Programming Assignment 1

ECE 661 Computer Vision

Histogram Equalization, Convolution, Separable filters

Name: Gowri Kurthkoti Sridhara Rao

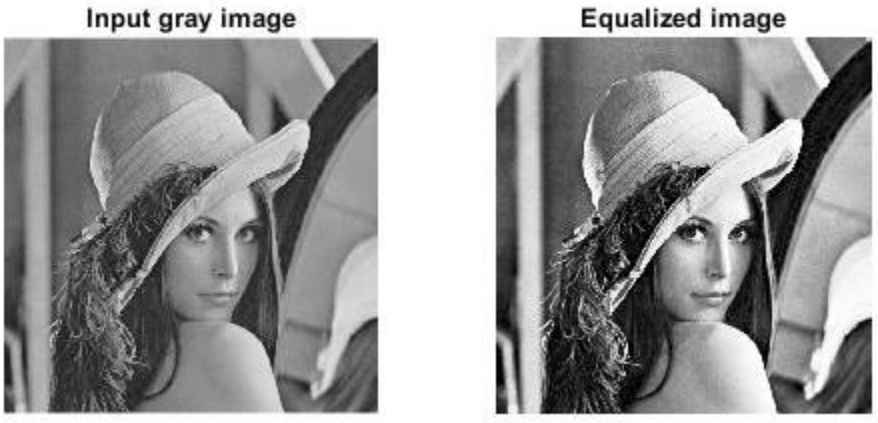
Part 1

1. Histogram Equalization
   Histogram equalization was performed on the given images on MATLAB. Below are the results for each image.

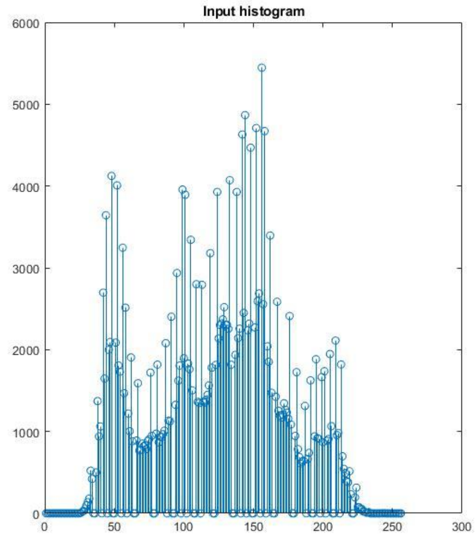
   Image 1

   - Input image

   - Input gray image and equalized image



- Input and output histograms


Input histogram

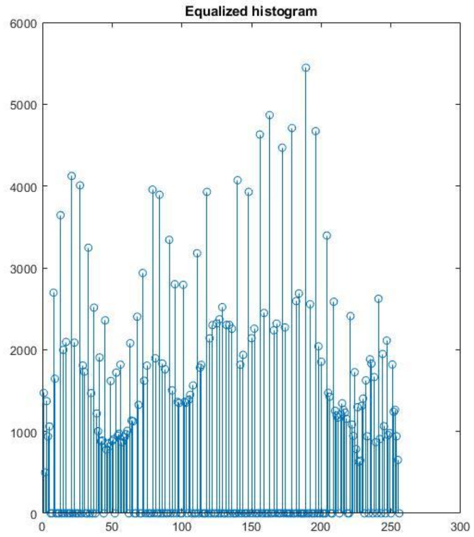
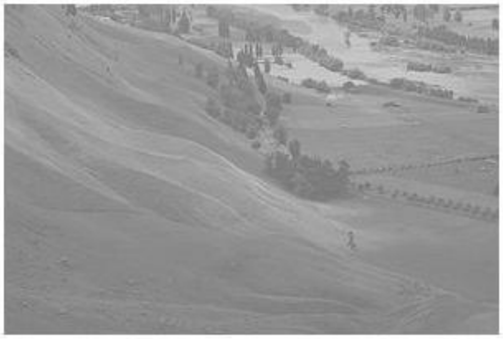
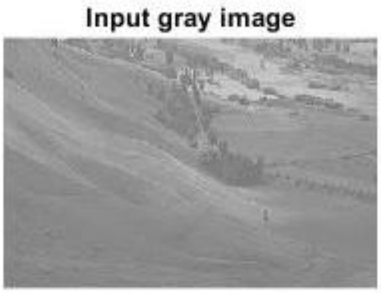

Equalized histogram

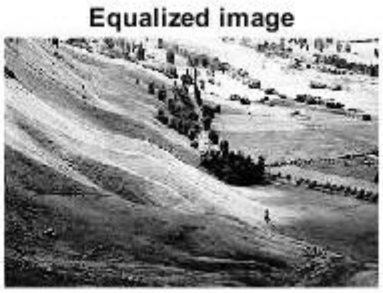
Image 2

- Input image



- Input gray and equalized image


Input gray image


Equalized image

- Input and equalized histograms

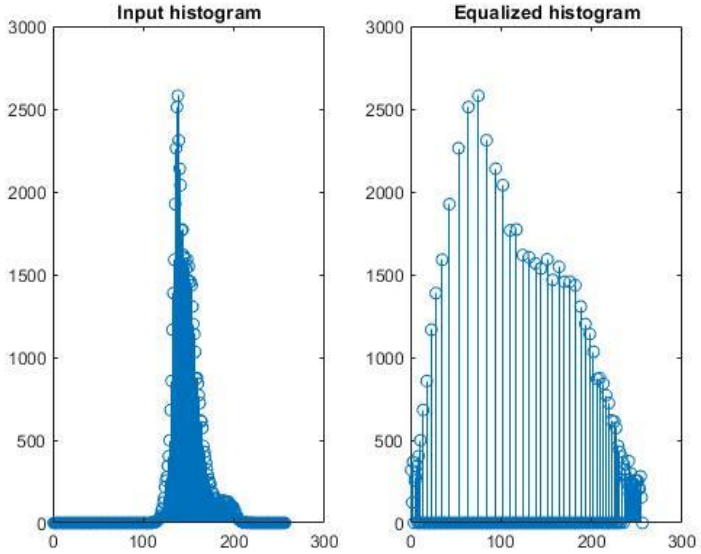
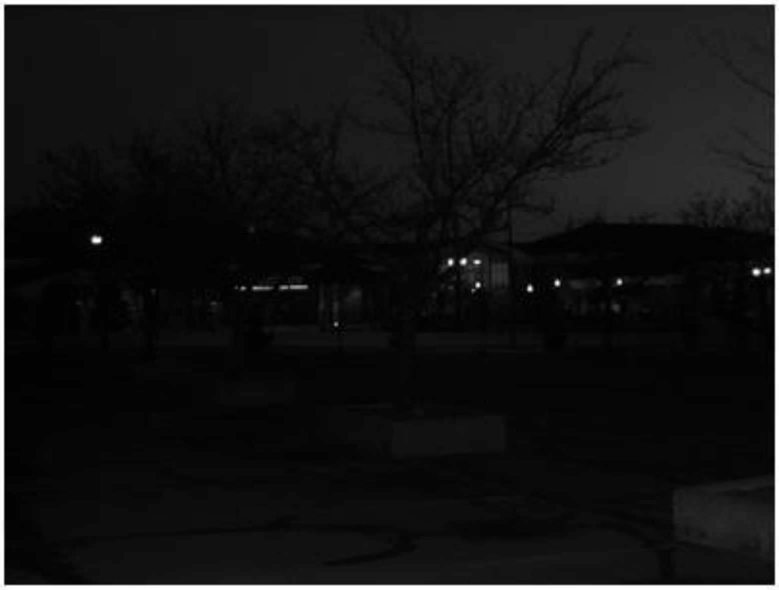
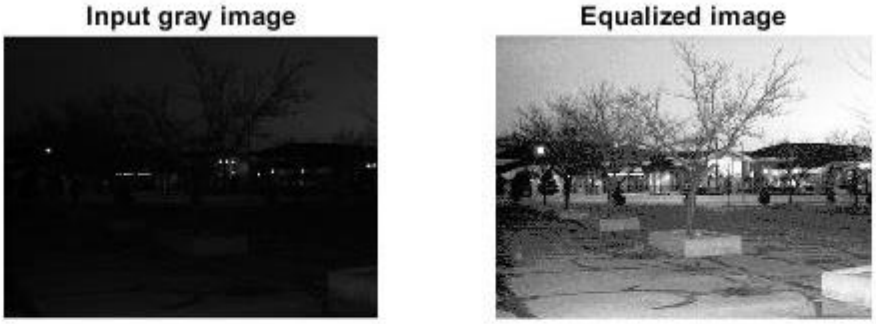


Image 3

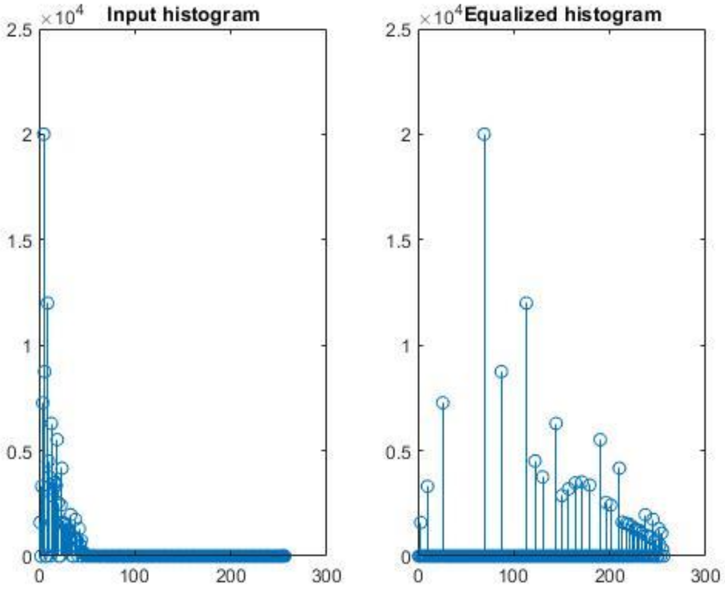
- Input image



- Input gray and equalized image



- Input and equalized histograms



Code

- Histogram function

```
function [histogram] = histo(img_gray)
%HISTO Computes the histogram
%   Input to this function is a gray scale image whose histogram has to
be
%   computed. Output is a [1,256] sized double which holds the histogram
of
%   the image.
img_gray=double(img_gray); %Convert image to double for easier processing
histogram=zeros(1,256); %Initialize output double
[rows,cols]=size(img_gray); %Initialize
```

```
for x=1:rows
    for y=1:cols
        value=img_gray(x,y); %Extract the value of the pixel
        if (value==0)
            value=value+1; %If pixel value is 0, increment it to 1
        end
        histogram(value)= histogram(value)+1; %Increment count of the
pixel
    end
end

end
```

- Histogram equalization function

```
function [eq_img,out_hist] = hist_equalize(img_gray,histogram)
%HIST_EQUALIZE This function performs histogram equalization of image
%   Input to this function is the image that has to be equalized. The
image
%   histogram is equalized and output image along with the equalized
%   histogram is provided as output.
hist_sum=zeros(1,256); %Initialize histogram sum variable
[rows,cols]=size(img_gray);
val=0; %Initialize value to 0
for i=1:256
    hist_sum(i)=histogram(i)+val; %Compute histogram sum
    val=hist_sum(i); %Update val variable
end
norm_sum=zeros(1,256); %Initialize another vector to store normalized sum
for i=1:256
    norm_sum(i)=round(hist_sum(i)*(1/(rows*cols))*255); %Compute norm_sum
end
eq_img=img_gray; %Initialize output image to input image

for x=1:rows
    for y=1:cols
        value=img_gray(x,y);
        if(value==0)
            value=value+1;
        end
        change=norm_sum(value);
        eq_img(x,y)= change; %Updating output image with updated values
    end
end
out_hist=histo(eq_img); %Compute histogram of output equalized image
end
```

- Main script

```
clc;
%Taking input
img_1=imread("P1_lena_gray_512.png");
img_2=imread("P1_Unequalized_Hawkes_Bay_NZ.jpg");
img_3=imread("P1_university.png");
```

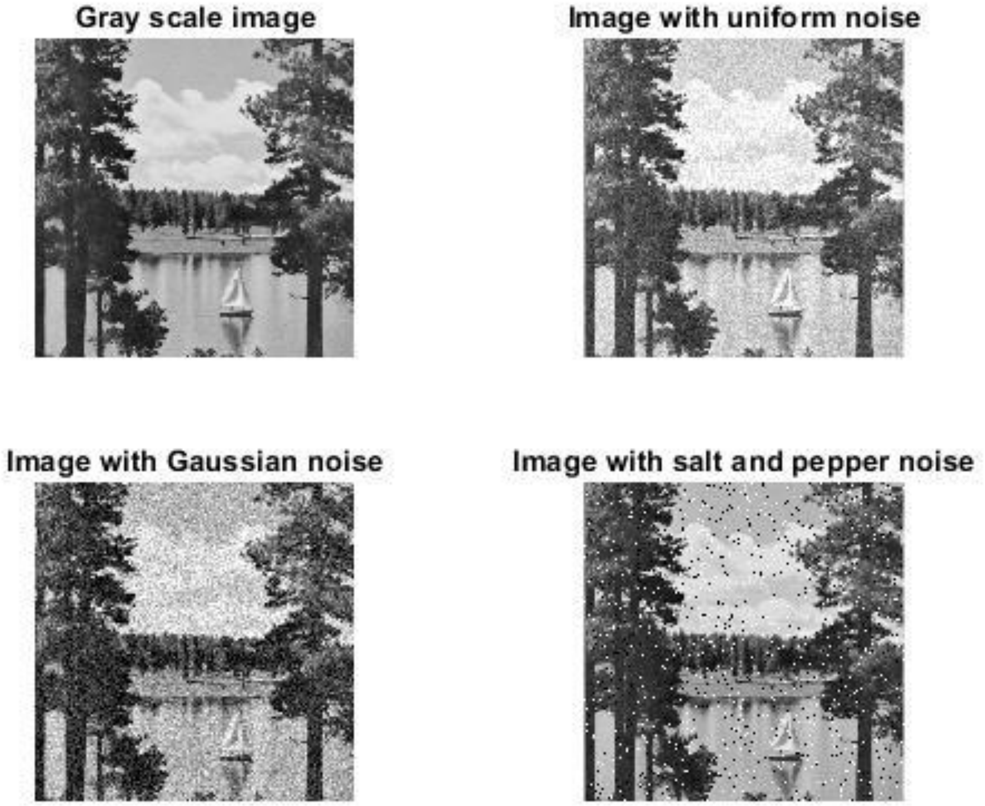
```
%%
img_rgb=img_3;
figure(1);
imshow(img_rgb);

%Converting rgb to gray if necessary
[row,col,d]=size(img_rgb);
if(d==3)
    red_img=img_rgb(:,:,1);
    green_img=img_rgb(:,:,2);
    blue_img=img_rgb(:,:,3);
    img_gray=0.21*red_img+0.72*green_img+0.07*blue_img;
else
   img_gray=img_rgb;
end
%%
[rows,cols]=size(img_gray);
% Histogram equalization
% Step 1: Histogram
%
histogram=histo(img_gray);
% step 2
% %% Histogram equalization
[img_gray_out,out_hist]=hist_equalize(img_gray,histogram);

figure(2);
subplot(1,2,1);
imshow(img_gray); title("Input gray image");
subplot(1,2,2);
imshow(img_gray_out);title("Equalized image");

figure(3);
subplot(1,2,1);
stem(histogram);title("Input histogram");
subplot(1,2,2);
stem(out_hist);title("Equalized histogram");
```

Part 2

2. Mean Filtering
   - Adding noise to images and mean filtering the images

     Image 1

     - Input gray image with added uniform, gaussian and salt and pepper noise



Gray scale image

Image with uniform noise

Image with Gaussian noise

Image with salt and pepper noise

     - All four images filtered using mean filter

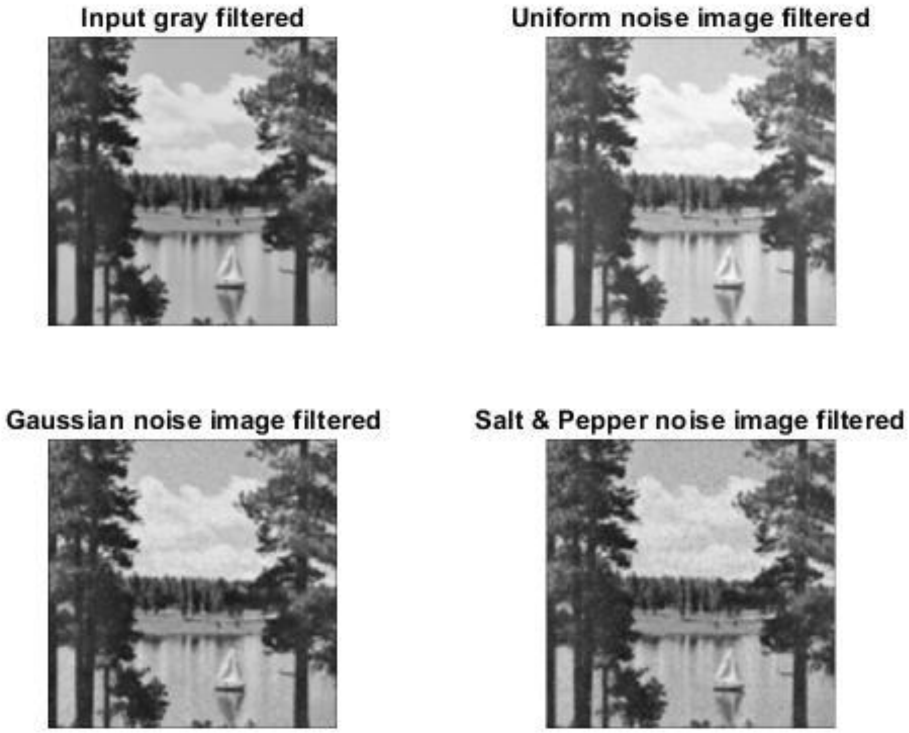
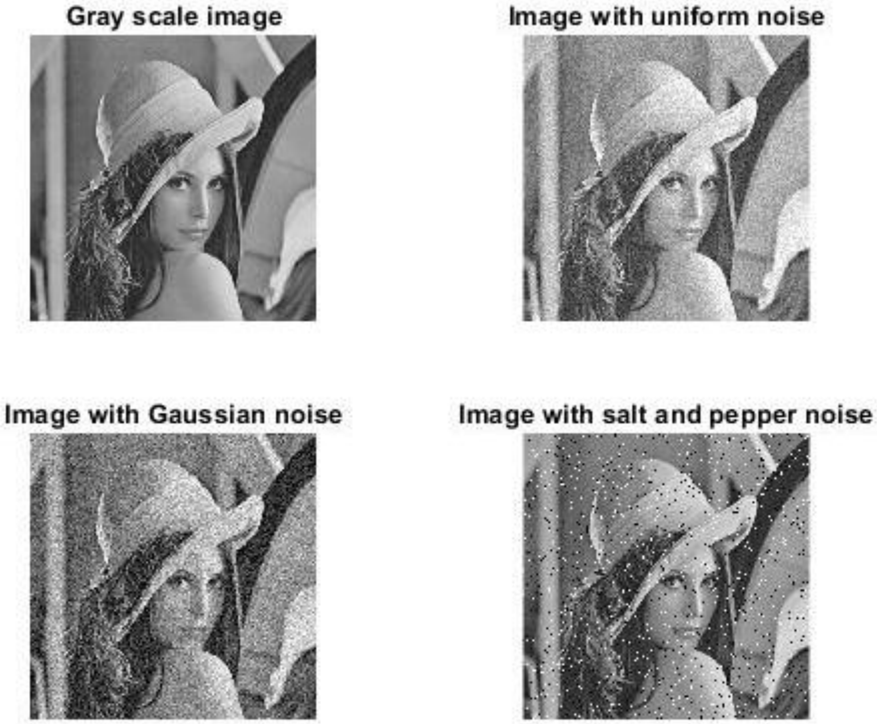

Input gray filtered
Uniform noise image filtered
Gaussian noise image filtered
Salt & Pepper noise image filtered

Image 2

- Input gray image with added uniform, gaussian and salt and pepper noise



- All four images filtered using mean filter

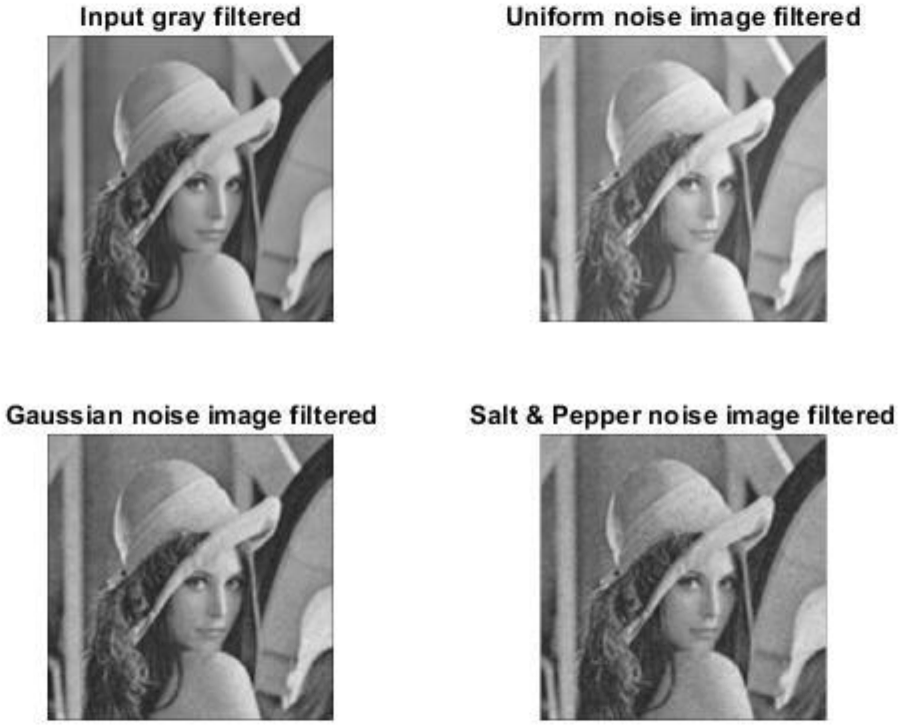
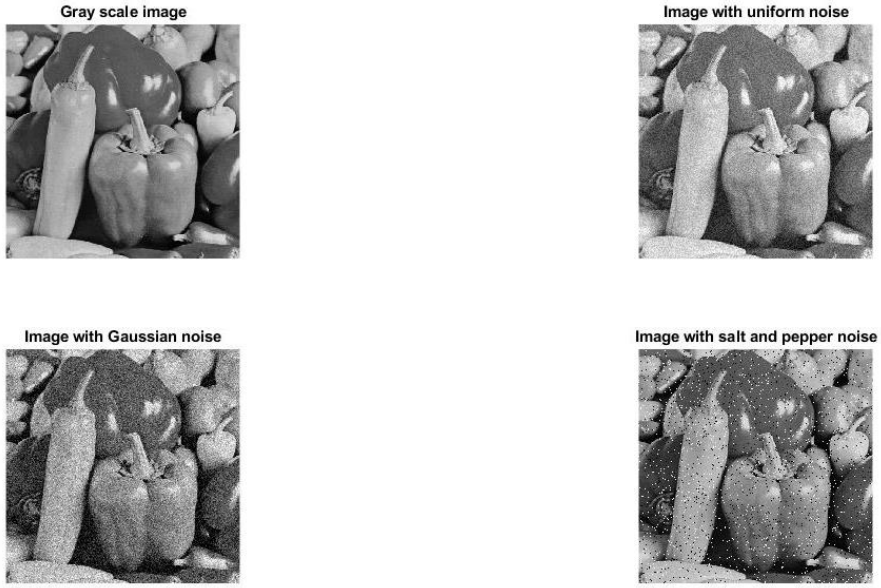

Input gray filtered
Uniform noise image filtered
Gaussian noise image filtered
Salt & Pepper noise image filtered

Image 3

- Input gray image with added uniform, gaussian and salt and pepper noise



- All four images filtered using mean filter

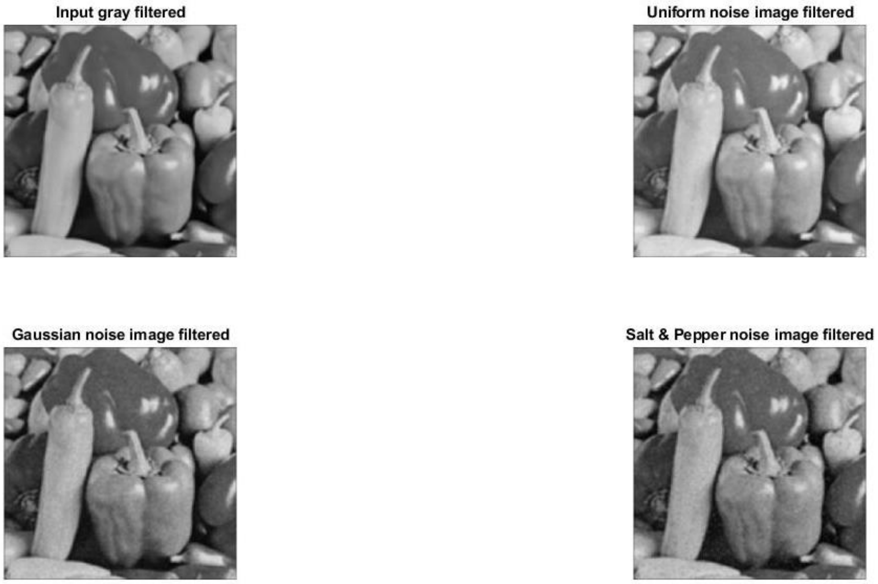
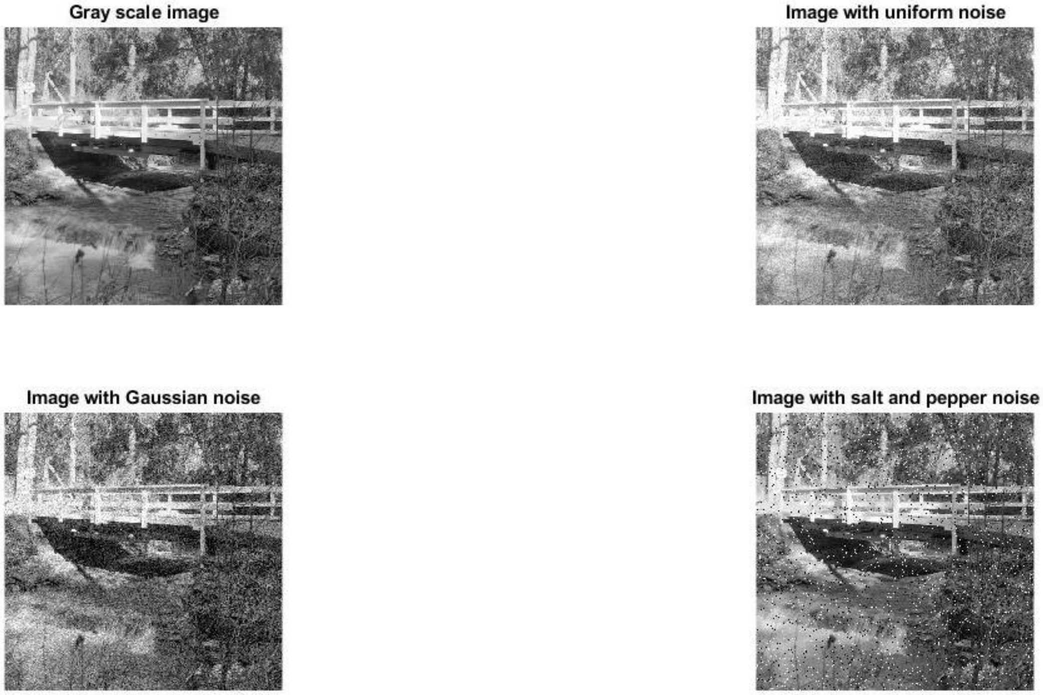
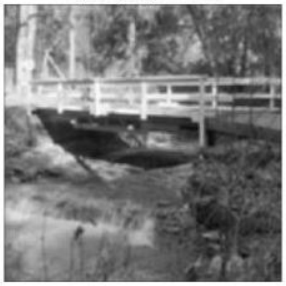

Input gray filtered
Uniform noise image filtered
Gaussian noise image filtered
Salt & Pepper noise image filtered

Image 4

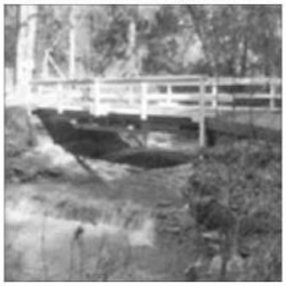
- Input gray image with added uniform, gaussian and salt and pepper noise



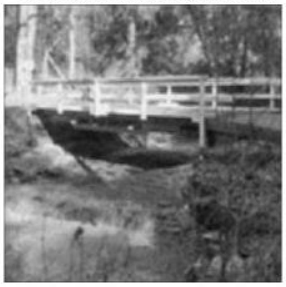
- All four images filtered using mean filter



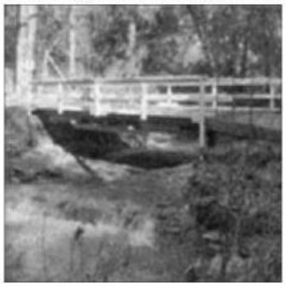
Input gray filtered



Uniform noise image filtered



Gaussian noise image filtered



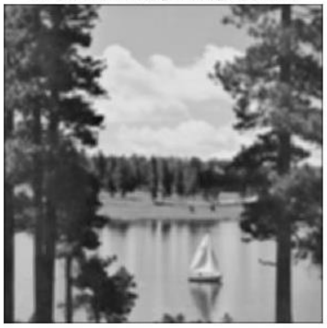
Salt & Pepper noise image filtered

- Comparison between 7X7 filter and separable filter

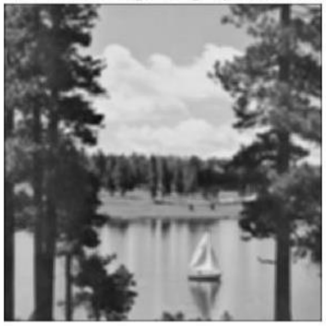
Image 1

  - Filtered images

Input image

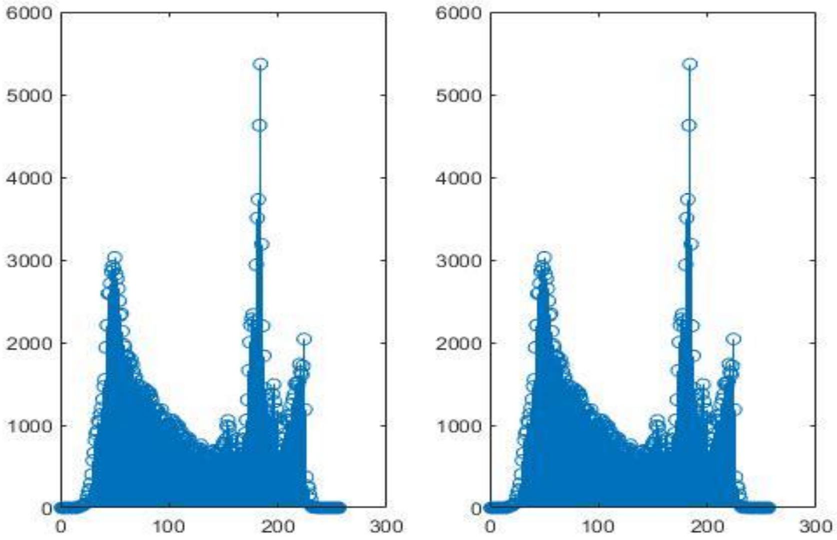


Mean filtered image using 7X7 filter



Mean filtered image using seperable filters

- Histograms of filtered images



- Timing differences

  With 7X7 filter = 3.022744s

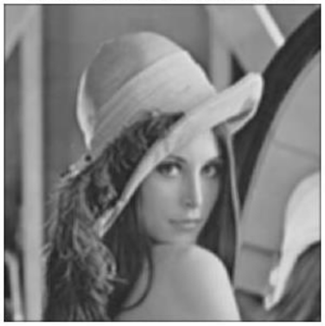
  With separable filter = 2.387335s

Image 2

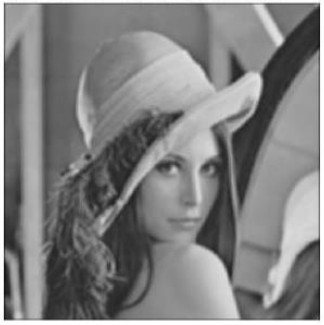
- Filtered images

Input image

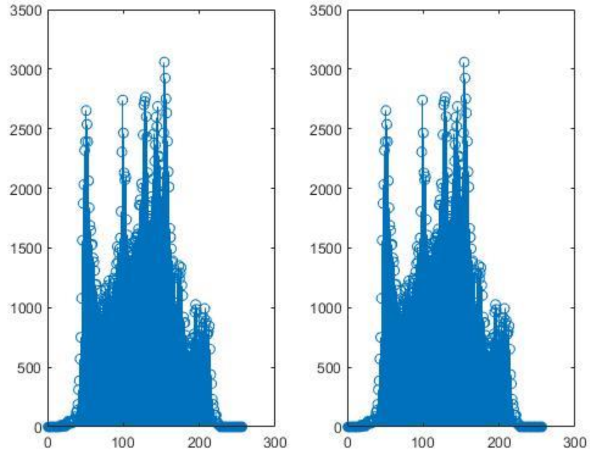
Mean filtered image using 7X7 filter



Mean filtered image using seperable filters



- Histograms of filtered images



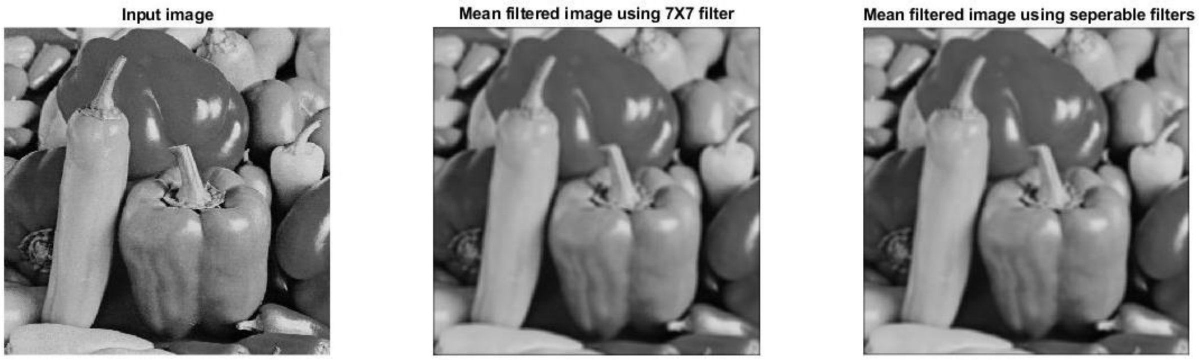
- Timing differences

  With 7X7 filter = 3.086674s

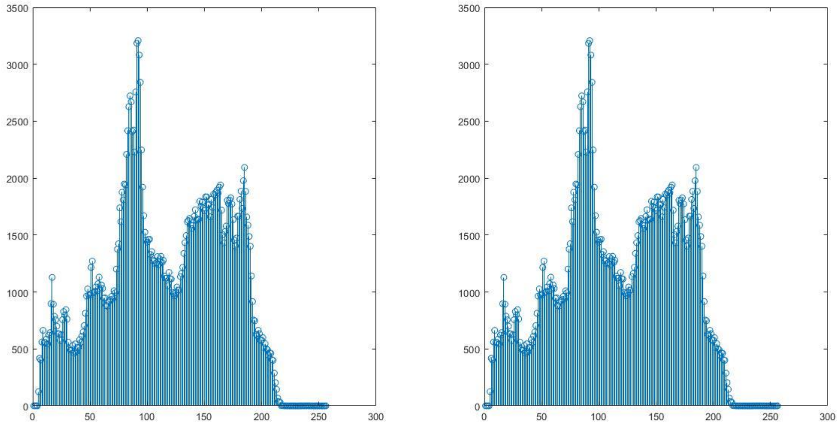
  With separable filter = 2.953981s

Image 3

- Filtered Images


Input image
Mean filtered image using 7X7 filter
Mean filtered image using seperable filters

- Histograms of filtered images



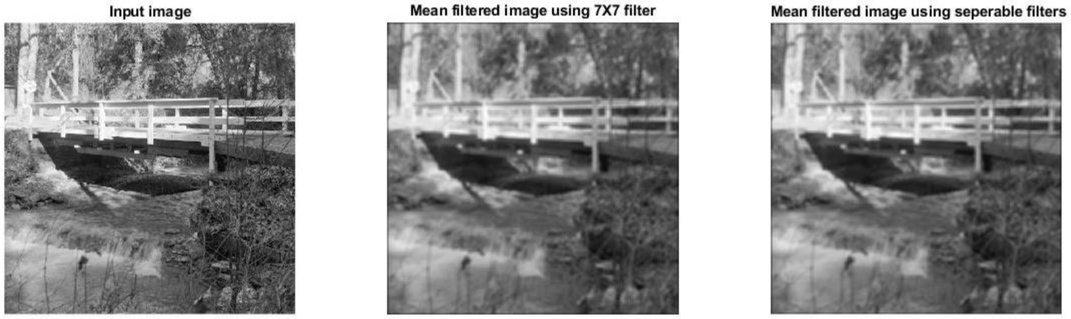
- Timing differences

  With 7X7 filter = 3.025491s

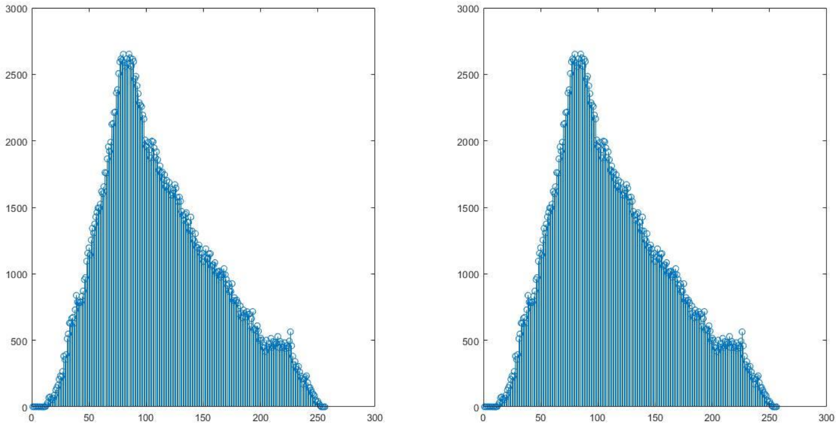
  With separable filter = 2.995880s

Image 4

- Filtered Images


Input image
Mean filtered image using 7X7 filter
Mean filtered image using seperable filters

- Histograms of filtered images



- Timing differences

  With 7X7 filter = 2.841268s

  With separable filter = 2.794355s

- Code
  - Noise part

```
%% Colour to gray
img_rgb=img_7;
[row,col,d]=size(img_rgb);
if(d==3)
    red_img=img_rgb(:,:,1);
    green_img=img_rgb(:,:,2);
    blue_img=img_rgb(:,:,3);
    img_gray=0.21*red_img+0.72*green_img+0.07*blue_img;
else
   img_gray=img_rgb;
end
%%
figure(1);
subplot(2,2,1);
imshow(img_gray); title("Gray scale image");
%Adding noise

% Uniform noise
scale=50;
img_uni_noise= img_gray + cast(scale.*rand(size(img_gray)),'uint8');

subplot(2,2,2);
imshow(img_uni_noise); title("Image with uniform noise");

%Gaussian Noise
img_GS_noise = imnoise(img_gray,'gaussian');
subplot(2,2,3);
imshow(img_GS_noise); title("Image with Gaussian noise");

%Salt pepper noise
img_sp_noise = imnoise(img_gray,'salt & pepper');
subplot(2,2,4);
imshow(img_sp_noise); title("Image with salt and pepper noise");

op1=nxnfilter(img_uni_noise,7,2);
op2=nxnfilter(img_GS_noise,7,2);
op3=nxnfilter(img_sp_noise,7,2);

figure(4);
subplot(2,2,1); imshow(cropped);title("Input gray filtered");
subplot(2,2,2);imshow(op1); title("Uniform noise image filtered");
subplot(2,2,3);imshow(op2); title("Gaussian noise image filtered");
subplot(2,2,4);imshow(op3); title("Salt & Pepper noise image
filtered");
```

- Filter function

```
function [out_img] = nxnfilter(inp_img,k,type)
%NXNFILTER This function mean filters the image
%   The function takes in three inputs, image that has to be filtered
and
%   the size of the filter that has to be used. Type input determines
if
%   the function will filter using NXN filter or a combination of 1XN
and
%   NX1. Output of the function is the output image which is mean
filtered.

mid=round(k/2); %Computing mid point of filter matrix
K=(k-1)/2; %Computing iterating range for convolution
%Padding image
[rows,cols]=size(inp_img);
padded=[zeros(rows,3),inp_img,zeros(rows,3)];
padded=[zeros(3,cols+(K*2)); padded;zeros(3,cols+(K*2))];
padded_img=double(padded); %Converting the image to double
[rows,cols]=size(padded_img);
out_img=zeros(rows,cols);
% When convolution has to be using NXN filter
if(type==1)
    kernel=ones(k)./k^2; %Compute the kernel
    kernel=flip(kernel,2);
    kernel= flip(kernel,1);

    for i=mid:rows-K
        for j=mid:cols-K
            sum=0;
            for u=-K:K
                for v=-K:K

sum=sum+padded_img(i+u,j+v)*kernel((u+mid),(v+mid));
                end %Multiply and accumulate
            end
            out_img(i,j)=sum;   %Update the output image
        end
    end
    out_i=uint8(out_img); %Convert double to uint8
    out_img=out_i(mid:rows-K,mid:cols-K); %Cropping the padded image
end
% For seperable filter of 1XN and NX1
if(type==2)
    fc=ones(1,k)./k; %Computing column filter
    fr=ones(k,1)./k; %Computing row filter

    out1=zeros(rows,cols);
    out2=zeros(rows,cols);

    %Processing rows
    for r=1:rows
        for c=mid:cols-K
            sum=0;
            for dc=-K:K
```

```
                sum=sum+padded_img(r,c+dc)*fc(dc+mid);
            end
            out1(r,c)=sum; %Image after row processing
        end
    end
    %Processing columns
    for r=mid:rows-K
        for c=1:cols
            sum=0;
            for dr=-K:K
                sum=sum+out1(r+dr,c)*fr(dr+mid);
            end
            out2(r,c)=sum; %Image after columns processing
        end
    end

    out_img=uint8(out2);
    out_img=out_img(mid:rows-K,mid:cols-K); %Cropping the padded part
of the image
end
```

- Timing the filters

  Timing the filters using tic and toc functions

```
clc;
% taking input images
tic
% img_4=imread("P2_lake.png");
% img_5=imread("P2_lena_gray_512.png");
% img_6=imread("P2_peppers_gray.png");
img_7=imread("P2_walkbridge.png");

figure(1);
imshow(img_7); title("Color image");

%% Colour to gray
img_rgb=img_7;
[row,col,d]=size(img_rgb);
if(d==3)
    red_img=img_rgb(:,:,1);
    green_img=img_rgb(:,:,2);
    blue_img=img_rgb(:,:,3);
    img_gray=0.21*red_img+0.72*green_img+0.07*blue_img;
else
   img_gray=img_rgb;
end

figure(2);
imshow(img_gray); title("Gray scale image");
%%
inp_img=img_gray;
cropped=nxnfilter(inp_img,7,1);

figure (3);
imshow(cropped); title("Mean filtered image using 7X7 filter");
toc
```

Conclusion

The objective of the programming assignment was to perform histogram equalization and mean filtering on images. Mean filtering was performed using 7X7 filter and separable filters. Mean filtering was also performed on noisy images and results were observed.