Digital Image Processing

BB1603391 116033910045 修宇亮

Problem 4 Requirement

4. Generating different noise and comparing different noise reduction methods

In this problem, you are required to write a program to generate different types of random noises started from the Uniform noise and Gaussian noise. (one of the reference may be "Digital image processing using Matlab" PP.143-150. And then add some of these noises to the circuit image (I will provide the image on ftp) and investigate the different mean filters and order statistics as the textbook did at pages 344-352.

Problem 4 solution

imnoise2.m

MATLAB

```matlab
function R = imnoise2(type,M,N,a,b)
% Imnoise2 to creates a random array with the specified PDF
% R = IMNOISE2(TYPE,M,N,A,B) generates an array, R, of size M by N whose
% elements are random numbers of the specified TYPE with parameters A and
% B. IF ony TYPE is included in the input argument list, a single random
% number of the specified TYPE and and default parameters show below is
% generated. If only TYPE, M and N are provided, the default parameters
% shown below are used. If M=N=1, IMNOISE2 generates a single random number
% of the specified TYPE and parameters A and B.
%
% Valid values for TYPE and parameters A and B are:
% 'uniform'        Uniform random numbers in the interval (A,B).
%                  The default values are (0,1).
%
% 'guassian'       Gaussian random numbers with mean A and standard
%                  deviation B.The default values are A = 0, B = 1.
%
```

```
% 'salt & pepper'  Salt and pepper numbers  of amplitude 1 with probability
%                  Pa  = A, and amplitude 0 with probability Pb = B. The
%                  default values are Pa = Pb = A = B = 0.05. Note that the
%                  noise has values 0(with probability Pa = A) and 1(with
%                  probability Pb = B), so scaling is necessary if values
%                  other than 0 and 1 are required. The noise matrix R is
%                  assigned three values. If R(x,y) = 0, the noise at (x,y)
%                  is pepper(black). If R(x,y ) = 1, the noise at (x,y) is
%                  salt(white). If R(x,y) = 0.5, there is no noise assigned
%                  to coordinates (x,y).
%
% 'lognormal'      Lognormal numbers with offset A and shape parameter B.
%                  The defaults are A = 1 and B = 0.25.
%
% 'rayleigh'       Rayleigh noise with parameters A and B. The default
%                  values are A = 0 and B = 1.
%
% 'exponential'    Exponential random numbers with parameter A. The default
%                  value is A = 1.
%
% 'erlang'         Erlang(gamma) random numbers with parameters A and B. B
%                  must be a positive integer. The defaults are A = 2 and B
%                  = 5. Erlang random numbers are approximated as the sum
%                  of B exponential random numbers.
%
% set default values.
if nargin == 1
    a = 0; b = 1;
    M = 1; N = 1;
elseif nargin == 3
    a = 0; b = 1;
end
%as we need only small letters as the type so...
switch lower(type)
    case 'uniform'
        R = a + (b-a)*rand(M,N);
    case 'gaussian'
        R = a + b*randn(M,N);
    case 'salt & pepper'
        if nargin <= 3
        a = 0.05; b = 0.05;
        end
% check to make sure that Pa + Pb is not > 1.
        if (a + b) > 1
            error('The sum of the Pa and Pb cannot exeed 1.')
        end
        R(1:M,1:N) = 0.5;
% Generate an M by N array of uniformly distributed random numbers in the
```

```
% range (0,1). Then, Pa*(M*N) of them will have values <= a. The
% coordinates of these points we call 0 (pepper noise). Similarly, Pb*(M*N)
% points will have values in the range > a & <= (a+b). These we call
% (salt noise).
        X = rand(M,N);
        c = find(X<=a);
        R(c) = 0;
        u = a + b;
        c = find(X > a & X <= u);
        R(c) = 1;
    case 'lognormal'
        if nargin<=3
            a = 1; b = 0.25;
        end
        R = a*exp(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('the value of a must b positive for exponential operation')
        end
        k = -1/a
        R = k*log(1 - rand(M,N));
    case 'erlang'
        if nargin <= 3
            a = 2; b = 5;
        end
        if (b ~= round(b)| b <= 0)
            error('Parameter b should b a negative value for erlang')
        end
        k = -1/a;
        R = zeros(M,N);
        for j = 1:b
            R = R + k*log(1 - rand(M,N));
        end
    otherwise
        error('Unknown distribution type.')
end

```

spfilt.m

MATLAB

```
function f = spfilt(g, type, varargin)
```

```
%SPFILT Performs linear and nonlinear spatial filtering.
%   F = SPFILT(G, TYPE, M, N, PARAMETER) performs spatial filtering
%   of image G using a TYPE filter of size M-by-N. Valid calls to
%   SPFILT are as follows:
%
%     F = SPFILT(G, 'amean', M, N)       Arithmetic mean filtering.
%     F = SPFILT(G, 'gmean', M, N)       Geometric mean filtering.
%     F = SPFILT(G, 'hmean', M, N)       Harmonic mean filtering.
%     F = SPFILT(G, 'chmean', M, N, Q)   Contraharmonic mean
%                                        filtering of order Q. The
%                                        default is Q = 1.5.
%     F = SPFILT(G, 'median', M, N)      Median filtering.
%     F = SPFILT(G, 'max', M, N)         Max filtering.
%     F = SPFILT(G, 'min', M, N)         Min filtering.
%     F = SPFILT(G, 'midpoint', M, N)    Midpoint filtering.
%     F = SPFILT(G, 'atrimmed', M, N, D) Alpha-trimmed mean filtering.
%                                        Parameter D must be a
%                                        nonnegative even integer;
%                                        its default value is D = 2.
%
%   The default values when only G and TYPE are input are M = N = 3,
%   Q = 1.5, and D = 2.

[m, n, Q, d] = processInputs(varargin{:});

% Do the filtering.
switch type
case 'amean'
   w = fspecial('average', [m n]);
   f = imfilter(g, w, 'replicate');
case 'gmean'
   f = gmean(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 = imdilate(g, ones(m, n));
case 'min'
   f = imerode(g, ones(m, n));
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'
   f = alphatrim(g, m, n, d);
```

```
otherwise
   error('Unknown filter type.')
end

%-------------------------------------------------------------------%
function f = gmean(g, m, n)
%  Implements a geometric mean filter.
[g, revertclass] = tofloat(g);
f = exp(imfilter(log(g), ones(m, n), 'replicate')).^(1 / m / n);
f = revertclass(f);

%-------------------------------------------------------------------%
function f = harmean(g, m, n)
%  Implements a harmonic mean filter.
[g, revertclass] = tofloat(g);
f = m * n ./ imfilter(1./(g + eps),ones(m, n), 'replicate');
f = revertclass(f);

%-------------------------------------------------------------------%
function f = charmean(g, m, n, q)
%  Implements a contraharmonic mean filter.
[g, revertclass] = tofloat(g);
f = imfilter(g.^(q+1), ones(m, n), 'replicate');
f = f ./ (imfilter(g.^q, ones(m, n), 'replicate') + eps);
f = revertclass(f);

%-------------------------------------------------------------------%
function f = alphatrim(g, m, n, d)
%  Implements an alpha-trimmed mean filter.
if (d <= 0) | (d/2 ~= round(d/2))
    error('d must be a positive, even integer.')
end
[g, revertclass] = tofloat(g);
f = imfilter(g, ones(m, n), 'symmetric');
for k = 1:d/2
   f = f - ordfilt2(g, k, ones(m, n), 'symmetric');
end
for k = (m*n - (d/2) + 1):m*n
   f = f - ordfilt2(g, k, ones(m, n), 'symmetric');
end
f = f / (m*n - d);
f = revertclass(f);

%-------------------------------------------------------------------%
function [m, n, Q, d] = processInputs(varargin)
m = 3;
n = 3;
Q = 1.5;
```

```
d = 2;
if nargin > 0
    m = varargin{1};
end
if nargin > 1
    n = varargin{2};
end
if nargin > 2
    Q = varargin{3};
    d = varargin{3};
end
```

prob.m

```
MATLAB

orig_img = imread('Circuit.tif');
noise_type = {'uniform','gaussian','salt & pepper','lognormal','rayleigh','exponentia
filter_type = {'amean','gmean','hmean','chmean','median','max','min','midpoint','atri
[M,N] = size(orig_img);
bins  = 51;

for i=3:3
    set(gcf, 'position', [0 0 1200 300]);
    subplot(1,3,1),imshow(orig_img);title('original image');
    switch i
        case 1
            noise = imnoise2(noise_type{i},M,N,0,0.1);
        case 2
            noise = imnoise2(noise_type{i},M,N,0,0.1);
        case 3
            % salt
            noise = imnoise2(noise_type{i},M,N,0,0.1);
            noise_img1 = orig_img;
            noise_img1(noise == 1) = 255;
            % pepper
            noise = imnoise2(noise_type{i},M,N,0.1,0);
            noise_img2 = orig_img;
            noise_img2(noise == 0) = 0;
        otherwise
            noise = imnoise2(noise_type{i},M,N);

    end

    subplot(1,3,2),hist(noise(:),bins);title([noise_type{i},' noise']);
    if i ~= 3
        noise_img = im2uint8(im2double(orig_img)+noise);
```

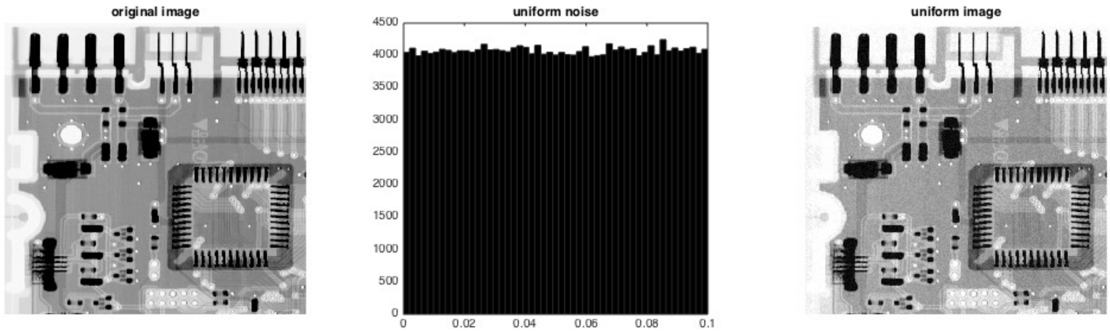
```
        subplot(1,3,3),imshow(noise_img);title([noise_type{i},' image']);
    else
        subplot(1,3,3),imshow(noise_img1);title([noise_type{i},' image']);
%         subplot(1,3,3),imshow(noise_img2);title([noise_type{i},' image']);
    end

end

i=5;
j=5;
num = [4,4,4,4,3];
switch i
    case 1
        % uniform
        noise = imnoise2(noise_type{i},M,N,0,0.1);
        noise_img = im2uint8(im2double(orig_img)+noise);
    case 2
        % gaussian
        noise = imnoise2(noise_type{i},M,N,0,0.1);
        noise_img = im2uint8(im2double(orig_img)+noise);
    case 3
        % salt
        noise = imnoise2(noise_type{i},M,N,0,0.1);
        noise_img1 = orig_img;
        noise_img1(noise == 1) = 255;
        % pepper
        noise = imnoise2(noise_type{i},M,N,0.1,0);
        noise_img2 = orig_img;
        noise_img2(noise == 0) = 0;
    case 4
        % salt & peper
        noise = imnoise2(noise_type{3},M,N,0.1,0.1);
        noise_img = orig_img;
        noise_img(noise == 1) = 255;
        noise_img(noise == 0) = 0;
    case 5
        % uniform
        noise = imnoise2(noise_type{1},M,N,0,0.1);
        noise_img1 = im2uint8(im2double(orig_img)+noise);
        % salt & peper
        noise = imnoise2(noise_type{3},M,N,0.1,0.1);
        noise_img2 = noise_img1;
        noise_img2(noise == 1) = 255;
        noise_img2(noise == 0) = 0;


end
switch j
```

```
    case 1
        set(gcf, 'position', [0 0 num(j)*400 300]);
        subplot(1,num(j),1),imshow(orig_img);title('original image');
        subplot(1,num(j),2),imshow(noise_img);title([noise_type{i},' image']);
        amean_uniform_img = spfilt(noise_img,'amean');
        subplot(1,num(j),3),imshow(amean_uniform_img);title([filter_type{1},' result']
        gmean_uniform_img = spfilt(noise_img,'gmean');
        subplot(1,num(j),4),imshow(gmean_uniform_img);title([filter_type{2},' result']
    case 2
        set(gcf, 'position', [0 0 num(j)*400 300]);
        subplot(1,num(j),1),imshow(noise_img1);title('salt image');
        subplot(1,num(j),2),imshow(noise_img2);title('pepper image');
        chmean_salt_img = spfilt(noise_img1,'chmean',3,3,-1.5);
        subplot(1,num(j),3),imshow(chmean_salt_img);title('salt result');
        chmean_pepper_img = spfilt(noise_img2,'chmean',3,3,1.5);
        subplot(1,num(j),4),imshow(chmean_pepper_img);title('pepper result');
    case 3
        set(gcf, 'position', [0 0 num(j)*400 300]);
        subplot(1,num(j),1),imshow(noise_img);title('salt&pepper image');
        median_salt_pepper_img1 = spfilt(noise_img,'median');
        subplot(1,num(j),2),imshow(median_salt_pepper_img1);title('median 1st');
        median_salt_pepper_img2 = spfilt(median_salt_pepper_img1,'median');
        subplot(1,num(j),3),imshow(median_salt_pepper_img2);title('median 2nd');
        median_salt_pepper_img3 = spfilt(median_salt_pepper_img2,'median');
        subplot(1,num(j),4),imshow(median_salt_pepper_img3);title('median 3rd');
    case 4
        set(gcf, 'position', [0 0 num(j)*400 300]);
        subplot(1,num(j),1),imshow(noise_img1);title('salt image');
        subplot(1,num(j),2),imshow(noise_img2);title('pepper image');
        min_salt_img = spfilt(noise_img1,'min',3,3);
        max_pepper_img = spfilt(noise_img2,'max',3,3);
        subplot(1,num(j),3),imshow(min_salt_img);title('min result');
        subplot(1,num(j),4),imshow(max_pepper_img);title('max result');
    case 5
        set(gcf, 'position', [0 0 num(j)*400 600]);
        subplot(2,num(j),1),imshow(noise_img1);title('uniform image');
        subplot(2,num(j),2),imshow(noise_img2);title('salt&pepper image');
        amean_result = spfilt(noise_img2,'amean',5,5);
        subplot(2,num(j),3),imshow(amean_result);title('amean result');
        gmean_result = spfilt(amean_result,'gmean',5,5);
        subplot(2,num(j),4),imshow(gmean_result);title('gmean result');
        median_result = spfilt(gmean_result,'median',5,5);
        subplot(2,num(j),5),imshow(median_result);title('median result');
        altrimmed_result = spfilt(median_result,'atrimmed',5,5,4);
        subplot(2,num(j),6),imshow(altrimmed_result);title('atrimmed result');
```

```
end
```

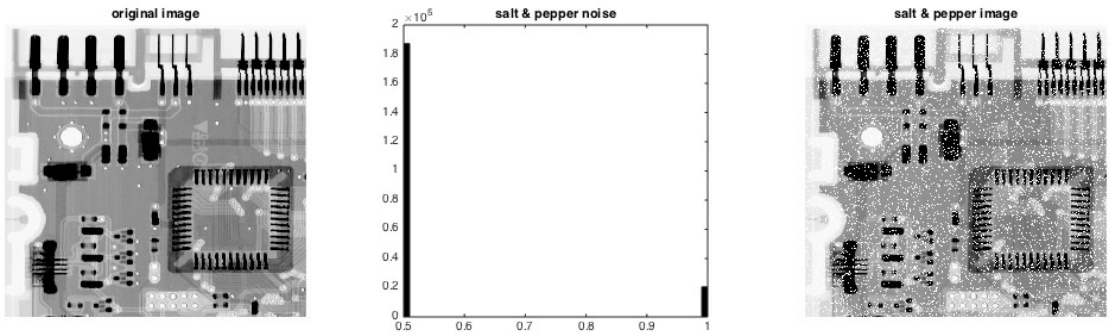
Noise Image

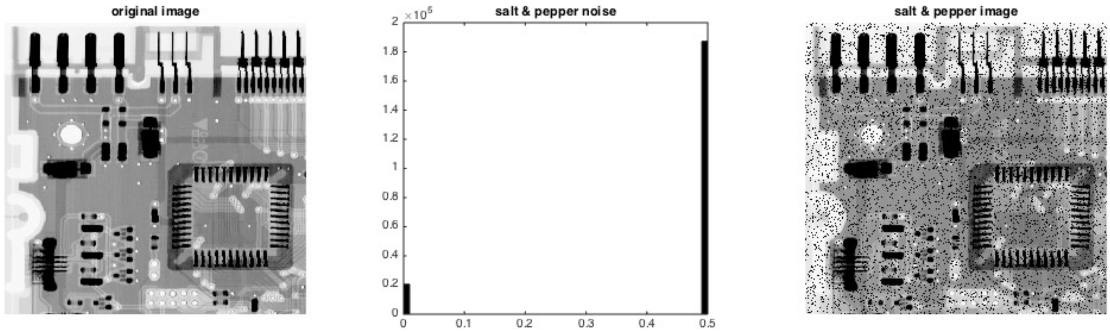
Uniform Noise Image



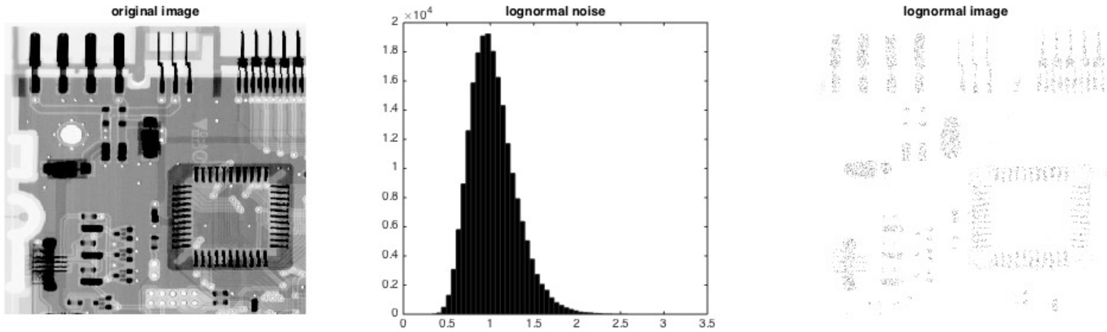
Gaussian Noise Image

Salt & Pepper Noise Image

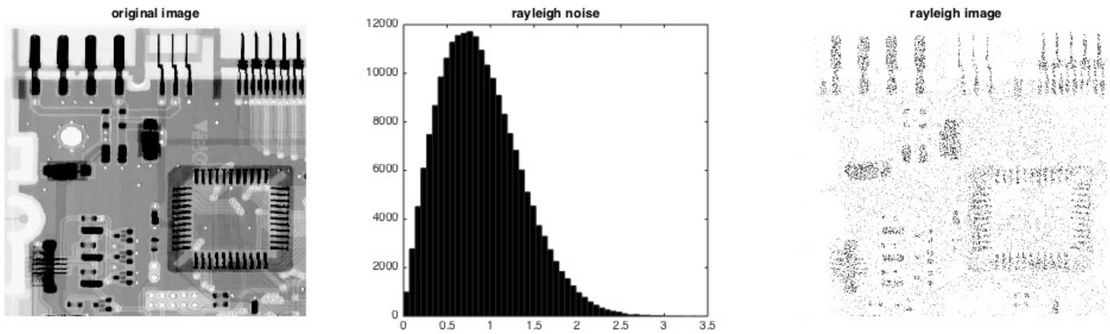




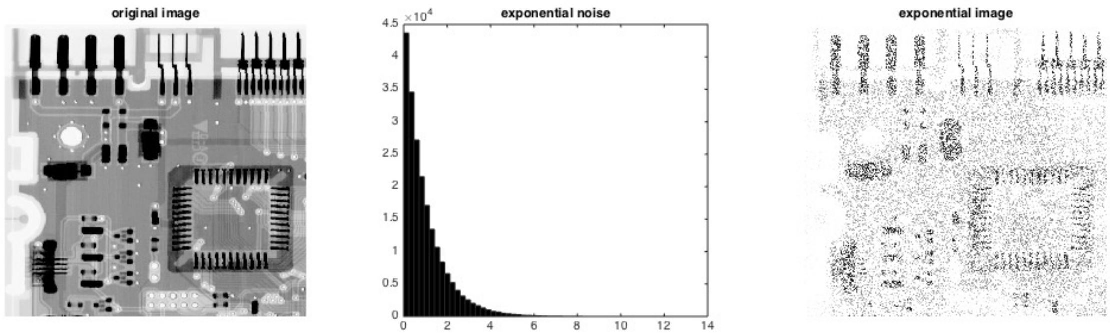
Lognormal Noise Image



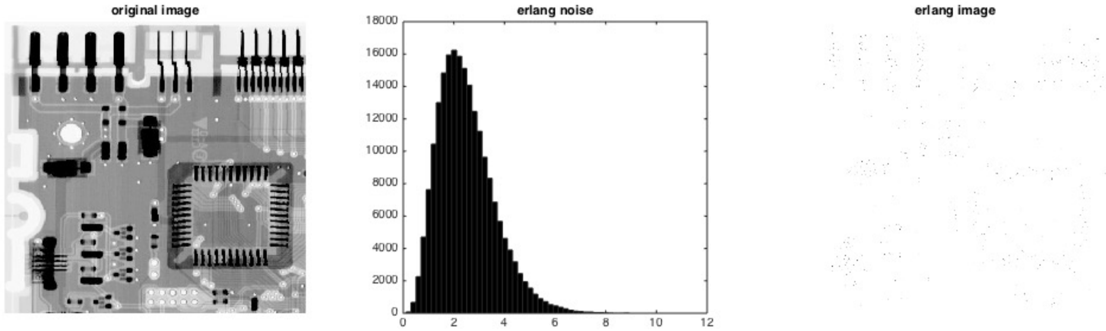
Rayleigh Noise Image



Exponential Noise Image

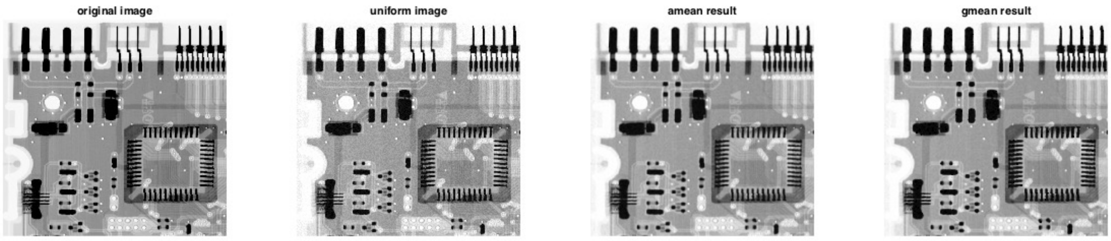


Erlang Noise Image

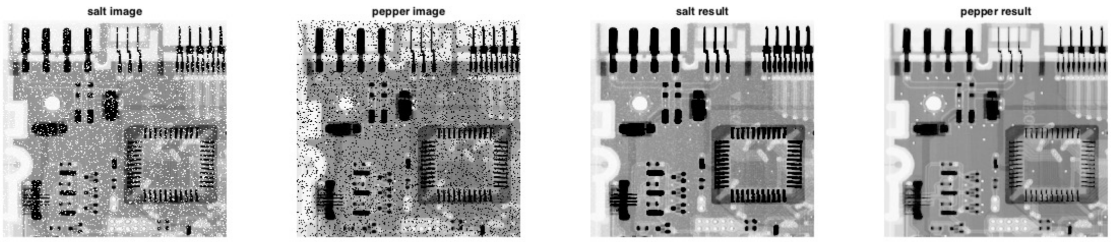


Filter

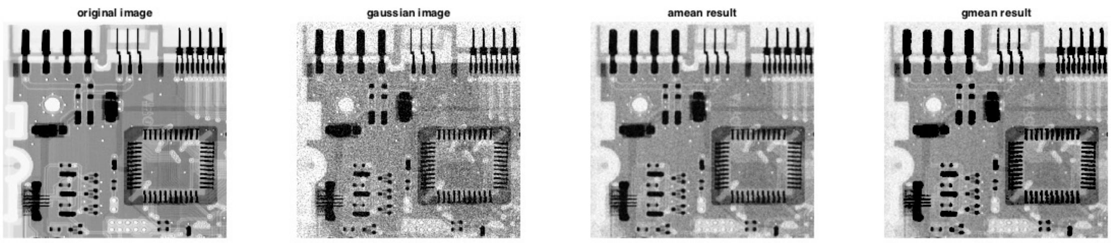
Arithmetic & Geometric Mean Filter on Uniform Noise



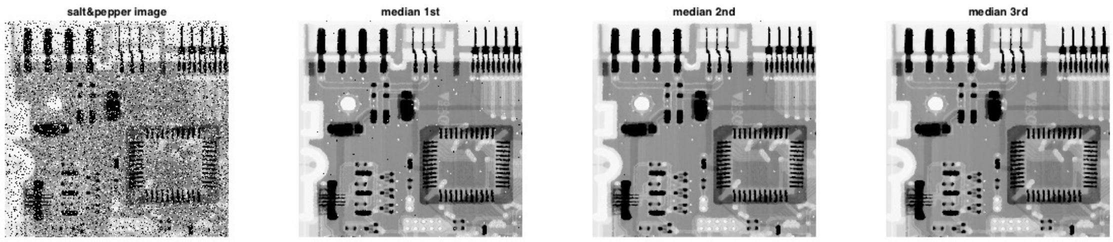
Contraharmonic Mean Filter on Salt & Pepper Noise



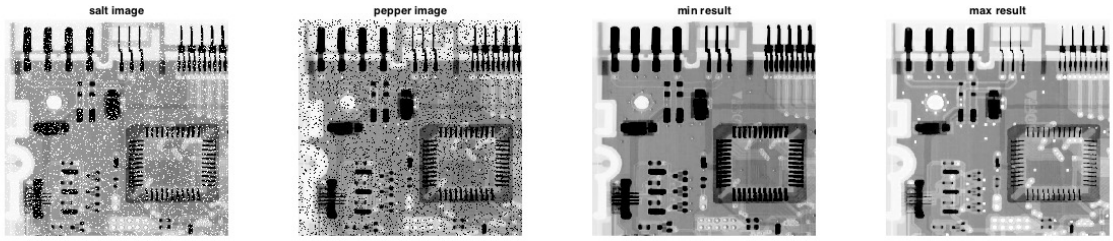
Arithmetic & Geometric Mean Filter on Gaussian Noise



Median FIlter on Salt & Pepper Noise



Max & Min Filter on Salt & Pepper Noise



Combined Filters on Gaussian & Salt & Pepper Noise