ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ - ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

"Ψηφιακή Επεξεργασία και Ανάλυση Εικόνας"

Ακαδημαϊκό Έτος 2020-21 (Εαρινό Εξάμηνο)

Εργαστηριακές Ασκήσεις - Μέρος Α΄

Βασιλική Στάμου AM: 1059543

4ο έτος

Διευκρινήσεις σχετικά με την εργασία και την αναφορά

- Η αναφορά περιλαμβάνει για κάθε μια από τις έξι ασκήσεις μια σύντομη περιγραφή της εκάστοτε τεχνικής , επαρκή σχολιασμό των αποτελεσμάτων τα οποία προκύπτουν σε κάθε περίπτωση και τον κώδικα ο οποίος γράφτηκε για την υλοποίηση των εκάστοτε ασκήσεων.
- Περιβάλλον ανάπτυξης για όλες τις ασκήσεις υπήρξε η Matlab.
- Στην άσκηση 2 , η εικόνα flower.mat είναι ασπρόμαυρη , οπότε η όλη επεξεργασία έγινε σε grayscale και όχι RBG/HSI.
- Στην άσκηση 5 , αντί του χρωματικού συστήματος HSI χρησιμοποιήθηκε το HSV επειδή έκανε την επεξεργασία πιο εύκολη και τον κώδικα πιο ευανάγνωστο.

Πίνακας Περιεχομένων

1. Φιλτράρισμα στο πεδίο συχνοτήτων

Σύντομη περιγραφή τεχνικής

Για την υλοποίηση του 2D-DFT και του αντίστροφου μετασχηματισμού χρησιμοποιήθηκε η συνάρτηση fft() / ifft() της matlab μιας και καθιστά την όλη διαδικασία πιο γρήγορη.

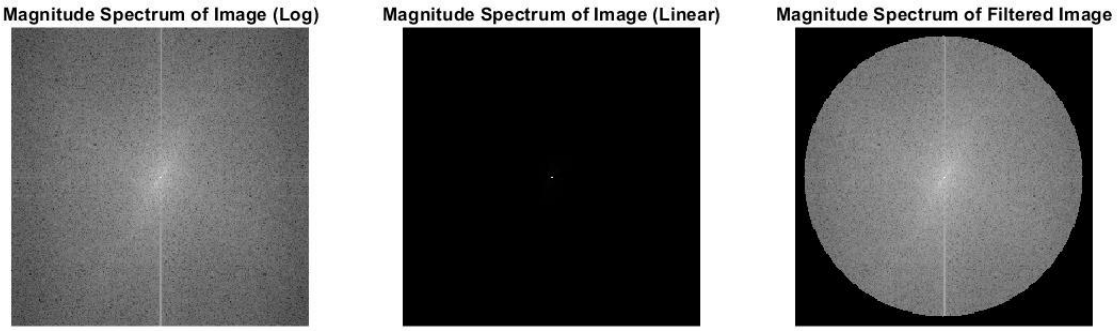
Η διαδικασία που ακολουθήσαμε για τον 2D-DFT είναι η εξής : Εφαρμόζουμε FFT σε κάθε γραμμή της εικόνας , έπειτα, εφαρμόζουμε FFT σε κάθε στήλη της εικόνας του προηγούμενου βήματος.

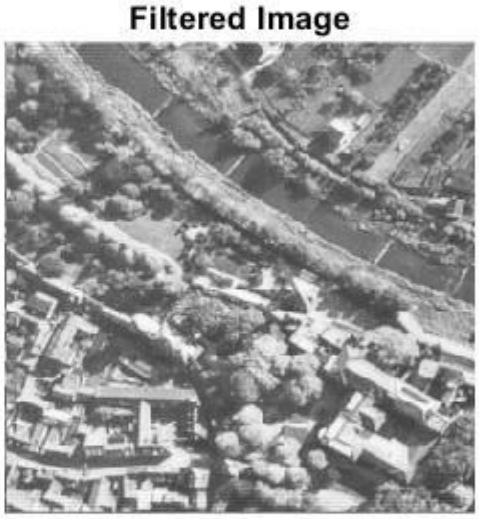
Η διαδικασία που ακολουθήσαμε για τον 2D-IDFT είναι η εξής : Εφαρμόζουμε IFFT σε κάθε στήλη της εικόνας , έπειτα, εφαρμόζουμε IFFT σε κάθε γραμμή της εικόνας του προηγούμενου βήματος.

Για το φιλτράρισμα της εικόνας στο πεδίο συχνοτήτων έγινε χρήση ιδανικού και gaussian κατωπερατού φίλτρου και υλοποιήθηκαν σαν συνάρτηση. Το ιδανικό φίλτρο χρησιμοποιήθηκε κυρίως για να δούμε το ringing effect .

Σχολιασμός αποτελεσμάτων

Ιδανικό κατωπερατό φίλτρο :

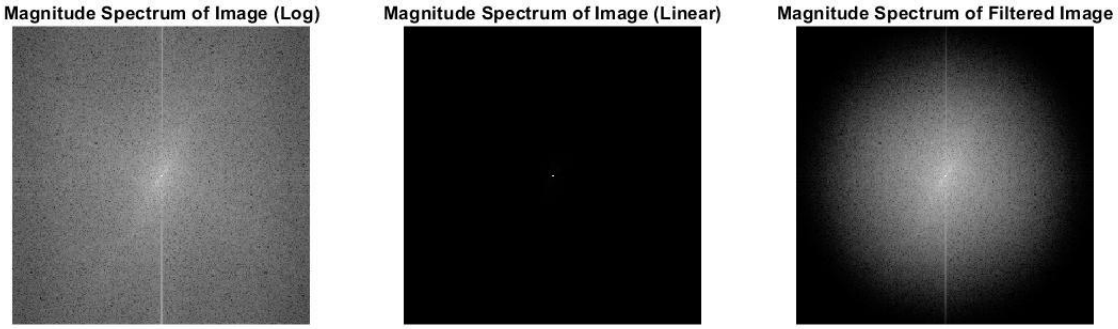




Παρότι το 90% της ενέργειας της εικόνας (μπορεί και περισσότερο) βρίσκεται εντός του κύκλου/δίσκου δεν έχω πολύ καλά αποτελέσματα. Αυτό το 10% που έχω αποκόψει , δεν είναι οποιοδήποτε 10% , είναι το 10% των περιγραμμάτων , για αυτό τα περιγράμματα της εικόνας έχουν αλλοιωθεί. Επίσης έχω και το λεγόμενο ringing effect , αυτό ουσιαστικά είναι αποτέλεσμα της sinc συνάρτησης και είναι γνωστό ως φαινόμενο Gibbs . Δημιουργείται λόγω απότομων μεταβάσεων , όπως τα περιγράμματα στην περίπτωση μας .Τα προβλήματα αυτά βελτιώνονται αν αυξήσω την συχνότητα αποκοπής αλλά τότε θα χάνω σε SNR.

Ένας τρόπος για να αντιμετωπίσουμε αυτά τα θέματα είναι να βάλουμε φίλτρο το οποίο αποσβένεται πιο ομαλά.

Gaussian κατωπερατό φίλτρο :



Όπως βλέπουμε , το gaussian φίλτρο , έχει ομαλή μετάβαση από την ζώνη διέλευσης στην ζώνη αποκοπής , αυτό έχει σαν αποτέλεσμα η λίγη ισχύς που κρατάω από τις υψηλές συχνότητες να διατηρεί πολύ καλύτερα τα περιγράμματα και να μην έχω ringing effect.

Κώδικας

```
%Read image
image=imread('aerial.tiff');

%Set pixel range to [0:255]
img = uint8(255*mat2gray(image));


% Perform 2D-FFT
[row,col]=size(img);
FFT_row = zeros(row);
FFT_col = zeros(col);
%Perform FFT on each row
for i=1:row
FFT_row(i,:) = fft(double(img(i,:)));
end
%Perform FFT on each column
for i=1:col
FFT_col(:,i) = fft(FFT_row(:,i));
end

%DFT of image
F = FFT_col;

%Shifting spectrum to centre
Fs=fftshift(F);


%Getting H(u,v)
H=myfilter2D('gaussianLPF',row,col,30);
%H=myfilter2D('idealLPF',row,col,120);

%Filtering
Fsf=Fs.*H;


% Perform 2D-IFFT
[row,col]=size(Fsf);
IFFT_row = zeros(row);
IFFT_col = zeros(col);
Fsfs = fftshift(Fsf);
%Perform IFFT on each column
for i=1:col
IFFT_col(:,i) = ifft(Fsfs(:,i));
end
%Perform IFFT on each row
for i=1:col
IFFT_row(i,:) = ifft(IFFT_col(i,:));
end
fimg = IFFT_row;
imgr=uint8(real(fimg));


% Plotting
figure,subplot(131);imshow(log(1+abs(Fs)),[]); title('Magnitude Spectrum of Image
(Log)')
subplot(132);imshow(abs(Fs),[]); title('Magnitude Spectrum of Image (Linear)')
subplot(133);imshow(log(1+abs(Fsf)),[]); title('Magnitude Spectrum of Filtered
Image')
```

```
figure,subplot(121);imshow(img); title('Input Image')
subplot(122);imshow(imgr,[]); title('Filtered Image')
```

```
%Function to implement filter's frequency response H(u,v)
function H=myfilter2D(type,M,N,D0,n) %myfilter2D(type,row,col,radius,order)

P=floor(M/2); Q=floor(N/2); %Getting centre of image
D=zeros(M,N); H=zeros(M,N);

switch type
    case 'idealLPF'
        for u=1:M
            for v=1:N
            D(u,v)=sqrt((u-P)^2+(v-Q)^2); %Distance from image center
            if D(u,v)<=D0
                H(u,v)=1;
            end
            end
        end

    case 'butterLPF'
        for u=1:M
            for v=1:N
                D(u,v)=sqrt((u-P)^2+(v-Q)^2);
                H(u,v)=1/(1+(D(u,v)/D0)^(2*n));
            end
        end
    case 'gaussianLPF'
        for u=1:M
            for v=1:N
                D(u,v)=sqrt((u-P)^2+(v-Q)^2);
                H(u,v)=exp(-((D(u,v)^2)/(2*D0^2)));
            end
        end
end
```

2. Συμπίεση Εικόνας με χρήση μετασχηματισμού DCT

Σύντομη περιγραφή τεχνικής

Για την **μέθοδο ζώνης** οι συντελεστές επιλέχθηκαν με κριτήριο την μέγιστη διακύμανση. Η κωδικοποίηση ζώνης στηρίζεται σε μια έννοια της θεωρίας πληροφορίας σύμφωνα με την οποία μπορούμε να θεωρήσουμε την πληροφορία ως αβεβαιότητα. Επομένως , οι συντελεστές του μετασχηματισμού που χαρακτηρίζονται από την μέγιστη διακύμανση , είναι και αυτοί που μεταφέρουν το μεγαλύτερο μέρος της πληροφορίας της εικόνας και αυτοί που θα πρέπει να διατηρηθούν κατά τη διαδικασία της κωδικοποίησης. Η μάσκα ζώνης κατασκευάζεται τοποθετώντας άσσους στις θέσεις μέγιστης διακύμανσης και μηδενικές τιμές σε όλες τις άλλες θέσεις.

Για την **μέθοδο κατωφλίου** οι συντελεστές επιλέχθηκαν με κριτήριο το μέγιστο μέτρο. Η έννοια στην οποία στηρίζεται αυτή η μέθοδος, είναι πως για κάθε υπο-εικόνα , οι συντελεστές του μετασχηματισμού που χαρακτηρίζονται από το μεγαλύτερο μέτρο έχουν και την μεγαλύτερη συνεισφορά στην ποιότητα της ανακατασκευαζόμενης εικόνας. . Η μάσκα ζώνης κατασκευάζεται τοποθετώντας άσσους στις θέσεις μέγιστου μέτρου και μηδενικές τιμές σε όλες τις άλλες θέσεις.

Η κωδικοποίηση ζώνης υλοποιείται με μια απλή μάσκα η οποία είναι ίδια για όλες τις υποεικόνες , ωστόσο η κωδικοποίηση κατωφλίου είναι από τη φύση της προσαρμοζόμενη , υπό την έννοια πως η θέση των συντελεστών του μετασχηματισμού που διατηρούνται για κάθε υπο-εικόνα , μεταβάλλονται από τη μια υπο-εικόνα στην άλλη. Βέβαια , ένας από τους τρείς βασικούς τρόπους για να εφαρμόσουμε την κωδικοποίηση κατωφλίου είναι να εφαρμόσουμε ένα συγκεκριμένο καθολικό κατώφλι σε όλες τις υποεικόνες , και αυτό υλοποιήθηκε στα πλαίσια της εργασίας λόγο απλότητας σχεδιασμού.

Σχολιασμός αποτελεσμάτων

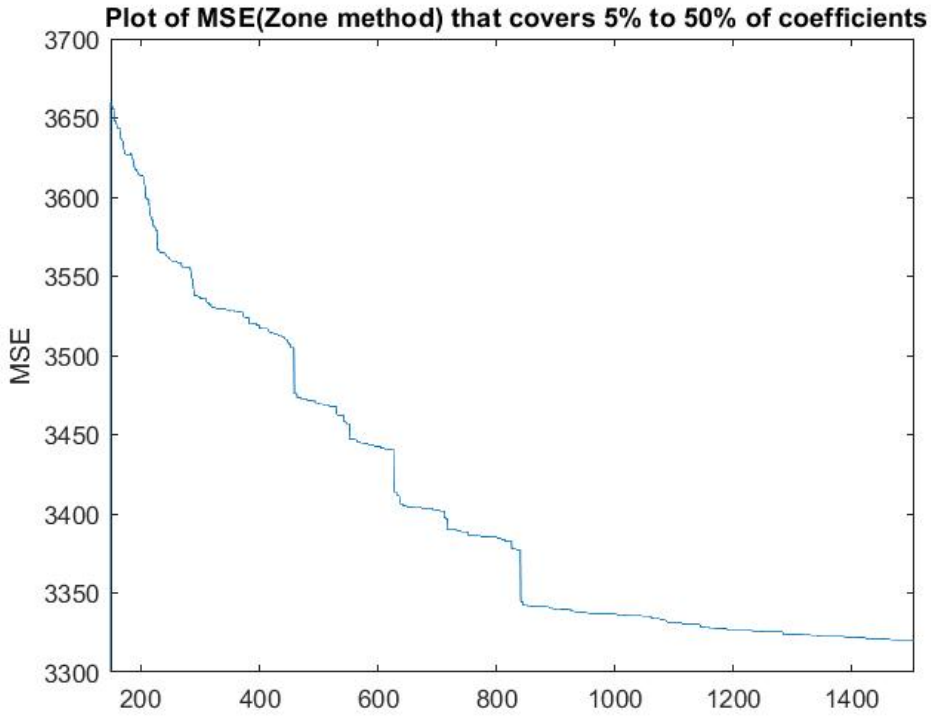
Με την μέθοδο ζώνης *:*

Χρησιμοποιήθηκε το **5%** των συντελεστών .

Με την μέθοδο ζώνης *:*

Χρησιμοποιήθηκε το **50%** των συντελεστών .

Μέσο τετραγωνικό σφάλμα για την μέθοδο ζώνης : MSE(5%) = 3660 MSE(50%)= 3325 , βελτιώθηκε κατά 335 μονάδες



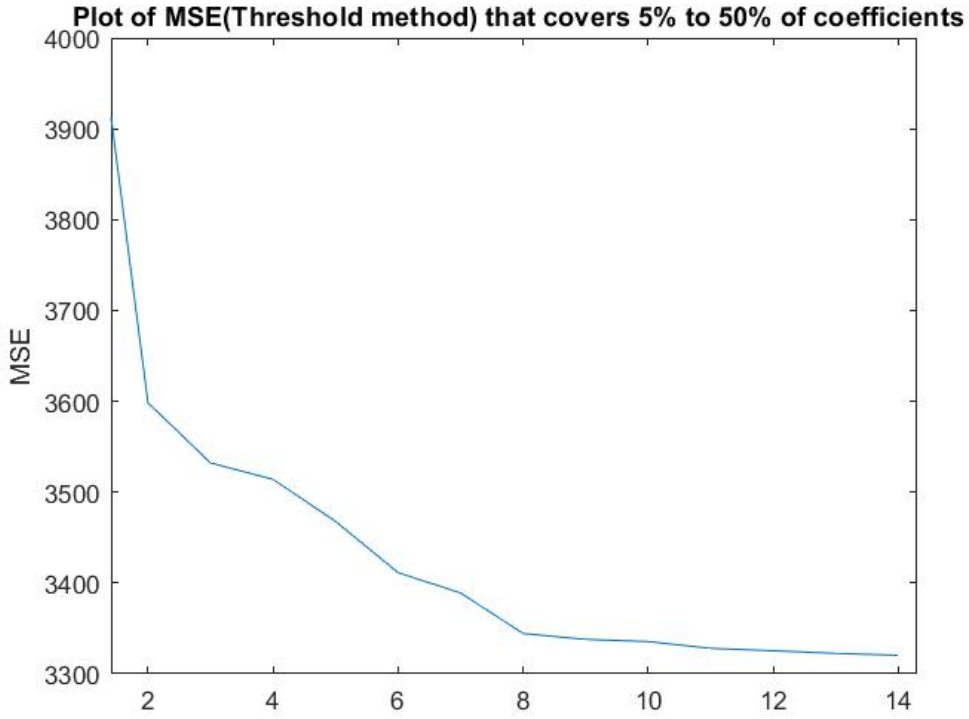
Με την μέθοδο κατωφλίου *:*

Χρησιμοποιήθηκε το **5%** των συντελεστών .

Με την μέθοδο κατωφλίου *:*

Χρησιμοποιήθηκε το **50%** των συντελεστών .

Μέσο τετραγωνικό σφάλμα για την μέθοδο κατωφλίου : MSE(5%) = 3900 MSE(50%)= 3310 , βελτιώθηκε κατά 590 μονάδες



Παρατηρούμε πως κρατώντας το 5% των συντελεστών καλύτερα αποτελέσματα έχουμε με την μέθοδο ζώνης ενώ κρατώντας το 50% των συντελεστών καλύτερα αποτελέσματα έχουμε με την μέθοδο κατωφλίου η οποία παρουσιάζει μεγαλύτερους ρυθμούς μείωσης του MSE.

Κώδικας

Για την μέθοδο ζώνης :

```
%Image Compression using DCT Transform- Zone Method
load flower.mat;
img= imshow(flower);
imsave(img);
%Read the input image
img=double(imread('flower.jpg'));

T = dctmtx(32); %Getting 32x32 transformation matrix
dct = @(block_struct)T*(block_struct.data)*T'; %Defining DCT operation
C = blockproc(img,[32 32],dct); %Finding DCT using block processing

for g=149:1505;
ct = stdfilt(img,true(33));
ct=(ct).^2;
Threshold=mean(ct(:));


for i=1:256
    for j=1:256
            if abs(C(i,j)) < (Threshold/g)
                ct(i,j) = 0;
            end
    end
end
%g=149 #coeff=52
%g=1505 #coeff=512

 mask= zeros(32,32);
 for i=1:32
    for j=1:32
            if ct(i,j) ~= 0
                mask(i,j) = 1;
            end
    end
 end
m = nonzeros(mask);
%Truncating DCT coefficients
Ct = blockproc(C,[32 32],@(block_struct) (mask .* block_struct.data));

invdct = @(block_struct)T' *(block_struct.data)*T'; %Defining IDCT operation
invC = blockproc(Ct,[32 32], invdct); %Finding Inverse DCT
err(g) = immse(img,invC);
plot(err)
xlim([149 1505])
title(' Plot of MSE(Zone method) that covers 5% to 50% of coefficients')
ylabel('MSE')
end
%Displaying Images
figure,imshowpair(uint8(img),uint8(invC),'montage');title('Original Grayscale Image
(Left) and DCT Compressed Image (Right)');
```

Για την μέθοδο κατωφλίου :

```
%Image Compression using DCT Transform- Threshold Method
load flower.mat;
img= imshow(flower);
imsave(img);
%Read the input image
img=double(imread('flower.jpg'));

T = dctmtx(32); %Getting 32x32 transformation matrix
dct = @(block_struct)T*(block_struct.data)*T'; %Defining DCT operation
C = blockproc(img,[32 32],dct); %Finding DCT using block processing

for g=1:14.3;
ct = C;
Threshold=mean(abs(C(:)));

for i=1:256
    for j=1:256
            if abs(C(i,j)) < (Threshold/g)
                ct(i,j) = 0;
            end
    end
end
%g=14.3 #coeff = 512
%g=1.41 #coeff= 52

 mask= zeros(32,32);
 for i=1:32
    for j=1:32
            if ct(i,j) ~= 0
                mask(i,j) = 1;
            end
    end
 end
m = nonzeros(mask);
%Truncating DCT coefficients
Ct = blockproc(C,[32 32],@(block_struct) (mask .* block_struct.data));

invdct = @(block_struct)T' *(block_struct.data)*T'; %Defining IDCT operation
invC = blockproc(Ct,[32 32], invdct); %Finding Inverse DCT
err(g) = immse(img,invC);
plot(err)
xlim([1.41 14.3])
title(' Plot of MSE(Threshold method) that covers 5% to 50% of coefficients')
ylabel('MSE')
end

%Displaying Images
figure,imshowpair(uint8(img),uint8(invC),'montage');title('Original Grayscale Image
(Left) and DCT Compressed Image (Right)');
```

3. Βελτίωση εικόνας – Φιλτράρισμα Θορύβου

Σύντομη περιγραφή τεχνικής

Για την υλοποίηση του **moving average filter** χρησιμοποιήθηκε η συνάρτηση imfilter() της matlab με παράθυρο 3x3.Το φίλτρο κινούμενου μέσου όρου προκαλεί την εξομάλυνση των τοπικών μεταβολών στην εικόνα και μείωση του θορύβου ως αποτέλεσμα της θόλωσης.

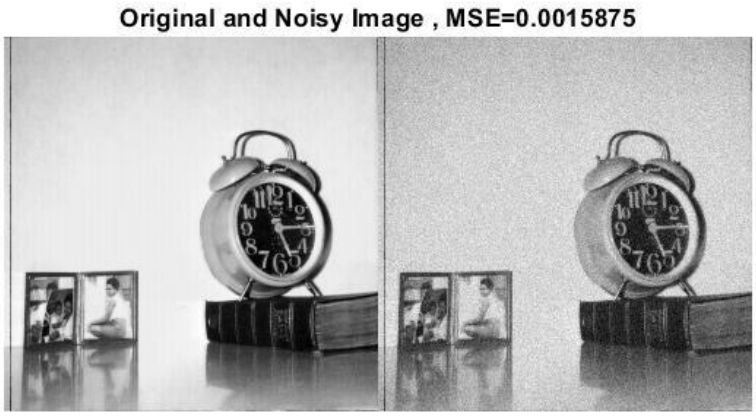
Για την υλοποίηση του **median filter** χρησιμοποιήθηκε η συνάρτηση medfilt2() της matlab με παράθυρο 5x5.

Για να **παράγουμε Gaussian noise** χρησιμοποιούμε την συνάρτηση randn() της matlab που παράγει αριθμούς σύμφωνα με ομοιόμορφη κατανομή και τους πολλαπλασιάζουμε την διασπορά του θορύβου την οποία υπολογίζουμε από το SNR που μας δίνεται.

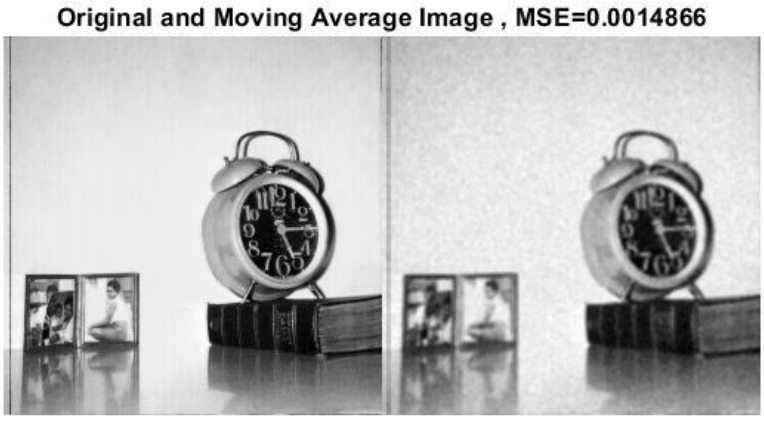
Για να **παράγουμε Salt and Pepper noise** χρησιμοποιούμε την συνάρτηση imnoise() της matlab.

Σχολιασμός αποτελεσμάτων

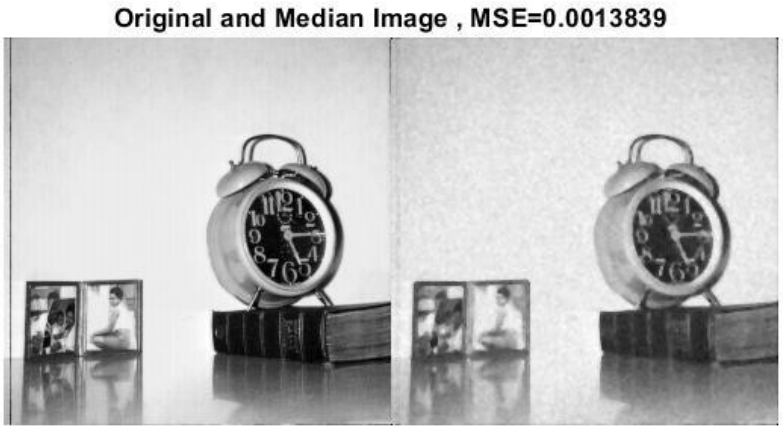
Για το πρώτο ερώτημα που προσθέτουμε λευκό Gaussian θόρυβο τα αποτελέσματα παρουσιάζονται παρακάτω :



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας μετά την προσθήκη **Gaussian θορύβου** είναι περίπου **0.0016** .



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την **εφαρμογή φίλτρου κινητού μέσου όρου** είναι περίπου **0.0015** .

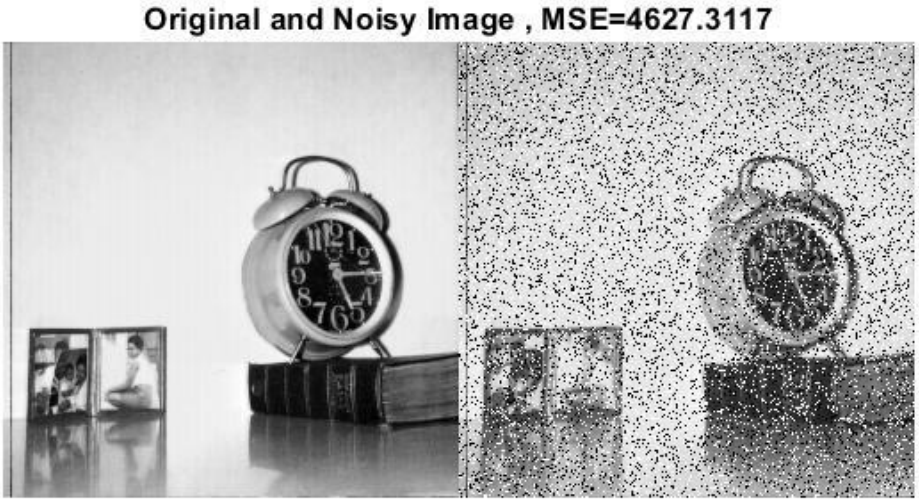


Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την εφαρμογή **φίλτρου μεσαίου** είναι περίπου **0.0014** .

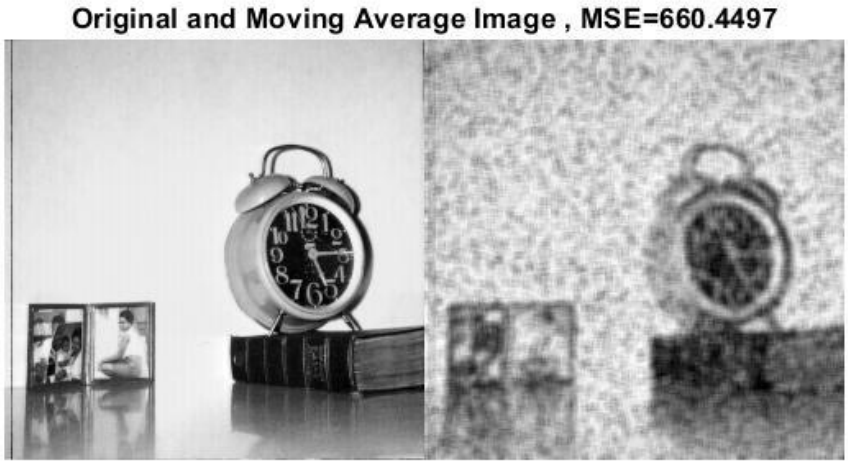
Παρατηρούμε πως και τα δύο φίλτρα έκαναν καλή δουλεία όσο αφορά στην μείωση των επιδράσεων του θορύβου. Το φίλτρο μεσαίου έκανε λίγο καλύτερα δουλειά καθαρίζοντας το υπόβαθρο , αν και το τίμημα που πληρώνουμε είναι η ελαφρά εκλέπτυνση και θόλωση των σκοτεινών περιοχών.

Μιλώντας γενικά , τα φίλτρα του κινούμενου μέσου όρου είναι κατάλληλα για τυχαίο θόρυβο όπως είναι ο θόρυβος Gauss.

Για το δεύτερο ερώτημα που προσθέτουμε κρουστικό θόρυβο τα αποτελέσματα παρουσιάζονται παρακάτω :



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας μετά την προσθήκη **κρουστικού θορύβου** είναι περίπου **4645** .



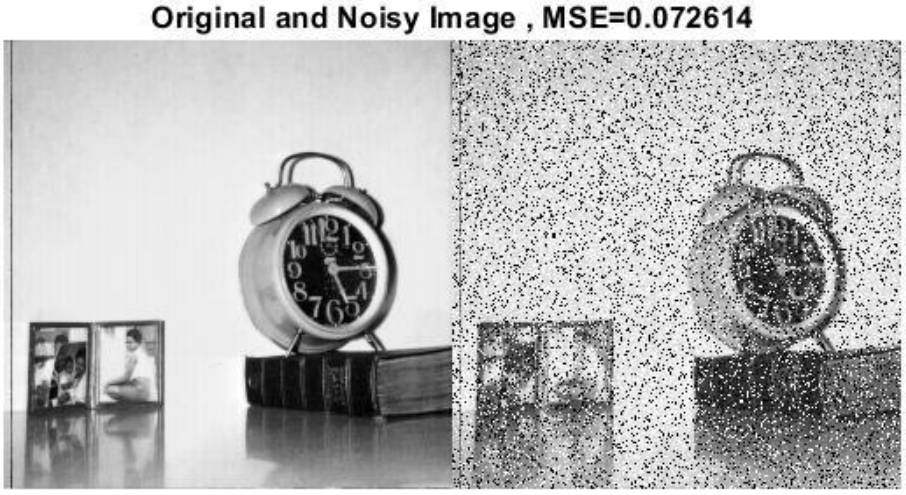
Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την **εφαρμογή φίλτρου κινητού μέσου όρου** είναι περίπου **660** .

Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την εφαρμογή **φίλτρου μεσαίου** είναι περίπου **264** .

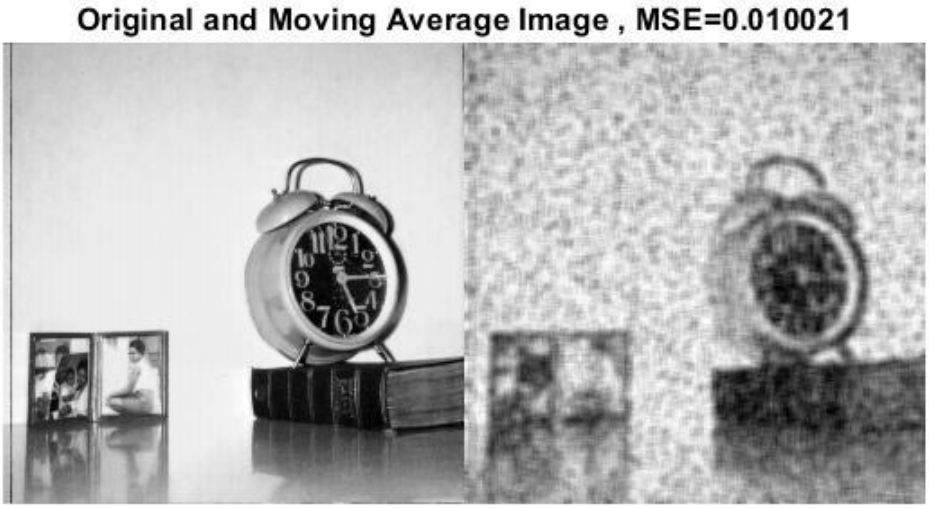
Όπως ήταν αναμενόμενο , το φίλτρο κινουμένου μέσου χαρακτηρίστηκε από φτωχή απόδοση , εξαιτίας της ύπαρξης κρουστικού θορύβου.

Από την άλλη πλευρά , το φίλτρο μεσαίου είχε πολύ καλύτερη απόδοση.

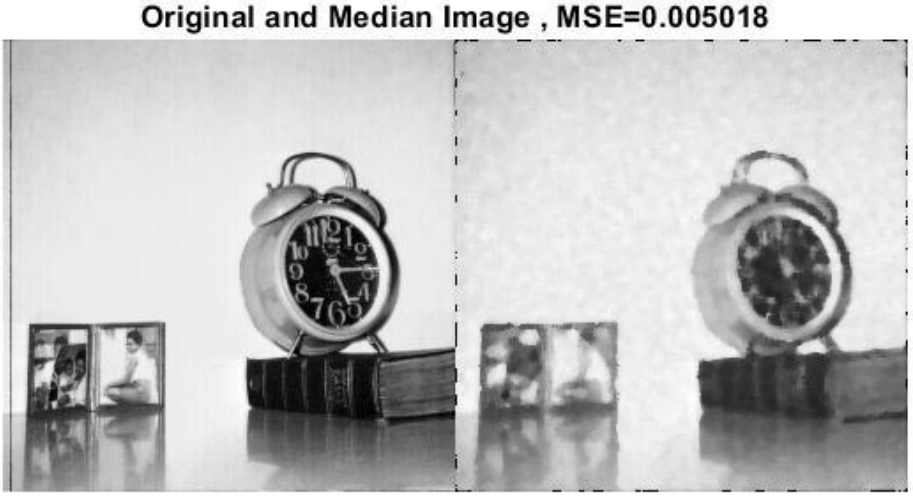
Για το τρίτο ερώτημα που προσθέτουμε λευκό Gaussian θόρυβο και κρουστικό θόρυβο τα αποτελέσματα παρουσιάζονται παρακάτω :



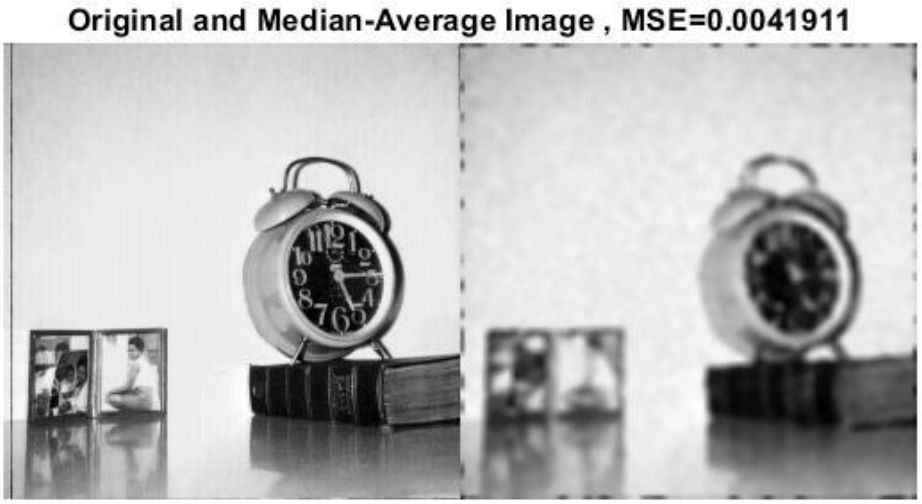
Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας μετά την προσθήκη **Gaussian και κρουστικού θορύβου** είναι περίπου **0.0726** .



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την **εφαρμογή φίλτρου κινητού μέσου όρου** είναι περίπου **0.0100** .



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την εφαρμογή **φίλτρου μεσαίου** είναι περίπου **0.0050** .



Το **μέσο τετραγωνικό σφάλμα** μεταξύ της **αρχικής εικόνας** και της εικόνας με θόρυβο μετά την εφαρμογή **πρώτα φίλτρου μεσαίου και έπειτα φίλτρου κινητού μέσου όρου** είναι περίπου **0.0042** .

Η προσθήκη δύο ειδών θορύβου συνιστά ένα υψηλό επίπεδο αλλοίωσης από θόρυβο και δικαιολογεί τη χρήση μεγαλύτερων φίλτρων . Οπότε σε όλα τα φίλτρα του ερωτήματος , οι διαστάσεις των φίλτρων είναι 5x5

Παρατηρούμε ότι η χρήση φίλτρου μεσαίου κάνει καλύτερη δουλειά από το φίλτρο κινουμένου μέσου όσον αφορά στην εξασθένηση της συνεισφοράς που οφείλεται στον θόρυβο . Παρόλα αυτά υπάρχουν ακόμα σημεία που περιέχουν θόρυβο . Μια δεύτερη διέλευση της εικόνας από το φίλτρο κινούμενου μέσου απομακρύνει και αυτά τα λίγα σημεία, αφήνοντας λίγα μόνο σημεία που μόλις μετά βίας είναι ορατά.

Κώδικας

Για το πρώτο ερώτημα :

```
%Read image
image_original = imread('clock.tiff');

%Estimate variance of input image
image_original = im2double(image_original);
var_image_original = std2(image_original)^2;

%Add white Gaussian noise , SNR=15db
SNRdB = 15;
sigma_noise = sqrt(var_image_original/10^(SNRdB/10));
noise = sigma_noise*randn(size(image_original));
image_with_noise = image_original+noise;
err0=immse(image_original,image_with_noise);
figure,imshowpair(image_original,image_with_noise,'montage');title("Original
and Noisy Image , MSE=" +err0);


%Moving Average Filter
windowSize = 3;
kernel = ones(windowSize, windowSize) / windowSize ^ 2;
image_filtered_motion_average = imfilter(image_with_noise, kernel,
'symmetric');
err1=immse(image_filtered_motion_average,image_original);
figure,imshowpair(image_original,image_filtered_motion_average,'montage');tit
le("Original and Moving Average Image , MSE=" +err1);

%Median Filter
image_filtered_median = medfilt2(image_with_noise);
err2=immse(image_filtered_median,image_original);
figure,imshowpair(image_original,image_filtered_median,'montage');title("Orig
inal and Median Image , MSE=" +err2);
```

Για το δεύτερο ερώτημα :

```
%Read image
image_original = imread('clock.tiff');

%Add Salt & Pepper Noise
image_with_noise = imnoise(image_original,'salt & pepper',0.2);

err0=immse(image_original,image_with_noise);
figure,imshowpair(image_original,image_with_noise,'montage');title("Original
and Noisy Image , MSE=" +err0);


%Moving Average Filter
windowSize = 5;
kernel = ones(windowSize, windowSize) / windowSize ^ 2;
image_filtered_motion_average = imfilter(image_with_noise, kernel,
'symmetric');

err1=immse(image_filtered_motion_average,image_original);
figure,imshowpair(image_original,image_filtered_motion_average,'montage');tit
le("Original and Moving Average Image , MSE=" +err1);

%Median Filter
image_filtered_median = medfilt2(image_with_noise,[5 5]);
err2=immse(image_filtered_median,image_original);
figure,imshowpair(image_original,image_filtered_median,'montage');title("Orig
inal and Median Image , MSE=" +err2);
```

Για το τρίτο ερώτημα :

```
%Read image
image_original = imread('clock.tiff');

%Estimate variance of input image
image_original = im2double(image_original);
var_image_original = std2(image_original)^2;

%Add white Gaussian noise , SNR=15db
SNRdB = 15;
sigma_noise = sqrt(var_image_original/10^(SNRdB/10));
noise = sigma_noise*randn(size(image_original));
image_with_noise = image_original+noise;

%Add Salt & Pepper Noise
image_with_noise = imnoise(image_with_noise,'salt & pepper',0.2);

err0=immse(image_original,image_with_noise);
figure,imshowpair(image_original,image_with_noise,'montage');title("Original
and Noisy Image , MSE=" +err0);

%Moving Average Filter
windowSize = 5;
kernel = ones(windowSize, windowSize) / windowSize ^ 2;
image_filtered_motion_average = imfilter(image_with_noise, kernel,
'symmetric');
err1=immse(image_filtered_motion_average,image_original);
figure,imshowpair(image_original,image_filtered_motion_average,'montage');tit
le("Original and Moving Average Image , MSE=" +err1);

%Median Filter
image_filtered_median = medfilt2(image_with_noise,[5 5]);
err2=immse(image_filtered_median,image_original);
figure,imshowpair(image_original,image_filtered_median,'montage');title("Orig
inal and Median Image , MSE=" +err2);

%First Median Second Moving Average
image_filtered_me_a= imfilter(image_filtered_median, kernel, 'symmetric');
err3=immse(image_filtered_me_a,image_original);
figure,imshowpair(image_original,image_filtered_me_a,'montage');title("Origin
al and Median-Average Image , MSE=" +err3);
```

4. Αποκατάσταση Εικόνας – Αποσυνέλιξη

Σύντομη περιγραφή τεχνικής – Μέρος Α

Το Wiener φίλτρο υλοποιήθηκε με την βοήθεια της συνάρτησης wiener2() της matlab η οποία υλοποιεί ένα **χωρικά προσαρμοστικό φίλτρο Wiener** , προσαρμόζεται δηλαδή , στις τοπικές στατιστικές ιδιότητες της εικόνας μέσω ενός παραθύρου 5x5.

Ο λόγος που δεν χρησιμοποιήθηκε κλασικό Wiener φίλτρο είναι ο εξής: Μια από τις προϋποθέσεις για την εφαρμογή φίλτρου Wiener με σκοπό την αποκατάσταση εικόνας από προσθετικό λευκό θόρυβο είναι ότι η εικόνα μας πρέπει να είναι ασθενώς στάσιμη και να έχει μέση τιμή μηδέν . Γίνεται κατανοητό ότι πρακτικά για να ισχύει το τελευταίο θα πρέπει όλα τα pixel της εικόνας να είναι μηδέν . Το πρόβλημα αυτό βέβαια λύνεται εάν αφαιρέσουμε την μέση τιμή της εικόνας από την εικόνα μας. Αν προχωρήσουμε σε αυτό το σημείο και εφαρμόσουμε το φίλτρο θα παρατηρήσουμε ότι θα εξακολουθούν να υπάρχουν εμφανείς παραμορφώσεις στην εικόνα , και αυτό διότι θεωρήσαμε ότι είναι ασθενώς στάσιμη ενώ στην πραγματικότητα αυτή είναι μια υπόθεση που σχεδόν ποτέ δεν ισχύει στις εικόνες. Για αυτό είναι προτιμότερο να εφαρμόσουμε ένα χωρικά προσαρμοστικό φίλτρο.

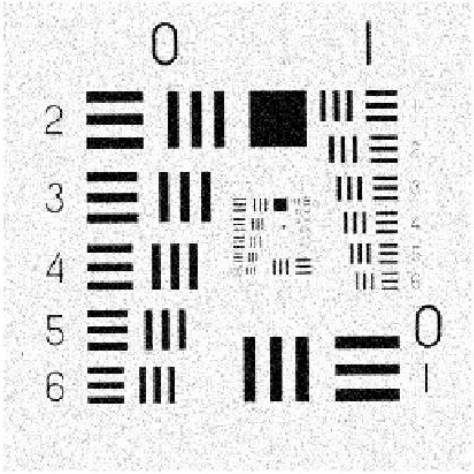
Αξιοποιώντας τη γνώση που έχουμε σχετικά με την ισχύ του θορύβου: Υπολογίζουμε το φάσμα ισχύος του λευκού θορύβου και το περνάμε ως όρισμα της συνάρτηση wiener2() , μαζί με το 5x5 παράθυρο και την εικόνα με τον θόρυβο.

Υποθέτοντας ότι δεν γνωρίζουμε την ισχύ του θορύβου: Αναφορικά και μόνο , στην περίπτωση που δεν χρησιμοποιούσαμε προσαρμοστικό φίλτρο αλλά κλασσικό Wiener για να βρούμε την ισχύ του λευκού θορύβου θα παίρναμε μετασχηματισμό Fourier της εικόνας με τον θόρυβο στις υψηλές συχνότητες . Στις υψηλές συχνότητες η πληροφορία είναι ελάχιστη (βρίσκεται κυρίως στο κέντρο) , ενώ ο θόρυβος έχει την ίδια παρουσία σε όλες τις συχνότητες . Και πάλι βέβαια η ισχύ που θα υπολογίζαμε θα ήταν κυρίως του θορύβου άλλα ένα μικρό ποσοστό θα ήταν της πληροφορίας της εικόνας.

Ο τρόπος που εν τέλη προσεγγίστηκε το φιλτράρισμα σε αυτή την περίπτωση είναι πάλι με χρήση της συνάρτησης wiener2() με μόνη διαφορά ότι δεν βάζουμε ως όρισμα την ισχύ του θορύβου . Η συνάρτηση στη θέση της ισχύς υπολογίζει τον μέσο όρο της τοπικής διασποράς .

Σχολιασμός αποτελεσμάτων – Μέρος Α

Image with Gaussian Noise , SNR=10db , MMSE=0.0088204



Original Image

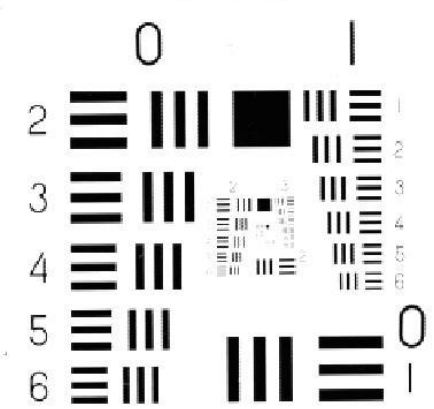
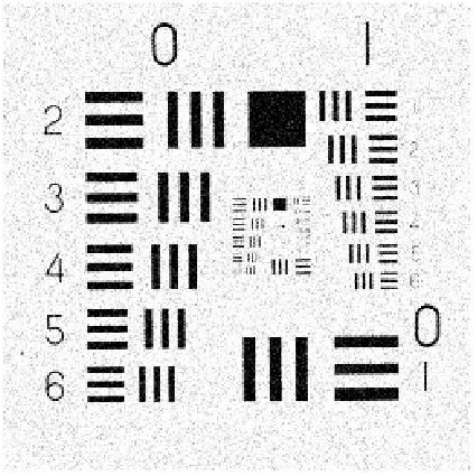


Image with Gaussian Noise , SNR=10db



Filterd with AdaptiveWiener (PSD of noise known),MMSE=0.0043422

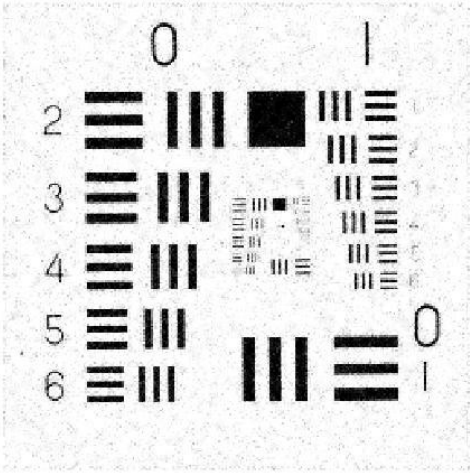
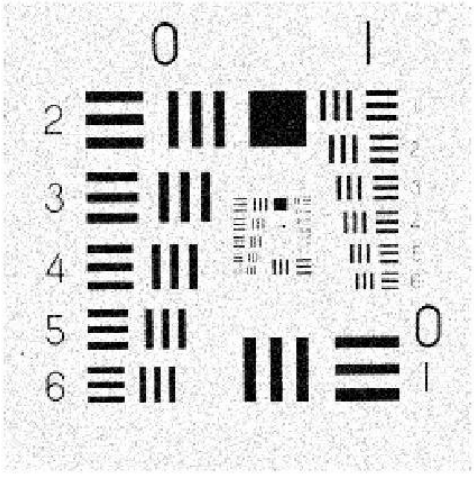
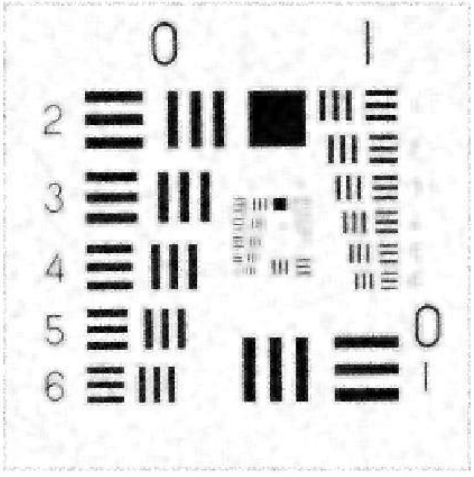


Image with Gaussian Noise , SNR=10db



Filterd with AdaptiveWiener (PSD of noise is unknown),MMSE=0.0041113



Βλέπουμε ότι η εικόνα με τον θόρυβο σε σχέση με την αρχική εικόνα (χωρίς θόρυβο) έχει MMSE = 0.0088.

Βλέπουμε ότι η εικόνα με τον θόρυβο σε σχέση με την φιλτραρισμένη γνωρίζοντας την ισχύ του θορύβου έχει MMSE = 0.0043.

Βλέπουμε ότι η εικόνα με τον θόρυβο σε σχέση με την φιλτραρισμένη μη γνωρίζοντας την ισχύ του θορύβου έχει MMSE = 0.0041.

Σαν γενικό συμπέρασμα το φίλτρο Wiener έχει εντυπωσιακά αποτελέσματα μειώνοντας το MSE περίπου στο μισό.

Ειδικότερα παρατηρούμε ότι στο φιλτράρισμα που δεν γνωρίζουμε την ισχύ του θορύβου και την προσεγγίζουμε μέσο του μέσο όρο της τοπικής διασποράς έχουμε λίγο καλύτερα αποτελέσματα από όταν γνωρίζαμε την ισχύ του. Αυτό οφείλεται , όπως εξηγήθηκε εκτενώς παραπάνω , στο ότι η εικόνα με τον θόρυβο (όπως και η εικόνα χωρίς τον θόρυβο) δεν θα είναι πότε WSS πρακτικά παρά μόνο θεωρητικά , έτσι ,προσεγγίζοντας την ισχύ του θορύβου με τοπικά στατιστικά χαρακτηριστικά μας δίνει καλύτερα αποτελέσματα.

Κώδικας – Μέρος Α

```
%Read image
image_original = imread('chart.tiff');

%Estimate variance of input image
image_original = im2double(image_original);
var_image_original = std2(image_original)^2;

%Add white Gaussian noise , SNR=10db
SNRdB = 10;
sigma_noise = sqrt(var_image_original/10^(SNRdB/10));
noise = sigma_noise*randn(size(image_original));
image_with_noise = image_original+noise;

%Check MSE between input image and noisy image
err0=immse(image_original,image_with_noise);

% Calculate power spectral density of noise
PSD_noise = abs(fft(noise)).^2/length(noise);

%Apply AdaptiveWiener filter with PSD of noise
filterd_image_1 = wiener2(image_with_noise,[5 5],PSD_noise);

%Check MSE between input image and filterd image
err1=immse(filterd_image_1,image_original);

%Apply AdaptiveWiener filter without PSD of noise
filterd_image_2 = wiener2(image_with_noise,[5 5]);

%Check MSE between input image and filterd image
err2=immse(filterd_image_2,image_original);
```

```
figure,subplot(121);imshow(image_with_noise);title("Image with Gaussian Noise ,
SNR=10db , MMSE=" +err0);
subplot(122);imshow(image_original);title('Original Image');


figure,subplot(121);imshow(image_with_noise);title('Image with Gaussian Noise ,
SNR=10db');
subplot(122);imshow(filterd_image_1);title("Filterd with AdaptiveWiener (PSD of
noise known),MMSE=" +err1 );

figure,subplot(121);imshow(image_with_noise);title('Image with Gaussian Noise ,
SNR=10db');
subplot(122);imshow(filterd_image_2);title("Filterd with AdaptiveWiener (PSD of
noise is unknown),MMSE=" +err2 );
```

Σύντομη περιγραφή τεχνικής – Μέρος Β

Για τον **υπολογισμό της κρουστικής απόκρισης** του άγνωστου συστήματος, έστω h , ακολουθήσαμε την εξής διαδικασία:

Εφαρμόσαμε στην αρχική εικόνα , έστω x , τον μετασχηματισμό που υλοποιείται στο αρχείο psf.p και έστω y η εικόνα εξόδου. Ισχύει ότι y = h **x (**=κυκλική συνέλιξη) στο χωρικό πεδίο και Y = H*X (γινόμενο) στο πεδίο των συχνοτήτων. Δηλαδή H = Y/X , τα x και y είναι γνωστά , εφαρμόζουμε 2D-FFT και βρίσκουμε τα Y και X , διαιρούμε το Y με το X και έχουμε το H , την απόκριση συχνότητας του άγνωστου συστήματος . Εφαρμόζουμε 2D-IFFT στο H και βρίσκουμε το h , την κρουστική απόκριση.

Για την **εφαρμογή του αντίστροφου φίλτρου** στο πεδίο της συχνότητας με χρήση κατωφλιού, ώστε να αντιμετωπίσουμε το θόλωμα που προκύπτει ακολουθήσαμε την εξής διαδικασία:

Ορίσαμε σαν κατώφλι (threshold) τιμές στο [400 , 450] , οι τιμές αυτές είναι ενδεικτικές για να απεικονίσουμε την μεταβολή του MSE μεταξύ της αρχικής εικόνας x και της προσέγγισης αυτής.

Ορίσαμε την απόκριση συχνότητας του αντίστροφου φίλτρου ,έστω invHs , ως εξής :

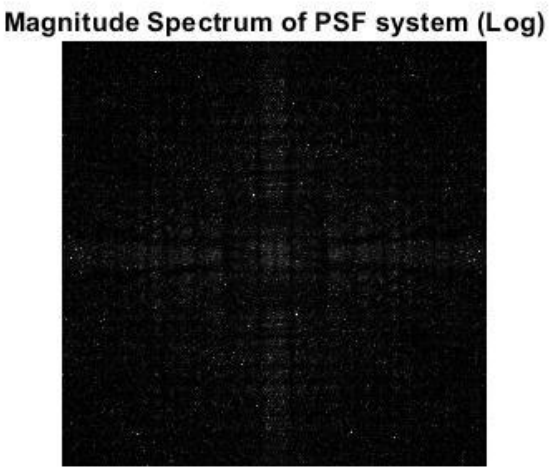
Αν : 1/|Hs| < κατώφλι , invHs = 1 / Hs .

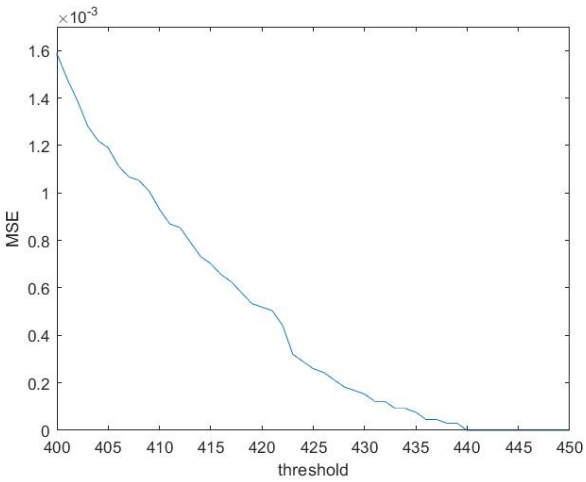
Αν : 1/|Hs| >= κατώφλι , invHs = (κατώφλι*|Hs|) / Hs .

Το γινόμενο της απόκρισης συχνότητας του αντίστροφου φίλτρου με τον μετασχηματισμός Fourier της y μας δίνει τον μετασχηματισμό Fourier της προσέγγιση της αρχικής εικόνας x . Εφαρμόζοντας αντίστροφο Fourier βρίσκουμε την προσέγγιση της αρχικής εικόνας x.

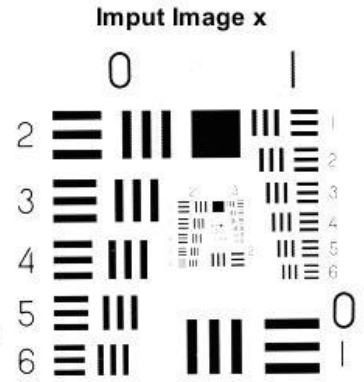
Σε περίπτωση που δεν γίνει χρήση κατωφλίου θα έχουμε : invHs = 1 / Hs . Αν η απόκριση συχνότητας του άγνωστου συστήματος , Hs , έχει τιμές μηδέν ή έστω και πολύ μικρές τιμές τότε το όρος 1 / Hs θα τείνει στο άπειρο και όταν πάρουμε αντίστροφο Fourier θα χαλάσει η εικόνα. Με την χρήση κατωφλίου ουσιαστικά ελέγχω κατά πόσο υπάρχουν τέτοια σημεία μηδενισμού ή πολύ μικρής τιμής στο Hs και φράσω κατά κάποιο τρόπο τις συνέπειες που αυτό θα προκαλέσει.

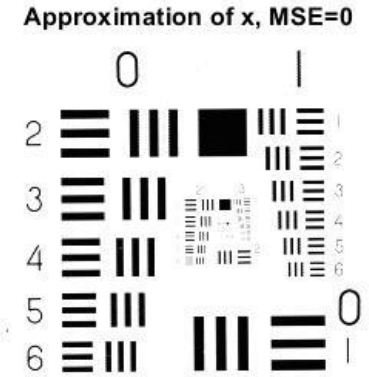
Σχολιασμός αποτελεσμάτων – Μέρος Β





Παρατηρούμε ότι όσο αυξάνεται το κατώφλι μειώνεται το μέσο τετραγωνικό σφάλμα μεταξύ της αρχικής εικόνας και της προσέγγισης της . Από την τιμή threshold = 440 και μετά το MSE γίνεται μηδενικό. Παρακάτω βλέπουμε την αρχική εικόνα και την προσέγγιση της με αντίστροφο φίλτρο με threshold = 450.





Κώδικας – Μέρος Β

```
%Read input image x
x = imread('chart.tiff');
%Output image y
y=psf(x);
y=cast(y,'uint8');
%Calculate X
X=fft2(double(x));
Xs=fftshift(X);
%Calculate Y
Y=fft2(double(y));
Ys=fftshift(Y);
%Calculate H
Hs = Ys./ Xs;
%Calculate h
h=ifft2(fftshift(Hs));
h=uint8(real(h));

%Inverse Filter with Threshold
for threshold = 400:450;
for i=1:256
    for j=1:256
        if (1/(abs(Hs(i,j))) < threshold)
             invHs(i,j) = 1/Hs(i,j) ;
        elseif (1/(abs(Hs(i,j))) >= threshold)
            invHs(i,j) = (threshold*abs(Hs(i,j)))./Hs(i,j);
        end
    end
end

approx_Xs=Ys.*invHs;

%Calculate approximation of x
approx_x=ifft2(fftshift(approx_Xs));
approx_x=uint8(real(approx_x));

%Calculate and Plot MMSE
err(threshold) = immse(approx_x,x);
plot(err)
xlabel('threshold')
ylabel('MSE')
axis([400 450 0 0.0017])
end

err1 = immse(approx_x,x);
figure,imshow(x);title('Imput Image x');
figure,imshow(approx_x);title("Approximation of x, MSE=" + err1);
figure,imshow(log(1+abs(Hs)),[]);title('Magnitude Spectrum of PSF system (Log)');
```

5. Βελτίωση εικόνας – Εξίσωση Ιστογράμματος

Σύντομη περιγραφή τεχνικής

Η **ολική εξίσωση ιστογράμματος** της εικόνας barbara.mat υλοποιήθηκε με την βοήθεια της συνάρτησης histeq() της matlab ενώ η **τοπική εξίσωση ιστογράμματος** υλοποιήθηκε με την βοήθεια της συνάρτησης adapthisteq() .

Ολική εξίσωση ιστογράμματος στο χρωματικό σύστημα RGB: Κάνουμε εξίσωση ιστογράμματος σε κάθε συνιστώσα (red,green,blue) ξεχωριστά και μετά συνθέτουμε τις συνιστώσες για να πάρουμε την εικόνα της οποίας το ιστόγραμμα έχει εξισωθεί.

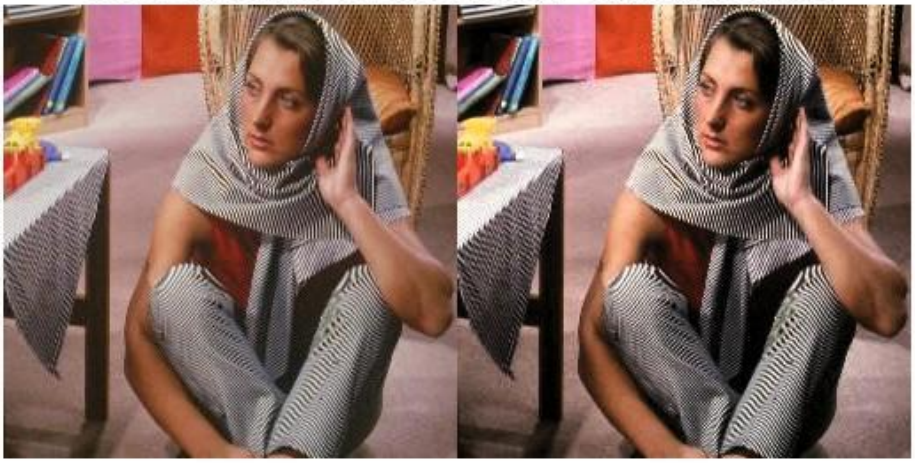
Ολική εξίσωση ιστογράμματος στο χρωματικό σύστημα HSV: Μετατρέπουμε την εικόνα από RGB σε HSV , εφαρμόζουμε ολική εξίσωση ιστογράμματος στην συνιστώσα Intesity , επανασυνθέτουμε την HSV και την μετατρέπουμε σε RGB.

Για την αποκάλυψη περιεχομένου που δεν φαίνεται , εφαρμόσαμε τοπική εξίσωση ιστογράμματος με παράθυρο 11x11 . Έπειτα για να εξισώσουμε σε ικανοποιητικό βαθμό το ιστόγραμμα χρησιμοποιήσαμε τις συναρτήσεις imadjust() και imsharpen() της matlab.

Σχολιασμός αποτελεσμάτων

Ολική εξίσωση ιστογράμματος στο χρωματικό σύστημα RGB και στο HSV:

Before and After Total Histogram Equalization HSV



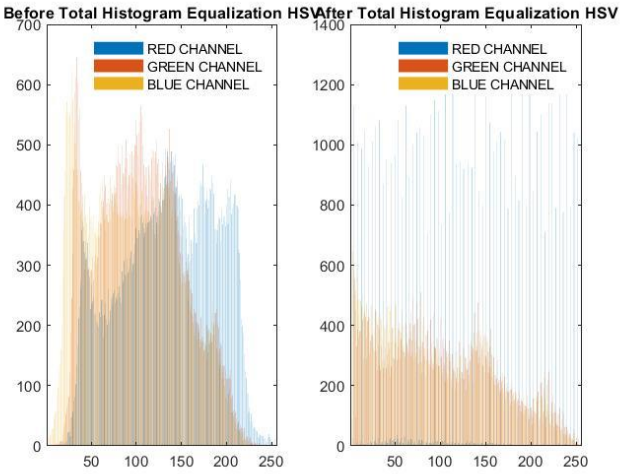
Το χρωματικό σύστημα HSV έχει μια βασική διαφορά από το RGB , απομονώνει την συνιστώσα Intesity οπότε η πληροφορία του χρώματος βρίσκεται στις δύο συνιστώσες Hue και Saturation.

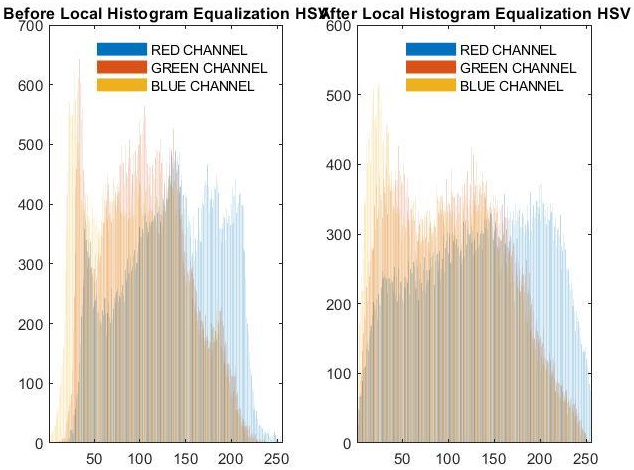
Η εξίσωση ιστογράμματος συνεπάγει την βελτίωση της αντίθεσης , κάτι που έχει να κάνει με την ένταση της φωτεινότητας (Intensity) και όχι με τις χρωματικές συνιστώσες. Συνεπώς είναι προτιμότερη η χρήση χρωματικού συστήματος HSV έναντι του RGB γιατί , όπως φαίνεται στις εικόνες παραπάνω στην περίπτωση του RGB , εκτός από την φωτεινότητα επηρεάζονται (αλλοιώνονται) και τα χρώματα.

Το ίδιο παρατηρούμε ότι ισχύει και για την τοπική εξίσωση ιστογράμματος.

Ολική και τοπική εξίσωση ιστογράμματος στο χρωματικό σύστημα HSV:

Total and Local HSV

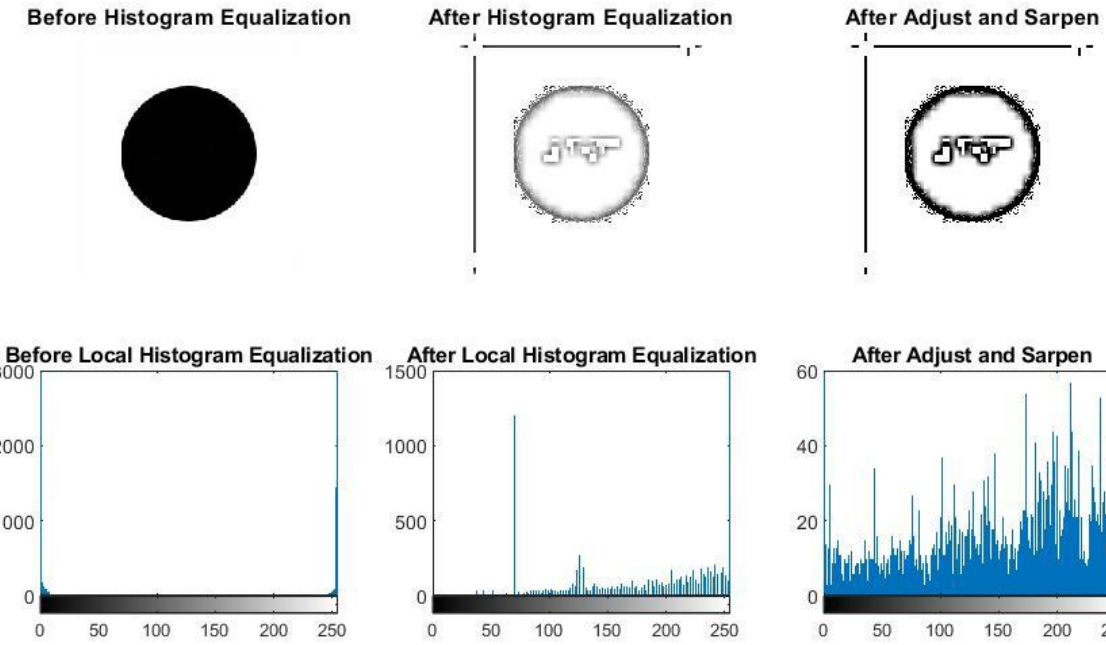




Παρατηρούμε ότι δεν μπορούμε να πετύχουμε αυτά που μας υπόσχεται η θεωρία δηλαδή εξισωμένο ιστόγραμμα. Αυτό δικαιολογείται καθώς έχουμε μικρό αριθμό από pixel (256x256) και τα επίπεδα χρώματος (256) είναι πολύ λίγα (8bits/pixel).

Παρατηρούμε επίσης πως σε μια εικόνα με πολλές ανομοιογενείς περιοχές , όπως η δική μας , ο τοπικός μετασχηματισμός είναι προτιμότερος μιας και υπάρχει μεγαλύτερη βελτίωση της αντίθεσης και αποκάλυψη περισσότερων περιοχών σε σχέση με την ολική.

Για την εικόνα της οποίας το περιεχόμενο θέλουμε να αποκαλύψουμε :



Στην πρώτη στήλη βλέπουμε την εικόνα μας και το ιστόγραμμα της . Παρατηρούμε ότι η πλειοψηφία των pixel έχει τιμή 250 και ένα μικρότερο ποσοστό τιμή 0 , οι ενδιάμεσες τιμές είναι σχεδόν ανύπαρκτες. Στην πραγματικότητα ο κύκλος δεν είναι μαύρος , μας φαίνεται έτσι , γιατί τα grayscales είναι πολύ κοντά στο 0 και δεν μπορούμε να τα διακρίνουμε . Αν εφαρμόζαμε ολική εξίσωση ιστογράμματος δεν θα έκανε τίποτα γιατί αυτή η εικόνα κυριαρχείται από τα άσπρα , αν όμως εφαρμόσω μια τοπική εξίσωση μπορώ να διακρίνω το περιεχόμενο του κύκλου.

Στην δεύτερη στήλη βλέπουμε την εικόνα μετά από τοπική εξίσωση ιστογράμματος (χωρίς την χρήση έτοιμων συναρτήσεων) και παρατηρούμε ότι το ιστόγραμμα έχει βελτιωθεί σε ότι αφορά την συγκέντρωση των τιμών των pixel σε ευρύτερο φάρμα τιμών . Διαφαίνεται το περιεχόμενο του κύκλου ωστόσο το ιστόγραμμα έχει περιθώρια βελτίωσης.

Στην τρίτη στήλη βλέπουμε την εικόνα που έχει υποστεί τοπική εξίσωση ιστογράμματος αφότου της έχουμε εφαρμόσει τεχνικές για βελτίωση της αντίθεσης και τα αποτελέσματα που βλέπουμε στο ιστόγραμμα είναι εντυπωσιακά με τις τιμές των pixel καλύπτουν ομοιόμορφα όλη την δυναμική περιοχή.

Κώδικας

Ο ακόλουθος κώδικα υλοποιεί ολική και τοπική εξίσωση ιστογράμματος στο χρωματικό σύστημα RGB.

```
load('barbara.mat')
img = imshow(barbara);
imsave(img);

%Read the input image
I=imread('barbara.jpg');
R = I(:,:,1);
G = I(:,:,2);
B = I(:,:,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Perform Total Histogram Equalization each component (RGB)
Req = histeq(R);
Geq = histeq(G);
Beq = histeq(B);
RGB_total = I;
RGB_total(:,:,1) = Req;
RGB_total(:,:,2) = Geq;
RGB_total(:,:,3) = Beq;

%Display the histogram of the original and the equalized Image
HIST_IN_TOTAL = zeros([256 3]);
HIST_OUT_TOTAL = zeros([256 3]);

%Histogram of the RED,GREEN and BLUE Components
HIST_IN_TOTAL(:,1) = imhist(I(:,:,1),256); %RED
HIST_IN_TOTAL(:,2) = imhist(I(:,:,2),256); %GREEN
HIST_IN_TOTAL(:,3) = imhist(I(:,:,3),256); %BLUE
HIST_OUT_TOTAL(:,1) = imhist(RGB_total(:,:,1),256); %RED
HIST_OUT_TOTAL(:,2) = imhist(RGB_total(:,:,2),256); %GREEN
HIST_OUT_TOTAL(:,3) = imhist(RGB_total(:,:,3),256); %BLUE
```

```
mymap=[1 0 0; 0.2 1 0; 0 0.2 1];

%Display images before and after histogram equalization
figure,imshowpair(I,RGB_total,'montage');title('Before and After Total Histogram
Equalization RGB');
figure,subplot(121),bar(HIST_IN_TOTAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('Before Total Histogram Equalization RGB');
subplot(122),bar(HIST_OUT_TOTAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('After Total Histogram Equalization RGB');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Perform Local Histogram Equalization on each component (RGB)

Req = adapthisteq(R);
Geq = adapthisteq(G);
Beq = adapthisteq(B);
RGB_local = I;
RGB_local(:,:,1) = Req;
RGB_local(:,:,2) = Geq;
RGB_local(:,:,3) = Beq;

%Display the histogram of the original and the equalized Image
HIST_IN_LOCAL = zeros([256 3]);
HIST_OUT_LOCAL = zeros([256 3]);

%Histogram of the RED,GREEN and BLUE Components
HIST_IN_LOCAL(:,1) = imhist(I(:,:,1),256); %RED
HIST_IN_LOCAL(:,2) = imhist(I(:,:,2),256); %GREEN
HIST_IN_LOCAL(:,3) = imhist(I(:,:,3),256); %BLUE
HIST_OUT_LOCAL(:,1) = imhist(RGB_local(:,:,1),256); %RED
HIST_OUT_LOCAL(:,2) = imhist(RGB_local(:,:,2),256); %GREEN
HIST_OUT_LOCAL(:,3) = imhist(RGB_local(:,:,3),256); %BLUE
mymap=[1 0 0; 0.2 1 0; 0 0.2 1];

%Display images before and after histogram equalization
figure,imshowpair(I,RGB_local,'montage');title('Before and After Local Histogram
Equalization RGB');
figure,subplot(121),bar(HIST_IN_LOCAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('Before Local Histogram Equalization RGB');
subplot(122),bar(HIST_OUT_LOCAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('After Local Histogram Equalization RGB');

figure,imshowpair(RGB_total,RGB_local,'montage');title('Total and Local RGB');
```

Ο ακόλουθος κώδικα υλοποιεί ολική και τοπική εξίσωση ιστογράμματος στο χρωματικό σύστημα HSV.

```
load('barbara.mat')
img = imshow(barbara);
imsave(img);

%Read the input image
I=imread('barbara.jpg');

%Convert the rgb image into hsv image format
HSV = rgb2hsv(I);

%Intensity Componet
v = HSV(:,:,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Perform Total Histogram Equalization on Intensity Component (HSV)

Heq_total = histeq(v);
HSV_mod_total = HSV;
HSV_mod_total(:,:,3)= Heq_total;
RGB_total = hsv2rgb(HSV_mod_total);

%Display the histogram of the original and the equalized Image
HIST_IN_TOTAL = zeros([256 3]);
HIST_OUT_TOTAL = zeros([256 3]);

%Histogram of the RED,GREEN and BLUE Components
HIST_IN_TOTAL(:,1) = imhist(I(:,:,1),256); %RED
HIST_IN_TOTAL(:,2) = imhist(I(:,:,2),256); %GREEN
HIST_IN_TOTAL(:,3) = imhist(I(:,:,3),256); %BLUE
HIST_OUT_TOTAL(:,1) = imhist(RGB_total(:,:,1),256); %RED
HIST_OUT_TOTAL(:,2) = imhist(RGB_total(:,:,2),256); %GREEN
HIST_OUT_TOTAL(:,3) = imhist(RGB_total(:,:,3),256); %BLUE
mymap=[1 0 0; 0.2 1 0; 0 0.2 1];

%Display images before and after histogram equalization
figure,imshowpair(I,RGB_total,'montage');title('Before and After Total Histogram
Equalization HSV');
figure,subplot(121),bar(HIST_IN_TOTAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('Before Total Histogram Equalization HSV');
subplot(122),bar(HIST_OUT_TOTAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('After Total Histogram Equalization HSV');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Perform Local Histogram Equalization on Intensity Component (HSV)
Heq_local = adapthisteq(v);
HSV_mod_local = HSV;
HSV_mod_local(:,:,3)= Heq_local;
RGB_local = hsv2rgb(HSV_mod_local);

%Display the histogram of the original and the equalized Image
HIST_IN_LOCAL = zeros([256 3]);
HIST_OUT_LOCAL = zeros([256 3]);

%Histogram of the RED,GREEN and BLUE Components
HIST_IN_LOCAL(:,1) = imhist(I(:,:,1),256); %RED
HIST_IN_LOCAL(:,2) = imhist(I(:,:,2),256); %GREEN
HIST_IN_LOCAL(:,3) = imhist(I(:,:,3),256); %BLUE
HIST_OUT_LOCAL(:,1) = imhist(RGB_local(:,:,1),256); %RED
```

```
HIST_OUT_LOCAL(:,2) = imhist(RGB_local(:,:,2),256); %GREEN
HIST_OUT_LOCAL(:,3) = imhist(RGB_local(:,:,3),256); %BLUE
mymap=[1 0 0; 0.2 1 0; 0 0.2 1];

%Display images before and after histogram equalization
figure,imshowpair(I,RGB_local,'montage');title('Before and After Local Histogram
Equalization HSV');
figure,subplot(121),bar(HIST_IN_LOCAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('Before Local Histogram Equalization HSV');
subplot(122),bar(HIST_OUT_LOCAL);colormap(mymap);legend('RED CHANNEL','GREEN
CHANNEL','BLUE CHANNEL');title('After Local Histogram Equalization HSV');

figure,imshowpair(RGB_total,RGB_local,'montage');title('Total and Local HSV');
```

Για την εικόνα με τον κύκλο :

```
load('circle.mat')
img = imshow(circle);
imsave(img);

%Read the input image
I=imread('circle.jpg');
Img = I;

%Window size
M=11;
N=11;
mid_val=round((M*N)/2);

%Find the number of rows and columns to be padded with zero
in=0;
for i=1:M
    for j=1:N
        in=in+1;
        if(in==mid_val)
            PadM=i-1;
            PadN=j-1;
            break;
        end
    end
end

%Padding the intensity componet with zero on all sides
B=padarray(I,[PadM,PadN]);

for i= 1:size(B,1)-((PadM*2)+1)

    for j=1:size(B,2)-((PadN*2)+1)
        cdf=zeros(256,1);
        inc=1;
        for x=1:M
            for y=1:N

  %Fing the middle element in the window
```

```
                if(inc==mid_val)
                    ele=B(i+x-1,j+y-1)+1;
                end
                    pos=B(i+x-1,j+y-1)+1;
                    cdf(pos)=cdf(pos)+1;
                   inc=inc+1;
            end
        end

        %Compute the CDF for the values in the window
        for l=2:256
            cdf(l)=cdf(l)+cdf(l-1);
        end
            Img(i,j)=round(cdf(ele)/(M*N)*255);
    end
end

J = imadjust(Img,[0.7 1],[]);
J = imsharpen(J);

%Display images before and after histogram equalization
figure,subplot(231),imshow(I);title('Before Histogram Equalization');
subplot(232),imshow(Img);title('After Histogram Equalization');
subplot(233),imshow(J);title('After Adjust and Sarpen');
subplot(234);imhist(I);ylim([0 3000]);title('Before Local Histogram Equalization');
subplot(235); imhist(Img);ylim([0 1500]);title('After Local Histogram
Equalization');
subplot(236); imhist(J);ylim([0 60]);title('After Adjust and Sarpen');
```

6. Ανίχνευση Ακμών – Κατάτμηση

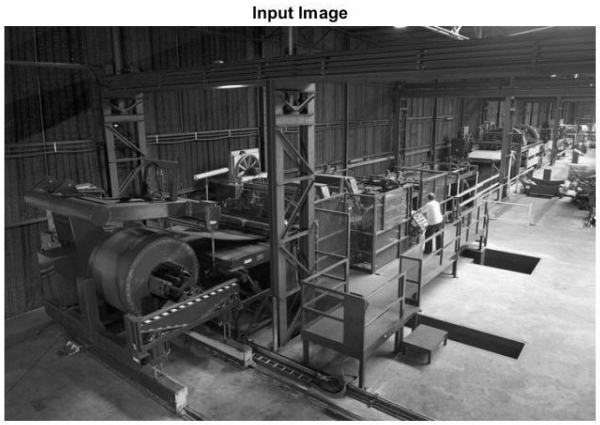
Σύντομη περιγραφή τεχνικής

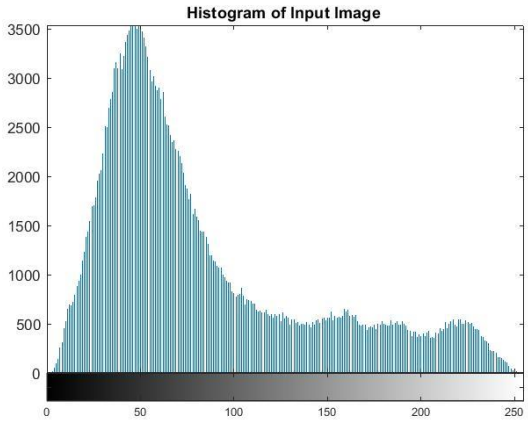
Για την **ανίχνευση ακμών** χρησιμοποιήθηκαν οι μάσκες Roberts και Sobel και υλοποιήθηκαν με την βοήθεια της συνάρτησης edge() της matlab.

Για την **ολική κατωφλίωση** το κατώφλι εκτιμήθηκε μέσω του ιστογράμματος.

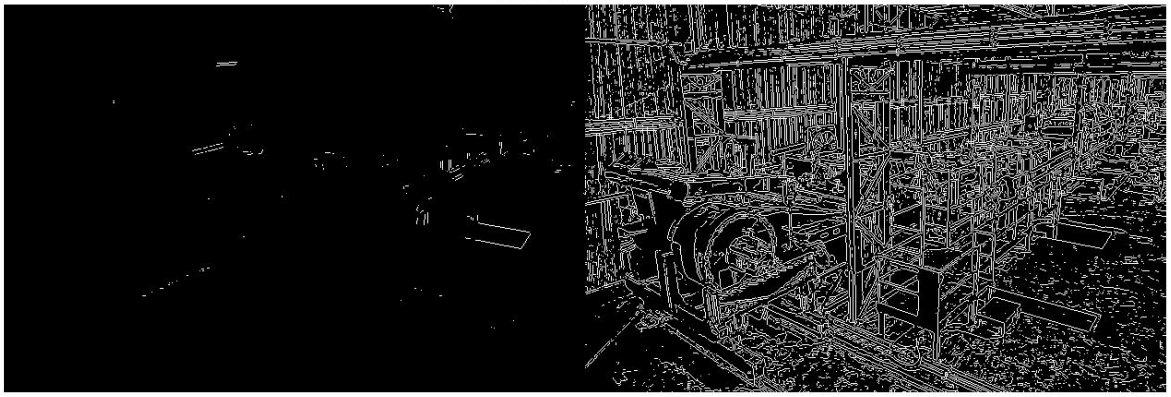
Για την **ανίχνευση και διασύνδεση ακμών** υλοποιήθηκε ο μετασχηματισμός Hough με την βοήθεια των συναρτήσεων hough() και houghpeaks() της matlab.

Σχολιασμός αποτελεσμάτων

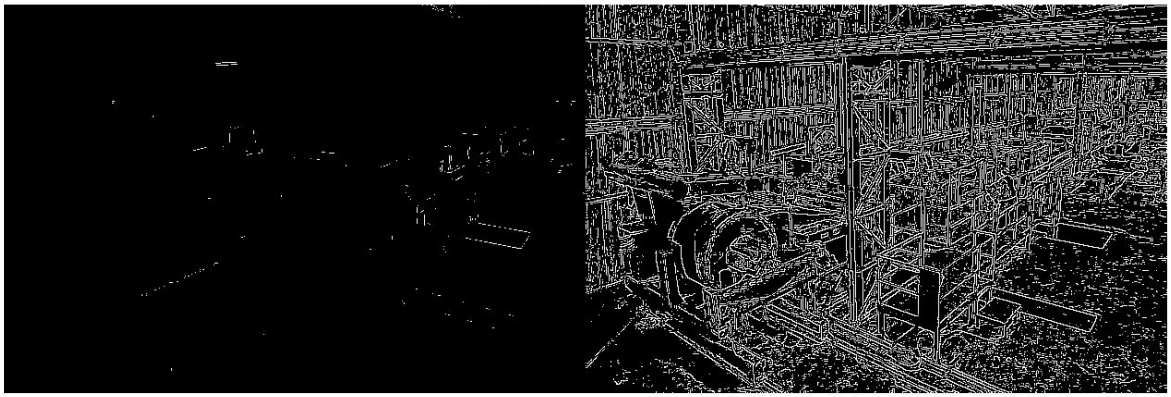




Sobel 30% Sobel 2%



Roberts 30% Roberts 2%



Σύγκριση Sober – Roberts:

Η μάσκα Roberts παρέχει περισσότερη πληροφορία ακμών για κάθε μια από τις δύο περιπτώσεις διότι αποτελεί μάσκα 3x3 ενώ η μάσκα Sobel είναι 2x2.

Σύγκριση Sober /Roberts για διαφορετικά ποσοστά:

Τα ποσοστά αντιστοιχούν στο εμβαδόν του ιστογράμματος από το σημείο που βάζουμε το κατώφλι (άξονας x-pixel) μέχρι το τέλος του ιστογράμματος.

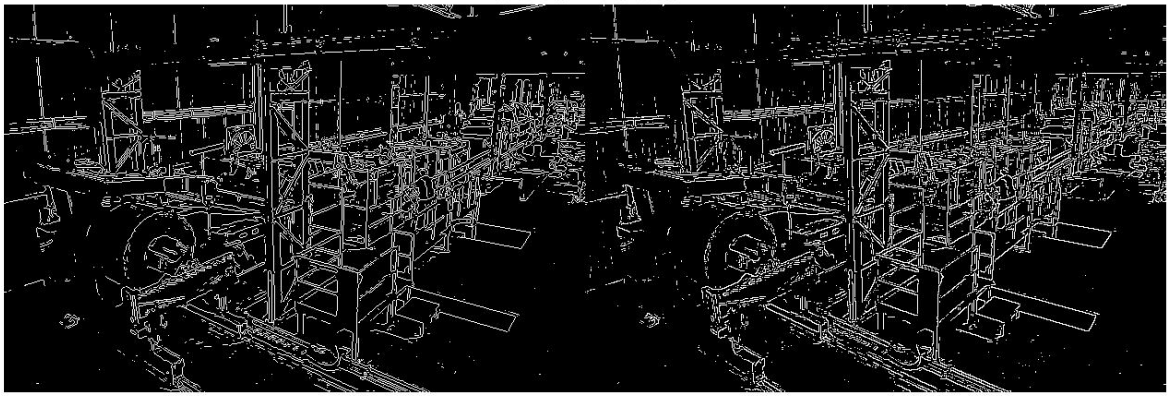
Αν το κατώφλι είναι μεγάλο (βλέπε 30%) κρατάμε λίγες ακμές και έχουμε πολλές σπασμένες περιοχές. Αν το κατώφλι είναι μικρό (βλέπε 2%) έχουμε πολύ λιγότερες σπασμένες πραγματικές ακμές αλλά έχουμε και αρκετές ψευδοακμές.

Σημείωση: Από το κατώφλι και μετά όλα τα pixel παίρνουν την τιμή 1.

Οι ακμές σε μια εικόνα είναι ένα μικρό ποσοστό της επιφάνειας της , περίπου το 5%. Οπότε βλέποντας το ιστόγραμμα εντοπίζω ποια περιοχή έχει εμβαδόν ίσο με 5% του συνολικού εμβαδού , εκεί βάζω το κατώφλι.

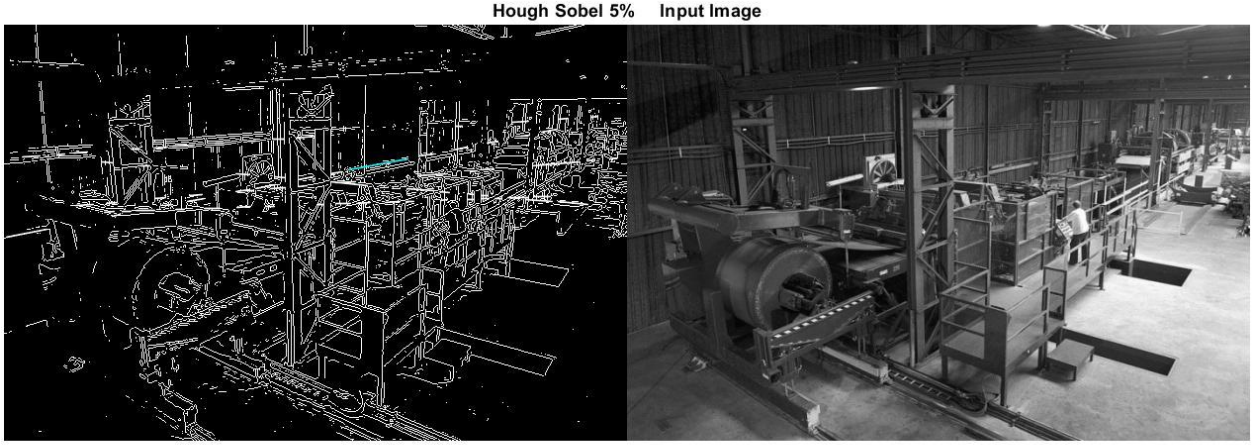
Από τα ποσοστά που είδαμε παραπάνω, 30% και 2%, καταλαβαίνουμε ότι μας ενδιαφέρει μια ενδιάμεση λύση ώστε να απαλλαγούμε από ψευδοακμές αλλά να μην σπάσουμε πολύ τις πραγματικές ακμές.

Sobel 5% Roberts 5%

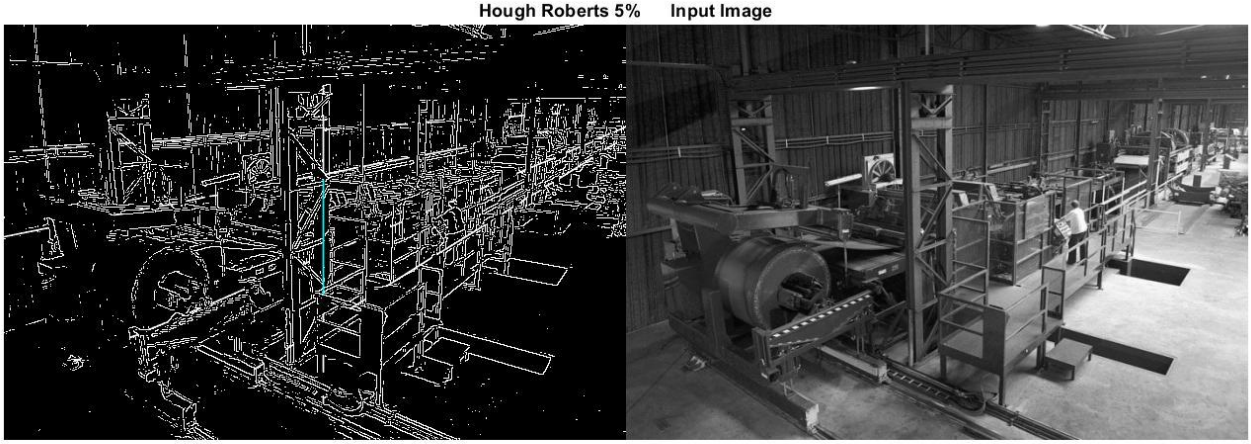


Αυτή η ενδιάμεση λύση δεν μπορεί να είναι το τελικό μου αποτέλεσμα γιατί και πάλι έχω ασυνέχειες στις ακμές . Χρειάζεται μια τεχνική ένωσης ακμών , στην συγκεκριμένη περίπτωση τον μετασχηματισμό Hough , ο οποίος εν ολίγης ενώνει ακμές εάν έχουν παρόμοια τιμή (μέτρο) και παρόμοια διεύθυνση.

Hough Sobel 5% Input Image



Hough Roberts 5% Input Image



Κώδικας

```
%1
%Read the input image
image=imread('factory.jpg');
img=rgb2gray(image);
%Roberts mask for 30% and 2%
edges_roberts_1 = edge(img,'Roberts',0.30);
edges_roberts_2 = edge(img,'Roberts',0.02);
%Sobel mask for 30% and 2%
edges_sobel_1 = edge(img,'Sobel',0.30);
edges_sobel_2 = edge(img,'Sobel',0.02);
%Display
figure,imshow(img); title('Input Image')
figure,imhist(img); title('Histogram of Input Image ')
figure,imshowpair(edges_sobel_1,edges_sobel_2,'montage');title('Sobel 30% Sobel 2%')
figure,imshowpair(edges_roberts_1,edges_roberts_2,'montage');title('Roberts 30%
Roberts 2%')
```

```
%2
%Roberts and Sobel for 5%
edges_sobel = edge(img,'Sobel',0.05);
edges_roberts = edge(img,'Roberts',0.05);
%Display
figure,imshowpair(edges_sobel,edges_roberts,'montage');title('Sobel 5%  Roberts 5%')
```

```
%3
%Hough for Sobel 5%

%Compute the Hough transform of the binary image returned by edge.
[H,theta,rho] = hough(edges_sobel);

%Find the peaks in the Hough transform matrix, H, using the houghpeaks function
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));

%Find lines in the image using the houghlines function.
lines = houghlines(edges_sobel,theta,rho,P,'FillGap',5,'MinLength',7);

%Create a plot that displays the binary image with the lines superimposed on it
figure,imshowpair(edges_sobel,img,'montage');title('Hough Sobel 5% Input
Image'),hold on
max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','white');
  % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;
   end
end

% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',1,'Color','cyan');


%Hough for Roberts 5%
%Compute the Hough transform of the binary image returned by edge.
[H,theta,rho] = hough(edges_roberts);

%Find the peaks in the Hough transform matrix, H, using the houghpeaks function
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));

%Find lines in the image using the houghlines function.
lines = houghlines(edges_roberts,theta,rho,P,'FillGap',5,'MinLength',7);

%Create a plot that displays the binary image with the lines superimposed on it
figure,imshowpair(edges_roberts,img,'montage');title('Hough Roberts 5%      Input
Image'),hold on
max_len = 0;
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',1,'Color','white');
   % Determine the endpoints of the longest line segment
   len = norm(lines(k).point1 - lines(k).point2);
   if ( len > max_len)
      max_len = len;
      xy_long = xy;
   end
end
% highlight the longest line segment
plot(xy_long(:,1),xy_long(:,2),'LineWidth',1,'Color','cyan');
```