Image Processing

MATLAB

LAB Assignment

Assignment 5

Submitted by:

Akanksha Bansal

071213

**Noises generation function**

function [ R ] = imnoise2( type, M,N,a,b )

if nargin ==1

a=0; b=1;

M=1;N=1;

elseif nargin==3

a=0;b=1;

end

switch lower(type)

case 'uniform'

R=a+(b-a)\*randn(M,N);

case 'gaussian'

R=a+b\*randn(M,N);

case 'rayleigh'

R=a+(-b\*log(1-rand(M,N))).^0.5;

case 'exponential'

if nargin<=3

a=1;

end

if a<=0

error('Parameter a must be positive dor exponential type.');

end

k=-1/a;

R=k\*log(1-rand(M,N));

case 'erlang'

if nargin<=3

a=2;b=5;

end

k=-1/a;

R=zeros(M,N);

for j=1:b

R=R+k\*log(a-rand(M,N));

end

otherwise

error('Unkown distribution type');

end

**Noise filters**

s=imread('2.jpg');

**%Gaussian Noise**

g = imnoise(s, 'gaussian'); %could have used imnoise2 function too

arith\_mean = fspecial('average', [5 5]);

arith\_filtered = imfilter(g, arith\_mean);

med\_filt = medfilt2(g, [5 5]);

max = ordfilt2(g, 3\*3, ones(3, 3));

min = ordfilt2(g, 1, ones(3, 3));

midpt\_filt = (max + min) \* 0.5;

figure(1);

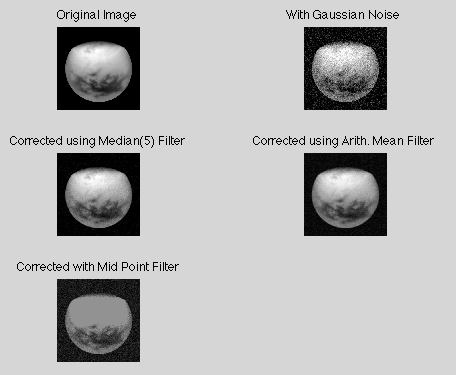
subplot(3, 2, 1), imshow(s);title('Original Image');

subplot(3, 2, 2), imshow(g);title('With Gaussian Noise');

subplot(3, 2, 3), imshow(med\_filt);title('Corrected using Median(5) Filter');

subplot(3, 2, 4), imshow(arith\_filtered);title('Corrected using Arith. Mean Filter');

subplot(3, 2, 5), imshow(midpt\_filt);title('Corrected with Mid Point Filter');



Arithmetic filter produces best results.

**%Salt and Pepper (Impulse) Noise**

**g**1 = imnoise(s, 'salt & pepper');

max\_filter = ordfilt2(g1, 3\*3, ones(3, 3));

min\_filter = ordfilt2(g1, 1, ones(3, 3));

med\_filt1 = medfilt2(g1, [5 5]);

figure(2);

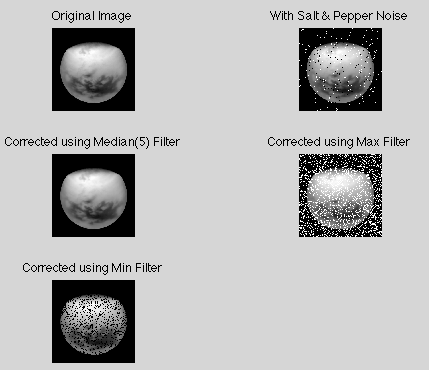
subplot(3, 2, 1), imshow(s);title('Original Image');

subplot(3, 2, 2), imshow(g1);title('With Salt & Pepper Noise');

subplot(3, 2, 3), imshow(med\_filt1);title('Corrected using Median(5) Filter');

subplot(3, 2, 4), imshow(max\_filter);title('Corrected using Max Filter');

subplot(3, 2, 5), imshow(min\_filter);title('Corrected using Min Filter');



Median filter produces best results.

**%Rayleigh Noise**

[M N]=size(s);

R=imnoise2('rayleigh',M,N,0.05,0.05);

B=im2double(s);

ray\_n=B+R;

arith\_mean\_r = fspecial('average', [5 5]);

arith\_filtered\_r = imfilter(ray\_n, arith\_mean\_r);

ray\_d = im2double(ray\_n);

geo\_filtered\_r = exp(imfilter(log(ray\_d), ones(3, 3), 'replicate')).^(1/3/3);

med\_filt\_r = medfilt2(ray\_n, [5 5]);

max\_r = ordfilt2(ray\_n, 3\*3, ones(3, 3));

min\_r = ordfilt2(ray\_n, 1, ones(3, 3));

midpt\_filt\_r = (max\_r + min\_r) \* 0.5;

figure(3);

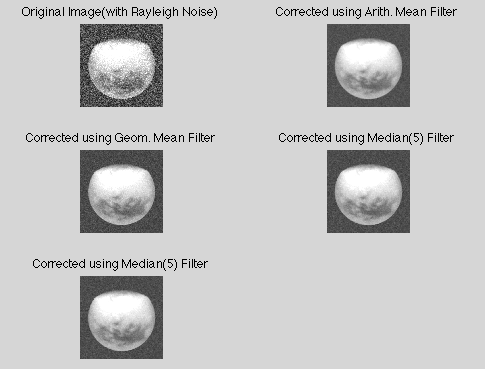
subplot(3, 2, 1), imshow(ray\_n);title('Original Image(with Rayleigh Noise)');

subplot(3, 2, 2), imshow(arith\_filtered\_r);title('Corrected using Arith. Mean Filter');

subplot(3, 2, 3), imshow(geo\_filtered\_r);title('Corrected using Geom. Mean Filter');

subplot(3, 2, 4), imshow(med\_filt\_r);title('Corrected using Median(5) Filter');

subplot(3, 2, 5), imshow(med\_filt\_r);title('Corrected using Median(5) Filter');



Geometric filter produces best results.

**%Uniform Noise**

[M N]=size(s);

R=imnoise2('uniform',M,N,0.1,0.05);

B=im2double(s);

uni=B+R;

arith\_mean\_u = fspecial('average', [5 5]);

arith\_filtered\_u = imfilter(uni, arith\_mean\_u);

uni\_d = im2double(uni);

geo\_filtered\_u = exp(imfilter(log(uni\_d), ones(3, 3), 'replicate')).^(1/3/3);

med\_filt\_u = medfilt2(uni, [5 5]);

max\_u = ordfilt2(uni, 3\*3, ones(3, 3));

min\_u = ordfilt2(uni, 1, ones(3, 3));

midpt\_filt\_u = (max\_u + min\_u) \* 0.5;

figure(5);

subplot(3, 2, 1), imshow(s);title('Original Image');

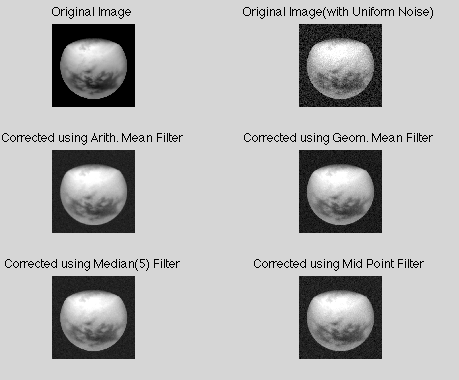
subplot(3, 2, 2), imshow(uni);title('Original Image(with Uniform Noise)');

subplot(3, 2, 3), imshow(arith\_filtered\_u);title('Corrected using Arith. Mean Filter');

subplot(3, 2, 4), imshow(geo\_filtered\_u);title('Corrected using Geom. Mean Filter');

subplot(3, 2, 5), imshow(med\_filt\_u);title('Corrected using Median(5) Filter');

subplot(3, 2, 6), imshow(midpt\_filt\_u);title('Corrected using Mid Point Filter');



Arithmetic filter and median filter produces best results.

**%Exponential Noise**

**[**M N]=size(s);

R=imnoise2('exponential',M,N,2,0.05);

B=im2double(s);

nexp=B+R;

arith\_mean\_e = fspecial('average', [5 5]);

arith\_filtered\_e = imfilter(nexp, arith\_mean\_e);

nexp\_e = im2double(nexp);

geo\_filtered\_e = exp(imfilter(log(nexp\_e), ones(3, 3), 'replicate')).^(1/3/3);

med\_filt\_e = medfilt2(nexp, [5 5]);

max\_e = ordfilt2(nexp, 3\*3, ones(3, 3));

min\_e = ordfilt2(nexp, 1, ones(3, 3));

midpt\_filt\_e = (max\_e + min\_e) \* 0.5;

figure(6);

subplot(3, 2, 1), imshow(s);title('Original Image');

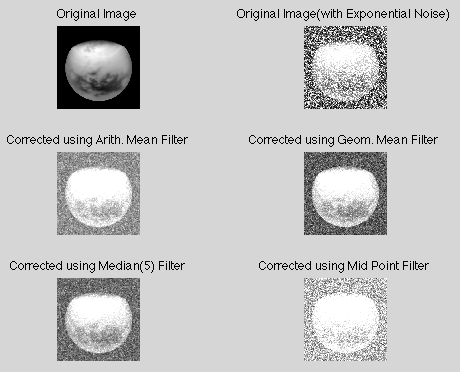
subplot(3, 2, 2), imshow(nexp);title('Original Image(with Exponential Noise)');

subplot(3, 2, 3), imshow(arith\_filtered\_e);title('Corrected using Arith. Mean Filter');

subplot(3, 2, 4), imshow(geo\_filtered\_e);title('Corrected using Geom. Mean Filter');

subplot(3, 2, 5), imshow(med\_filt\_e);title('Corrected using Median(5) Filter');

subplot(3, 2, 6), imshow(midpt\_filt\_e);title('Corrected using Mid Point Filter');



Geometric filter and median filter produces best results.

**Special filters**

function f=spfilt(g,type,m,n,parameter)

if nargin==2

m=3;n=3;Q=1.5;d=2;

elseif nargin==5

Q=parameter;d=parameter;

elseif nargin==4

Q=1.5;d=2;

else

error('Wrong number of inputs');

end

switch type

case 'amean'

w=fspecial('average',[m n]);

f=imfilter(g,w,'replicate');

case 'gmean'

f=gmean1(g,m,n);

case 'hmean'

f=harmean(g,m,n);

case 'chmean'

f=charmean(g,m,n,Q);

case 'median'

f=medfilt2(g,[m n],'symmetric');

case 'max'

f=ordfilt2(g,m\*n,ones(m,n),'symmetric');

case 'min'

f=ordfilt2(g,1,ones(m,n),'symmetric');

case 'midpoint'

f1=ordfilt2(g,1,ones(m,n),'symmetric');

f2=ordfilt2(g,m\*n,ones(m,n),'symmetric');

f=imlincomb(0.5,f1,0.5,f2);

case 'atrimmed'

if(d<0)|(d/2~=round(d/2))

error('d must be a non negative,even integer');

end

f=alphatrim(g,m,n,d);

otherwise

error('Unknown filter type');

end

%gmean

function f=gmean1(g,m,n)

g=im2double(g);

warning off;

f=exp(imfilter(log(g),ones(m,n),'replicate')).^(1/m/n);

warning on;

%harmean

function f=harmean(g,m,n)

g=im2double(g);

f=m\*n./imfilter(1./(g+eps),ones(m,n),'replicate');

%charmean

function f=charmean(g,m,n,q)

g=im2double(g);

f=imfilter(g.^(q+1),ones(m,n),'replicate');

f=f./(imfilter(g.^q,ones(m,n),'replicate')+eps);

%alphatrim

function f=alphatrim(g,m,n,d)

g=im2double(g);

f=imfilter(g,ones(m,n),'symmetric');

for k=1:d/2

f=imsubtract(f,ordfilt2(g,k,ones(m,n),'symmetric'));

end

for k=(m\*n-(d/2)+1):m\*n

f=imsubtract(f,ordfilt2(g,k,ones(m,n),'symmetric'));

end

f=f/(m\*n-d);

**Code**

A=imread('2.jpg');

[M N]=size(A);

R=imnoise2('gaussian',M,N,0.05,0.05);

B=im2double(A);

F=B+R;

g=spfilt(F,'amean');

h=spfilt(F,'gmean');

h1=spfilt(F,'hmean');

c=spfilt(F,'chmean');

max=spfilt(F,'max');

min=spfilt(F,'min');

median=spfilt(F,'median');

mid=spfilt(F,'midpoint');

atrim=spfilt(F,'atrimmed');

subplot(3,4,1);imshow(A,[]);title('Original Image');

subplot(3,4,2);imshow(F,[]);title('Noisy Image');

subplot(3,4,3);imshow(g,[]);title('Arithmetic Mean Filter');

subplot(3,4,4);imshow(h,[]);title('Geometric Mean Filter');

subplot(3,4,5);imshow(h1,[]);title('Harmonic Mean Filter');

subplot(3,4,6);imshow(c,[]);title('Contraharmonic Filter');

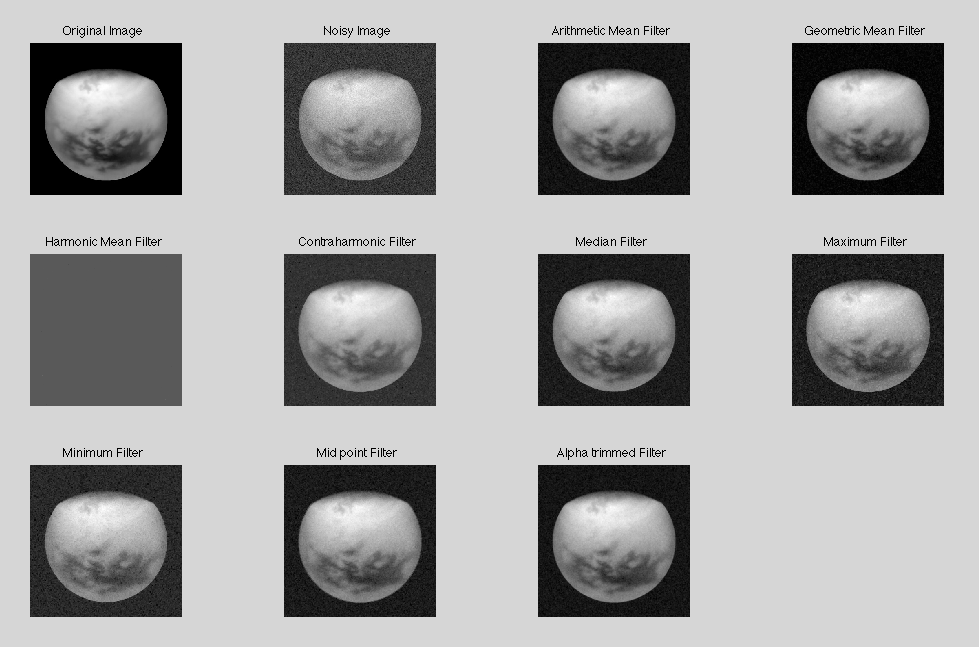
subplot(3,4,7);imshow(median,[]);title('Median Filter');

subplot(3,4,8);imshow(max,[]);title('Maximum Filter');

subplot(3,4,9);imshow(min,[]);title('Minimum Filter');

subplot(3,4,10);imshow(mid,[]);title('Mid point Filter');

subplot(3,4,11);imshow(atrim,[]);title('Alpha trimmed Filter');



Alpha trimmed produces best results.

Huffman code

sig = repmat([3 3 1 3 3 3 3 3 2 3],1,50); % Data to encode

symbols = [1 2 3]; % Distinct data symbols appearing in sig

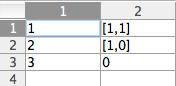
p = [0.1 0.1 0.8]; % Probability of each data symbol

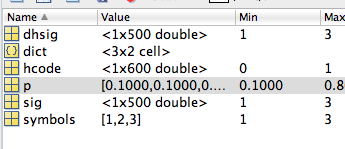
dict = huffmandict(symbols,p); % Create the dictionary.

hcode = huffmanenco(sig,dict); % Encode the data.

dhsig = huffmandeco(hcode,dict); % Decode the code.

**dict**





**huffmandict:** Generate Huffman code dictionary for source with known probability model

**Syntax**

[dict,avglen] = huffmandict(symbols,p)

[dict,avglen] = huffmandict(symbols,p,N)

[dict,avglen] = huffmandict(symbols,p,N,variance)

**Description**

The huffmandict function generates a Huffman code dictionary corresponding to a source with a known probability model. The required inputs are symbols, which lists the distinct signal values that the source produces. It can have the form of a numeric vector, numeric cell array, or alphanumeric cell array. If it is a cell array, it must be either a row or a column. p, a probability vector whose kth element is the probability with which the source produces the kth element of symbols. The length of p must equal the length of symbols.

The outputs of huffmandict are:

dict, a two-column cell array in which the first column lists the distinct signal values from symbols and the second column lists the corresponding Huffman codewords. In the second column, each Huffman codeword is represented as a numeric row vector.

avglen, the average length among all codewords in the dictionary, weighted according to the probabilities in the vector p.

**huffmanenco**

comp = huffmanenco(sig,dict)

**Description**

comp = huffmanenco(sig,dict) encodes the signal sig using the Huffman codes described by the code dictionary dict. The argument sig can have the form of a numeric vector, numeric cell array, or alphanumeric cell array. If sig is a cell array, it must be either a row or a column. dict is an N-by-2 cell array, where N is the number of distinct possible symbols to be encoded. The first column of dict represents the distinct symbols and the second column represents the corresponding codewords. Each codeword is represented as a numeric row vector, and no codeword in dict can be the prefix of any other codeword in dict. Y

**huffmandeco :Huffman decoder**

**Syntax:** dsig = huffmandeco(comp,dict)

**Description**

dsig = huffmandeco(comp,dict) decodes the numeric Huffman code vector comp using the code dictionary dict. The argument dict is an N-by-2 cell array, where N is the number of distinct possible symbols in the original signal that was encoded as comp. The first column of dict represents the distinct symbols and the second column represents the corresponding codewords. Each codeword is represented as a numeric row vector, and no codeword in dict is allowed to be the prefix of any other codeword in dict. You can generate dict using the huffmandict function and comp using the huffmanenco function. If all signal values in dict are numeric, dsig is a vector; if any signal value in dict is alphabetical, dsig is a one-dimensional cell array.

**Arithmetic Coding:**

a = imread('11.jpg');

b = a(1:10 , 1:10);

low = 0.0;

high = 1.0;

[m n] = size(b);

h=imhist(b);

[count x] = imhist(b);

p = count./(m\*n);

low\_arr = zeros(1,256);

high\_arr = zeros(1,256);

high\_arr(1) = p(1);

for i=2:256

low\_arr(i) = high\_arr(i-1);

high\_arr(i) = low\_arr(i) + p(i);

end

b\_flat = b(:);

for i=1:size(b\_flat, 1)

range= high-low;

temp = low;

low = low + range\*(low\_arr(b\_flat(i)+1));

high = temp + range\*(high\_arr(b\_flat(i)+1));

end

low = vpa(low,50);

save low ;

Output:

low =

0.50773945384392660251648976554861292243003845214844