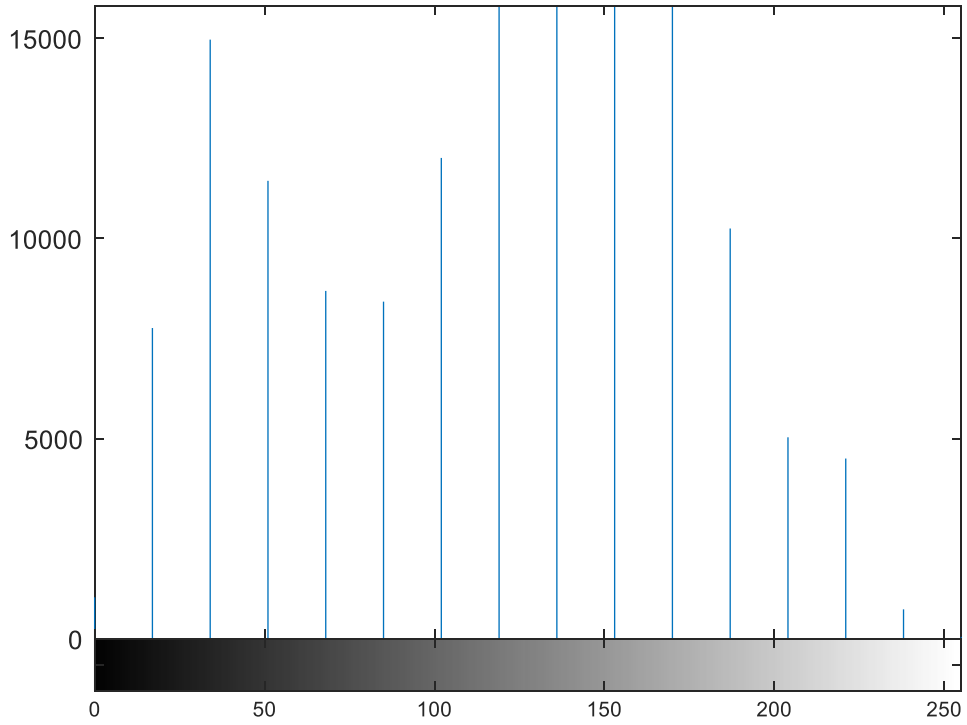


RobertDL:

RobertDR:

Total Threshold for better results:

Image histogram:



Διαισθητικά θέλουμε να βρούμε τη τιμή εκείνη όπου το 50% των pixel θα είναι από πάνω και το 50% από κάτω. Από το ιστόγραμμα βλέπουμε ότι αυτή η τιμή θα είναι κάπου στο βαρύκεντρο του σχήματος άρα με λίγο ψάξιμο βρήκα ότι:

```
>> numel(f(f>152))

ans =

      121330

>> numel(f(f<152))

ans =

      140814
```

Έχω κάνει την συνάρτηση **Edge** συνάρτηση η οποία βρίσκει τα max values των μασκών sobel και κάνει και Thresholding αναλόγως με το threshold που θα του δώσω. Το αποτέλεσμα φαίνεται παρακάτω:

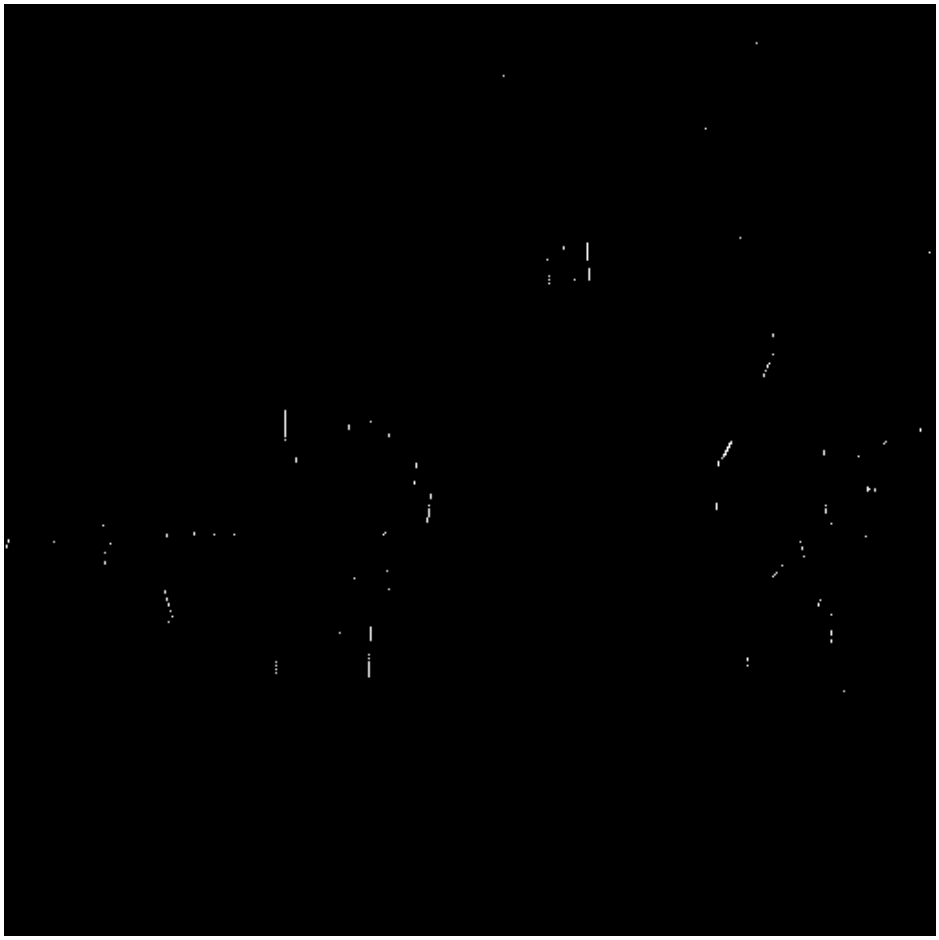
```
%% Edge detection
clear;
f = imread('C:\Users\OrestisDrow\Desktop\ImageProccessing\DIP_MEROS_A\Images\boat.tiff');
f = f(:,:,1);
f1 = SobelHor(f);
f2 = SobelVer(f);
f3 = SobelUL(f);
f4 = SobelDL(f);
%f5 = RobertDL(f);
%f6 = RobertDR(f);
%imhist(f);
%imshow(f,[0 255]);

%% Thresholding
[f_thres, f_mask] = EdgeSobel(f1,f2,f3,f4,152);
%[f_thres, f_mask] = EdgeRobert(f5,f6,152);


imshow(f_mask, [0 255]);
```

Max Sobel mask:

After threshold:

Maximized Robert:

After threshold:



Συμπεράσματα:

Η μάσκα sobel βγάζει πολύ πιο ικανοποιητικά αποτελέσματα απότι η Robert.