The University of Windsor

ELEC4490: Sensors and Vision Systems

Summer 2020

Assignment # 4

Mask Operations

Friday, July 24, 2020

Emmanuel Mati

104418019

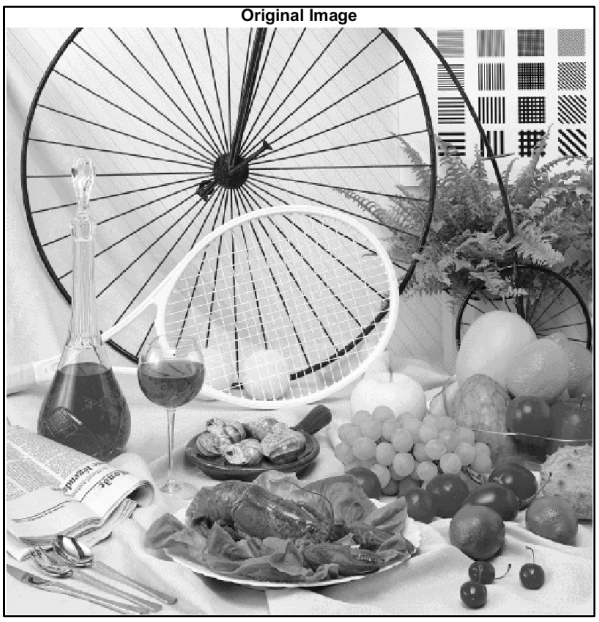
1)

CODE:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 4.1

%Clearing previous results
close all
clear all
clc

%Retreiving our image and displaying it
x = imread('picture.gif');
figure
imshow(x);
title('Original Image');

gcon = zeros(514, 514); %Our matrix that will hold our convoultion variables
gcor = zeros(514, 514); %Our matrix that will hold our  correlation variables

f = double(x); %Making our image a double so we can manipulate it

w = [0.02 0.02 0.02 %our mask User Can input values here
    .1 .1 1
    .1 0 0];

disp('user inputted mask:')
w

wrot = rot90(w,2); %180 degree rotated mask

for i = 2:513
    for j = 2:513
            gcon((i-1):(i+1),(j-1):(j+1)) = gcon((i-1):(i+1),(j-1):(j+1)) +
f(i-1, j-1)*w; %conducting our convolution
    end
end
%Convolution Results
%Displaying our Convoluted image
figure
imshow(uint8(gcon))
title('Convolution Results')

for i = 2:513
    for j = 2:513
            gcor((i-1):(i+1),(j-1):(j+1)) = gcor((i-1):(i+1),(j-1):(j+1)) +
f(i-1, j-1)*wrot; %conducting our correlation
    end
end
%Correlation Results
%Displaying our Correlation image
```

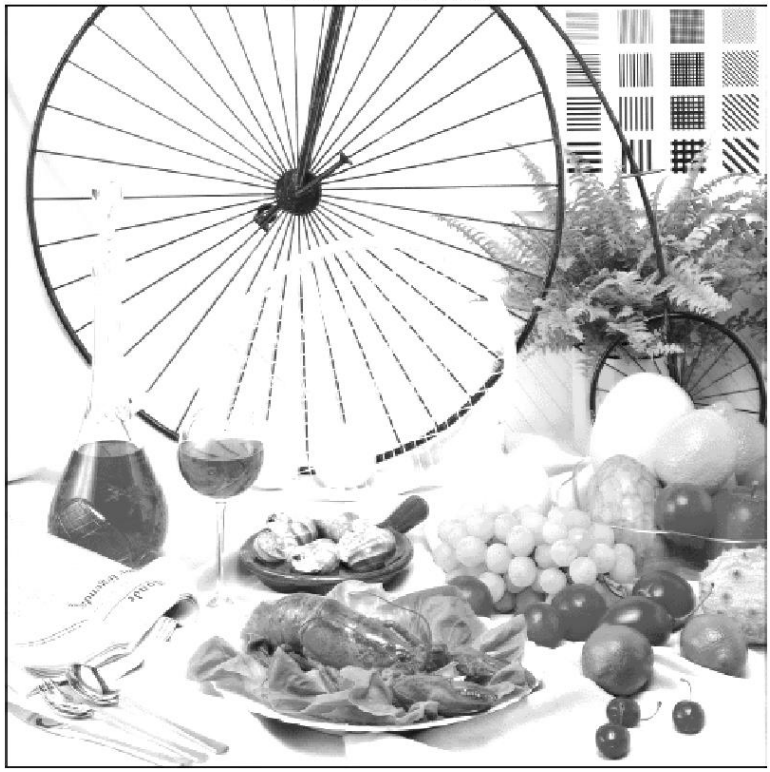
```
figure
imshow(uint8(gcor))
title('Correlation Results')

%%Testing using conv2
%using for convolution conv2
g = conv2(f,w);
%Displaying our convoluted image
figure
imshow(uint8(g))
title('Convolution using conv2')

%using for correlation conv2
g = conv2(f,wrot);
%Displaying our correlation image
figure
imshow(uint8(g))
title('Correlation using conv2')
```

Output results

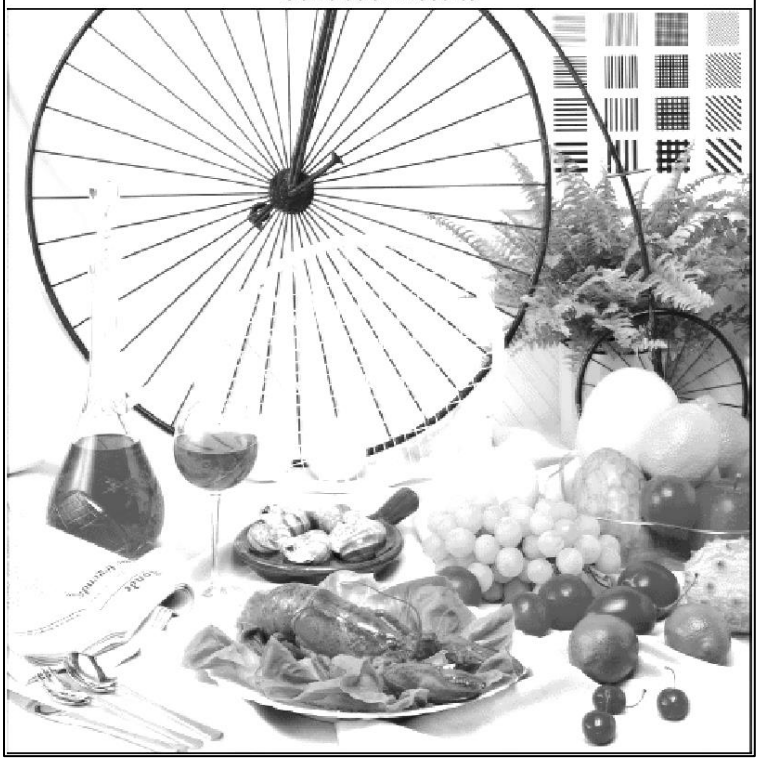


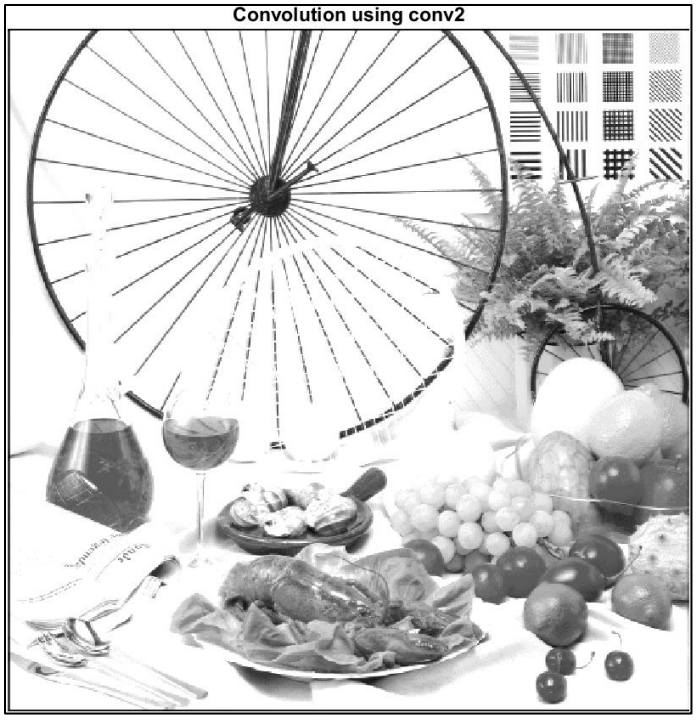

```
user inputted mask:

w =

    0.0200    0.0200    0.0200
    0.1000    0.1000    1.0000
    0.1000         0         0
```

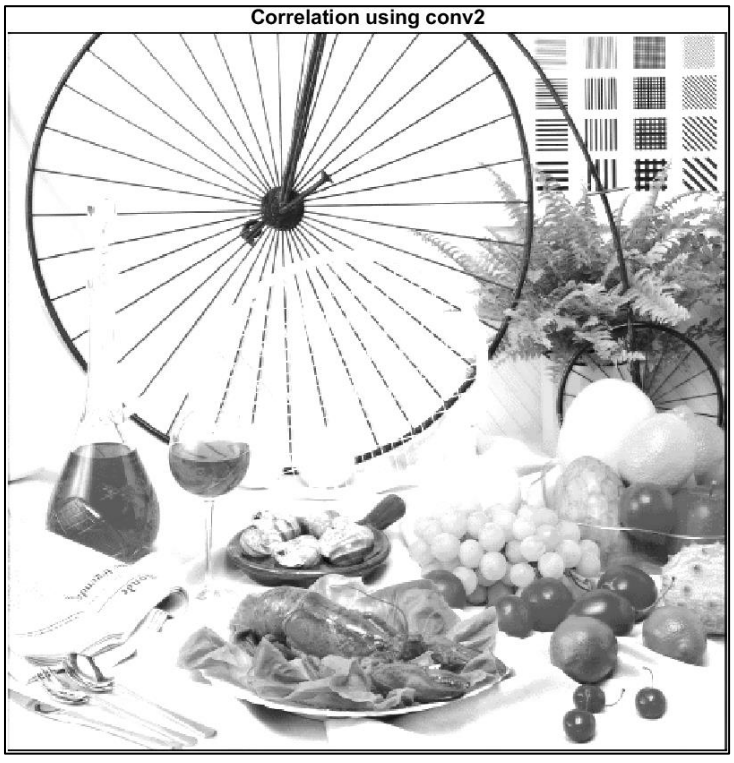
Convolution Results



Correlation Results







Results from my coded convolution and correlation photos are the exact same as the results from using the conv2 MATLAB function.

2)

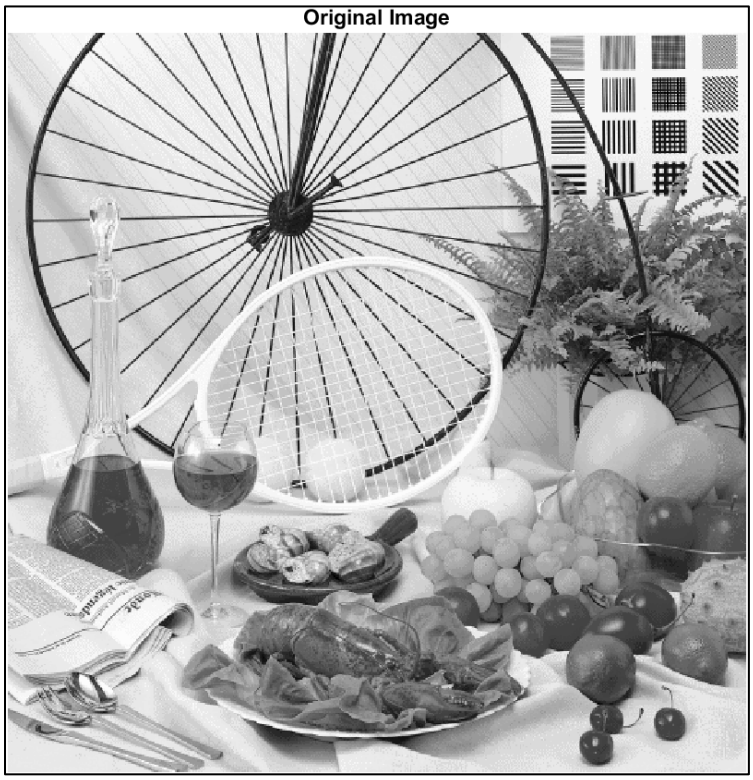
Code:

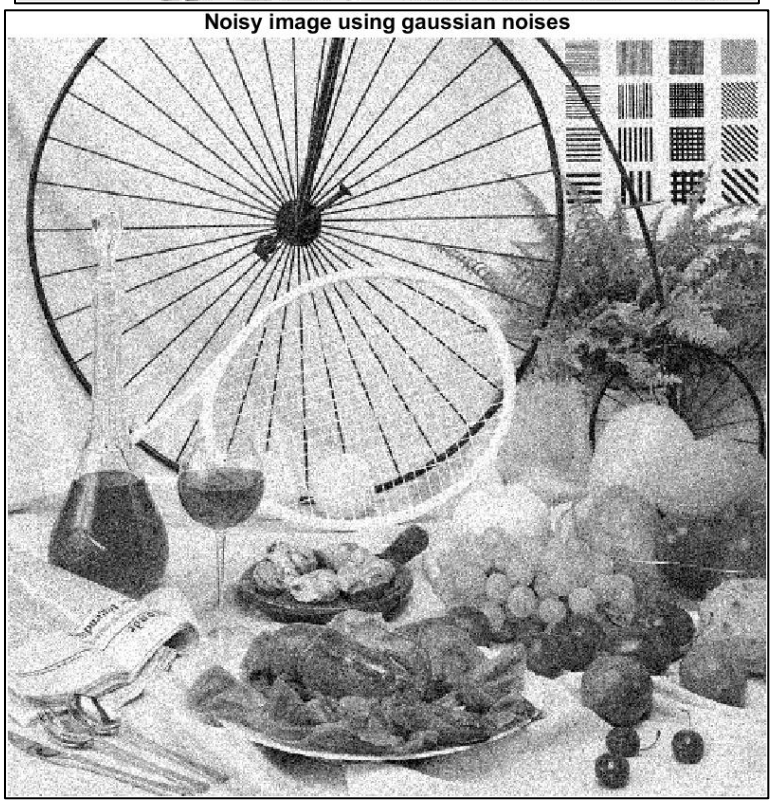
```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 4.2

%Clearing previous results
close all
clear all
clc

%%Part A

%Retreiving our image and displaying it
x = imread('picture.gif');
figure
imshow(x);
title('Original Image');

f = imnoise(x, 'gaussian');
f2 = imnoise(x, 'salt & pepper');

figure
imshow(f)
title('Noisy image using gaussian noises')
figure
imshow(f2)
title('Noisy image using salt and pepper noises')

%%Part B filtering

%Mean Filter
filter = fspecial('average', [3 3]);
filteredimage = imfilter(f, filter);
figure
imshow(filteredimage)
title('Gaussian noise filtered using fspecial-mean with a mask of [3 3]')

%Median Filter
filteredimage = medfilt2(f2,[3,3]);
figure
imshow(filteredimage)
title('Gaussian noise filtered using median with a mask of [3 3]')

%Gaussian Filter
filter = fspecial('gaussian', [3 3]);
filteredimage = imfilter(f, filter);
figure
imshow(filteredimage)
title('Gaussian noise filtered using fspecial-gaussian with a mask of [3 3]')

%Salt And Pepper
```

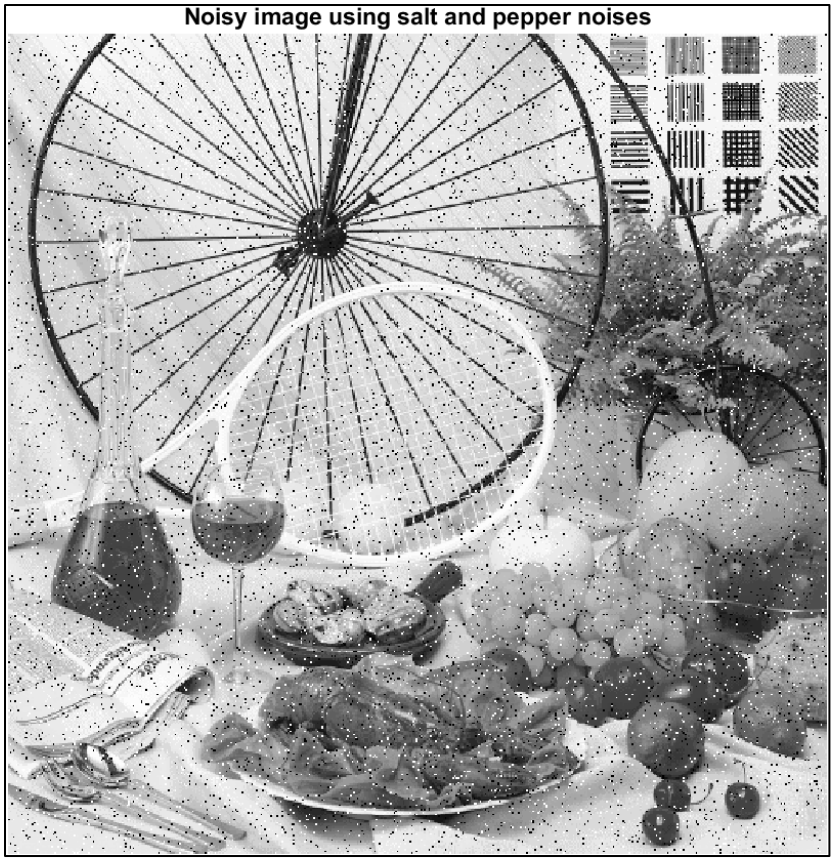
```
%Mean Filter
filter = fspecial('average', [3 3]);
filteredimage = imfilter(f2, filter);
figure
imshow(filteredimage)
title('Salt & Pepper noise filtered using fspecial-mean with a mask of [3
3]')

%Median Filter
filteredimage = medfilt2(f2,[3,3]);
figure
imshow(filteredimage)
title('Salt & Pepper noise filtered using median with a mask of [3 3]')

%Gaussian Filter
filter = fspecial('gaussian', [3 3]);
filteredimage = imfilter(f2, filter);
figure
imshow(filteredimage)
title('Salt & Pepper noise filtered using fspecial-gaussian with a mask of [3
3]')

%%Part C

%Creating our avearging filter for a mask of 3x3
avgmask1 = 1/4 *[ 0 1 0; 1 0 1; 0 1 0];
avgmask2 = 1/32 *[ 1 13 1; 3 16 3; 1 3 1];

filteredimage = imfilter(f, avgmask1); %Filtering a Gaussian image with
averaging mask
figure
imshow(filteredimage)
title('Part C, filtering an image using averaging 3x3 mask of: 1/4 *[ 0 1 0;
1 0 1; 0 1 0] ')

filteredimage = imfilter(f, avgmask2); %Filtering a Gaussian image with
averaging mask
figure
imshow(filteredimage)
title('Part C, filtering an image using averaging 3x3 mask of: 1/32 *[ 1 13
1; 3 16 3; 1 3 1]')

%% Part D


%Gaussian Filter 3x3
filter = fspecial('gaussian', [3 3]);
disp('Gaussian filter time for 3x3:');
tic
filteredimage1 = imfilter(f, filter);
toc

%Gaussian Filter 5x5
filter = fspecial('gaussian', [9 9]);
disp('Gaussian filter time for 9x9:');
```

```
tic
filteredimage2 = imfilter(f, filter);
toc

%Gaussian Filter 5x5
filter = fspecial('gaussian', [13 13]);
disp('Gaussian filter time for 13x13:');
tic
filteredimage3 = imfilter(f, filter);
toc
```

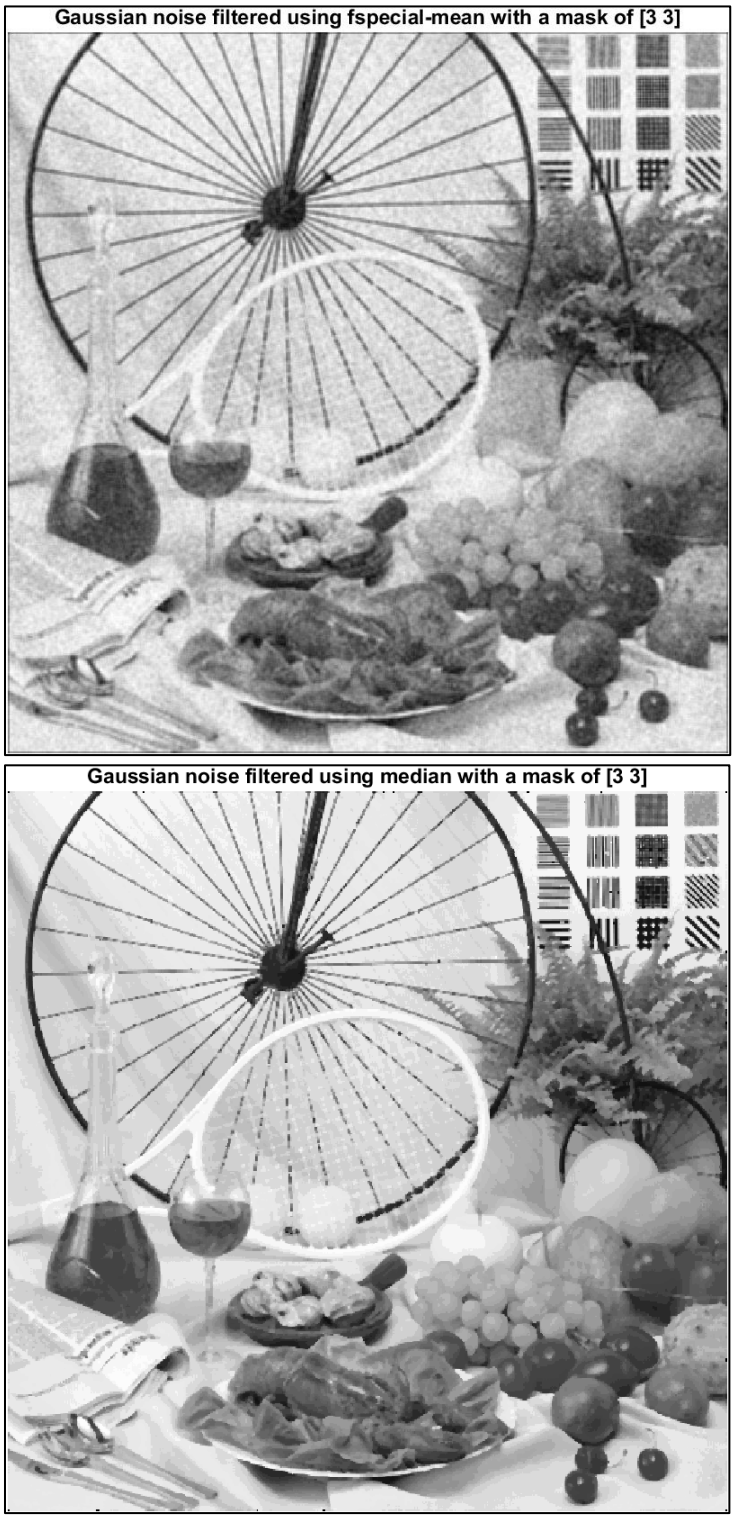
a) Part A output



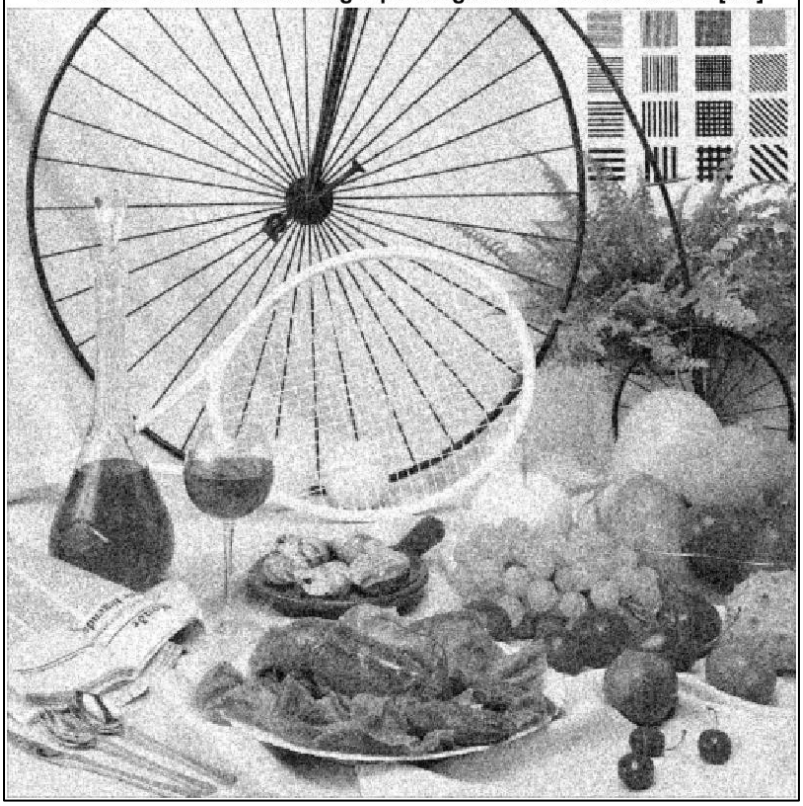



Noisy image using salt and pepper noises

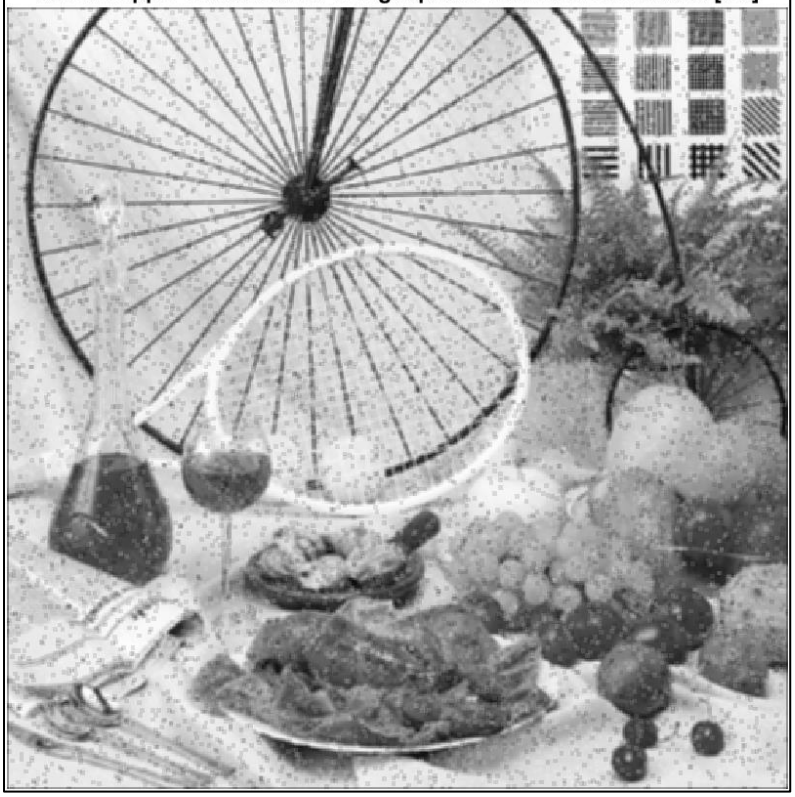
b)


Gaussian noise filtered using fspecial-mean with a mask of [3 3]
Gaussian noise filtered using median with a mask of [3 3]

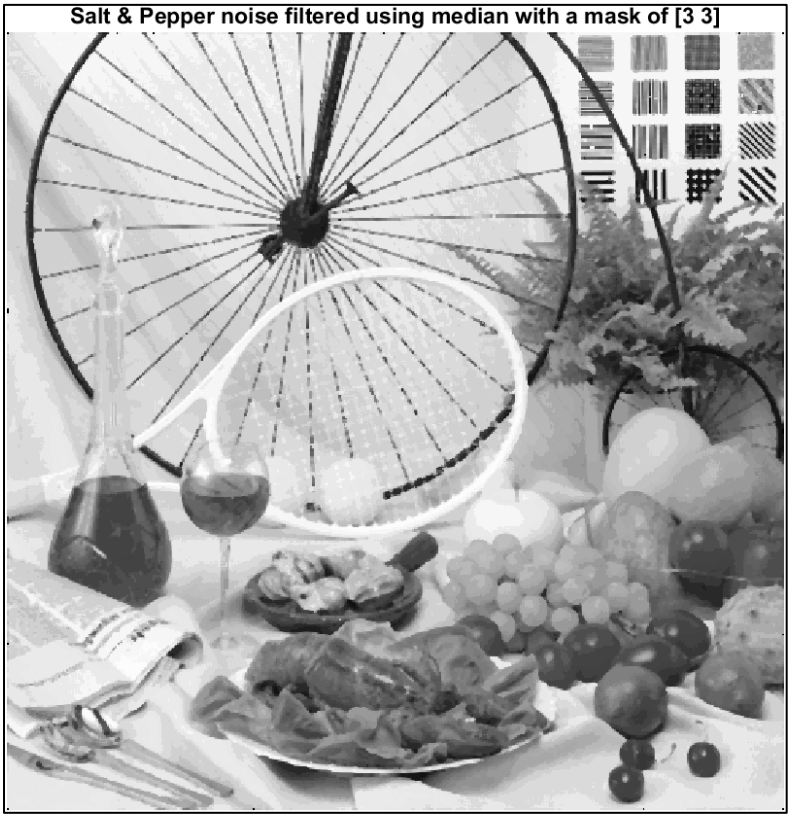
Gaussian noise filtered using fspecial-gaussian with a mask of [3 3]



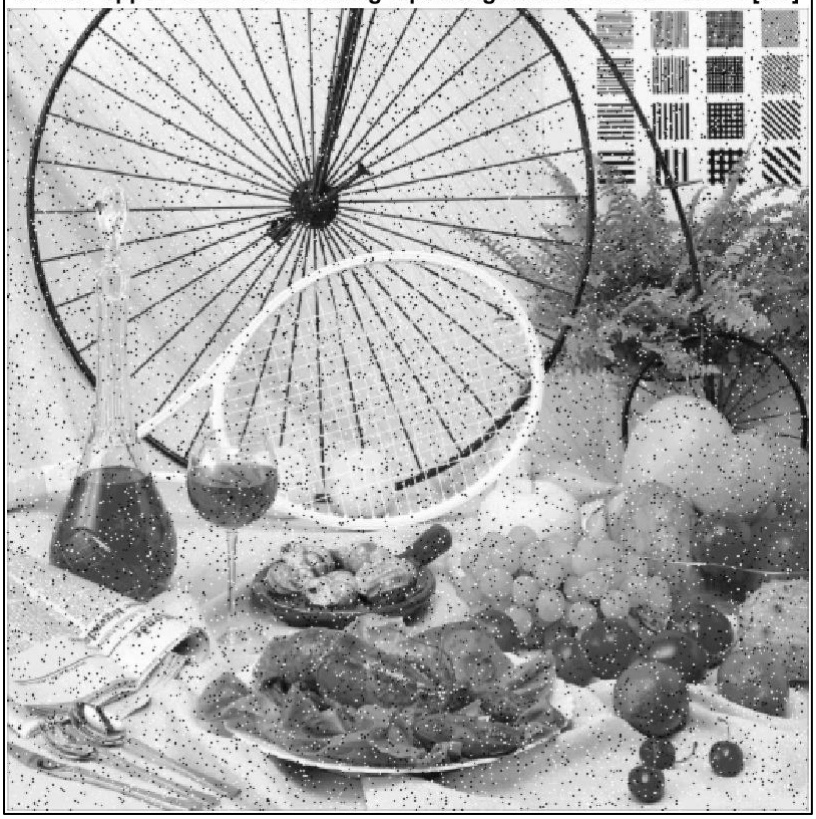
Salt & Pepper noise filtered using fspecial-mean with a mask of [3 3]



Salt & Pepper noise filtered using median with a mask of [3 3]

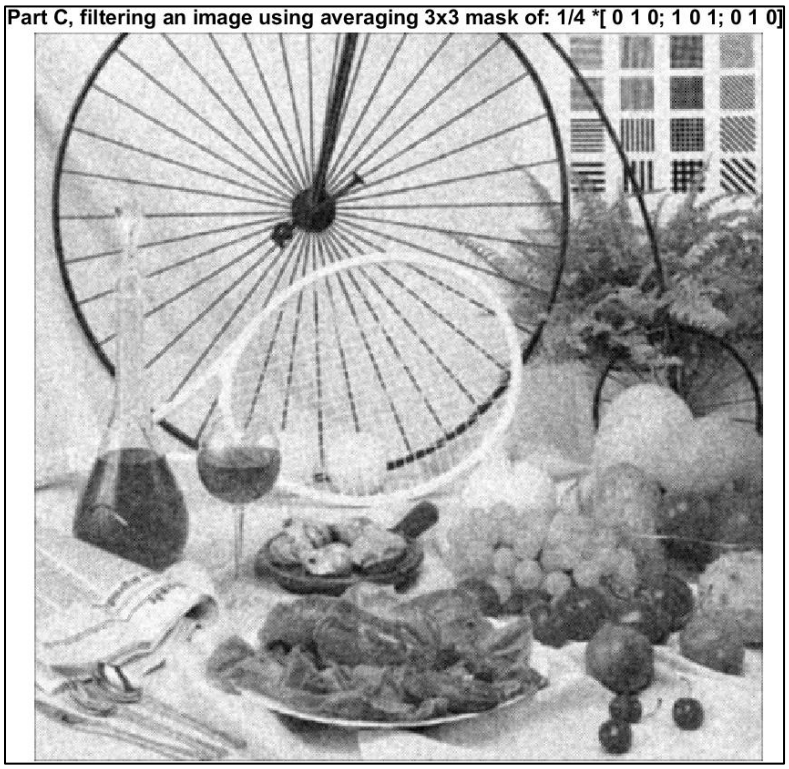


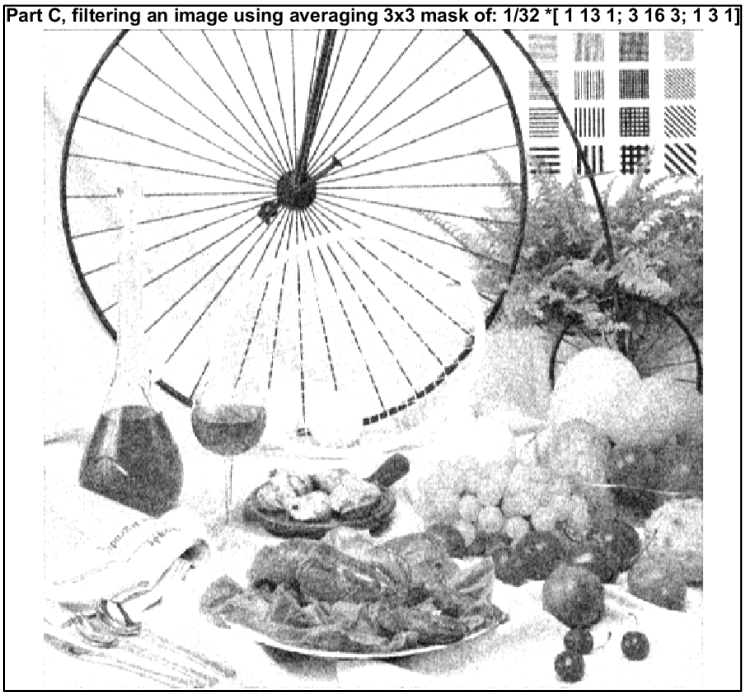
Salt & Pepper noise filtered using fspecial-gaussian with a mask of [3 3]



"Gaussian" noise was best reduced using imfilter's gaussian feature while "salt & pepper" noise was best reduced using medfilt2.

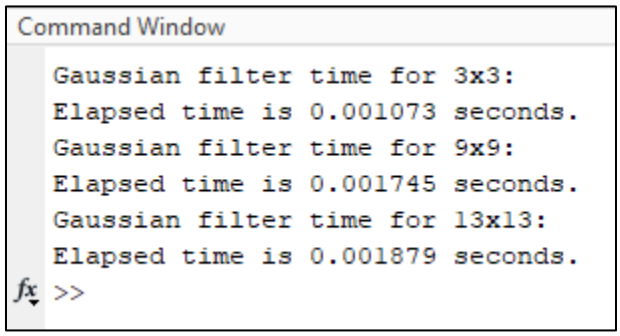
c) Averaging Masks results





After investigating the results, we can say that averaging masks with higher order of numbers inside of them tend to have lighter features after filtering which loses contrast detail.

d) Command Window Output for averaging times



As the filter size increases the computation time increases. The timing increases somewhat linearly. The reason this happens is because when the size of the matrix increases, there are more numbers to compute on the outer parts of the mask. This however does not seem to scale exactly linearly as shown with our output results. Hence, we can say as the mask size increases, the gaussian computation time scales somewhat linearly.

3)

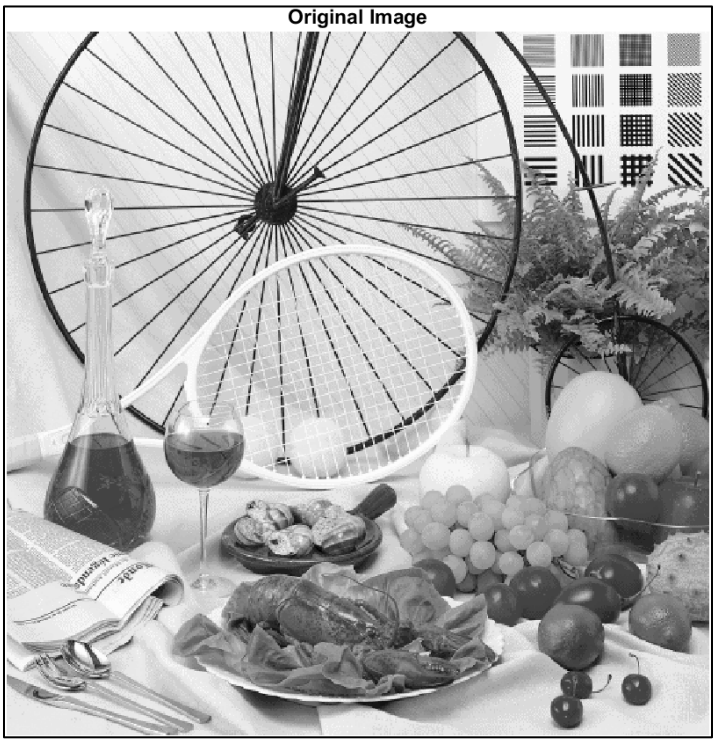
Code:

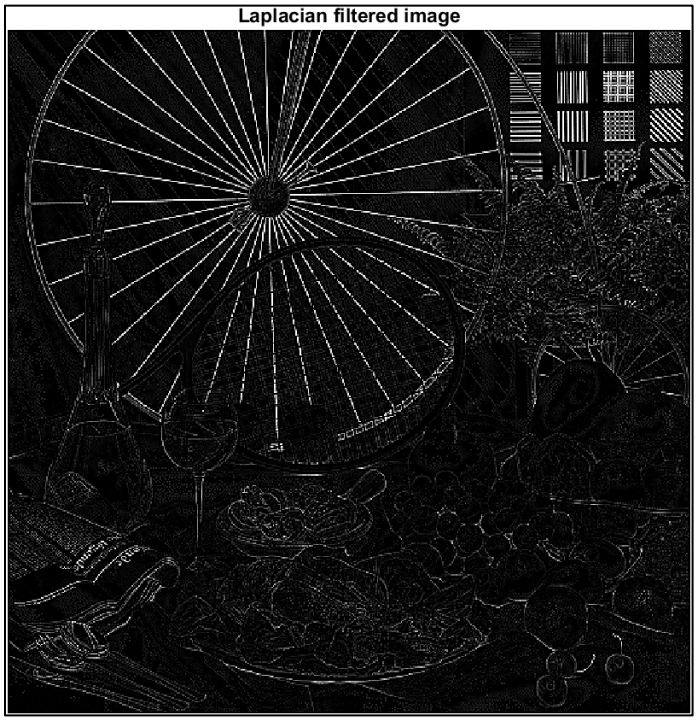
```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 4.3

%Clearing previous results
close all
clear all
clc

%%Part A

%Retreiving our image and displaying it
x = imread('picture.gif');
figure
imshow(x);
title('Original Image');

%Laplacian filter
filter = fspecial('laplacian');
lap = imfilter(x, filter);
figure
imshow(lap);
title('Laplacian filtered image');

%Prewitt filter
filter = fspecial('prewitt');
pre = imfilter(x, filter);
figure
imshow(pre);
title('Prewitt filtered image');

%Sobel filter
filter = fspecial('sobel');
sob = imfilter(x, filter);
figure
imshow(sob);
title('Sobel filtered image');

%Unsharp filter
filter = fspecial('unsharp');
unsharp = imfilter(x, filter);
figure
imshow(unsharp);
title('Unsharp filtered image');

%%Part B

%Adding our Gaussian Noise to x
x = imnoise(x, 'gaussian');
figure
```

```
imshow(x)
title('Our new noisy image before filtering')

%Laplacian filter
filter = fspecial('laplacian');
lap = imfilter(x, filter);
figure
imshow(lap);
title('Laplacian filtered image');

%Prewitt filter
filter = fspecial('prewitt');
pre = imfilter(x, filter);
figure
imshow(pre);
title('Prewitt filtered image');

%Sobel filter
filter = fspecial('sobel');
sob = imfilter(x, filter);
figure
imshow(sob);
title('Sobel filtered image');

%Unsharp filter
filter = fspecial('unsharp');
unsharp = imfilter(x, filter);
figure
imshow(unsharp);
title('Unsharp filtered image');

%%Part C

%edge detection using roberts and sobel

%Retreiving our rgb image and displaying it
y = imread('pennies.jpg');
figure
imshow(y);
title('Original RGB Image');
empty = zeros(size(y, 1), size(y, 2), 'uint8');


y1 = y(:, :, 1).*uint8(edge(y(:, :, 1), 'roberts')); %Our Roberts edge
y2 = y(:, :, 1).*uint8(edge(y(:, :, 1), 'sobel')); %Our Sobel edge
y3 = y(:, :, 2).*uint8(edge(y(:, :, 2), 'roberts')); %Our Roberts edge
y4 = y(:, :, 2).*uint8(edge(y(:, :, 2), 'sobel')); %Our Sobel edge
y5 = y(:, :, 1).*uint8(edge(y(:, :, 3), 'roberts')); %Our Roberts edge
y6 = y(:, :, 1).*uint8(edge(y(:, :, 3), 'sobel')); %Our Sobel edge

%displaying Roberts edge
figure
imshow(cat(3, y1, empty, empty))
title('Roberts Red edges for RGB image');
```

```
%displaying Sobels edge
figure
imshow(cat(3, y2, empty, empty))
title('Sobels Red edges for RGB image');

%displaying Roberts edge
figure
imshow(cat(3, empty, y3, empty))
title('Roberts Green edges for RGB image');

%displaying Sobels edge
figure
imshow(cat(3, empty, y4, empty))
title('Sobels Green edges for RGB image');

%displaying Roberts edge
figure
imshow(cat(3, empty, empty, y5))
title('Roberts Blue edges for RGB image');

%displaying Sobels edge
figure
imshow(cat(3, empty, empty, y6))
title('Sobels Blue edges for RGB image');

%displaying Roberts combined colour channel edges
figure
imshow(cat(3,y1,y3,y5))
title('Roberts edges combined');

%displaying Sobels edge
figure
imshow(cat(3,y2,y4,y6))
title('Sobels edges combined');
```

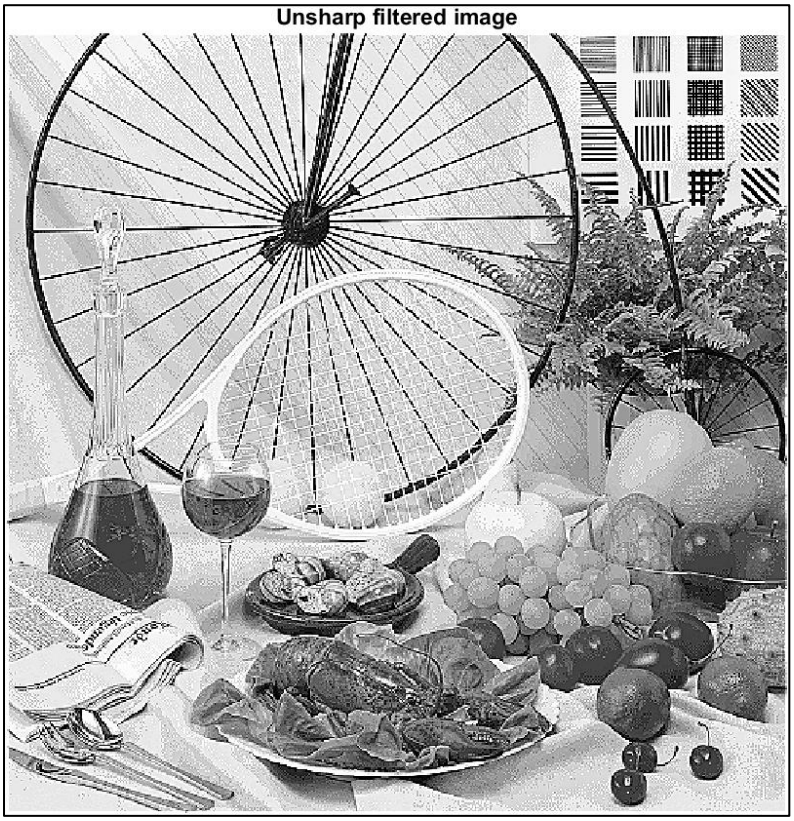
a) Our output images



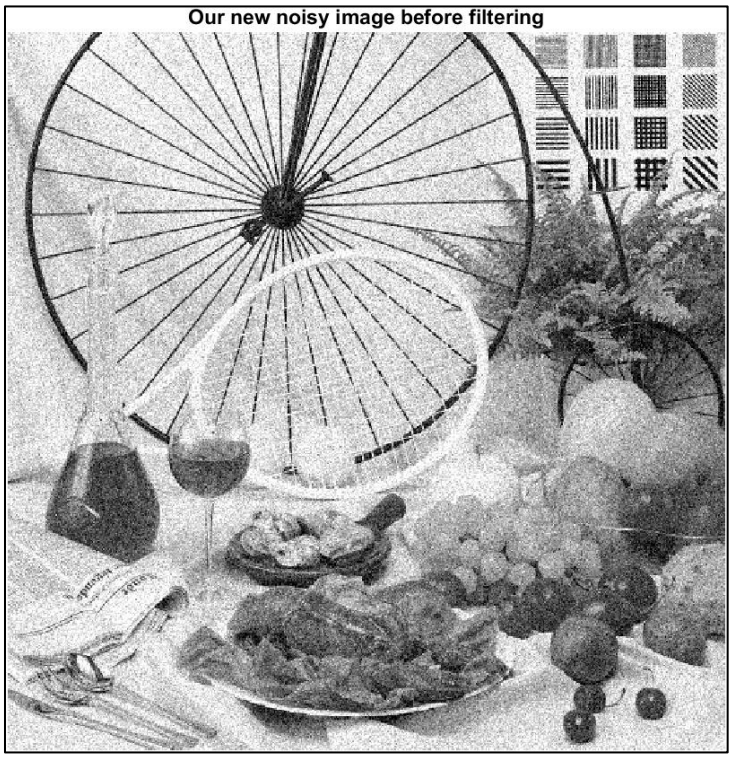


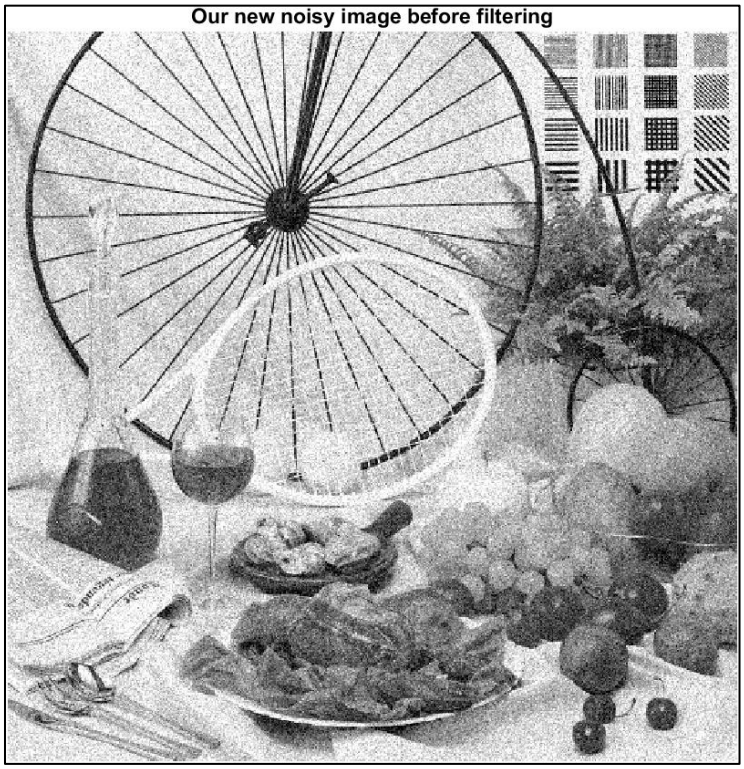
Prewitt filtered image

Unsharp filtered image

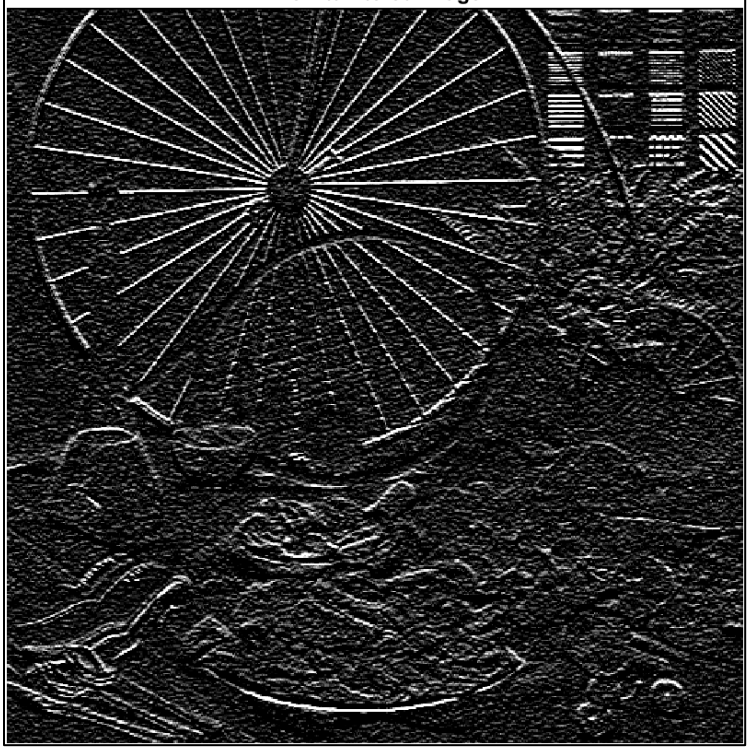


b) Repeating A with a noisy image

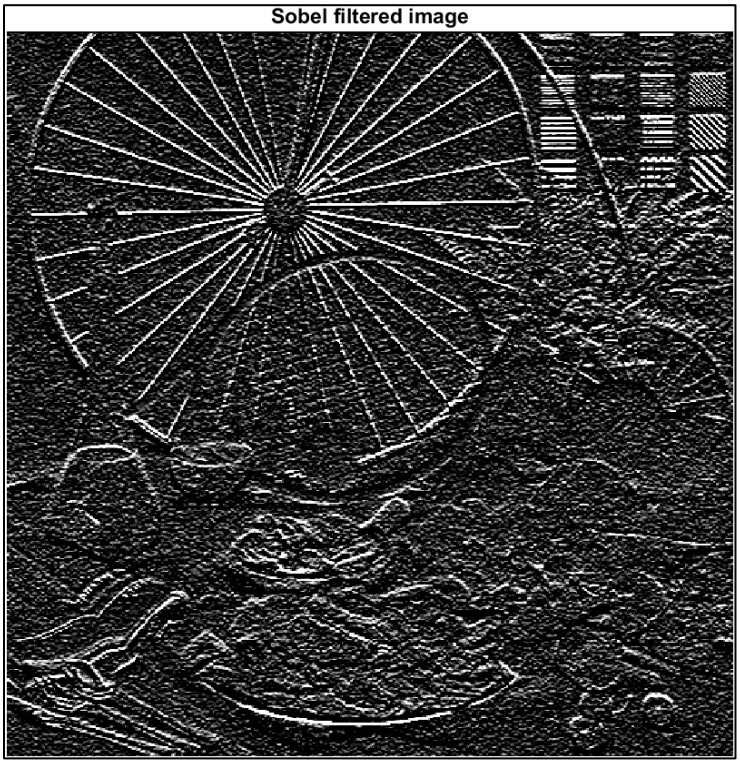




Prewitt filtered image



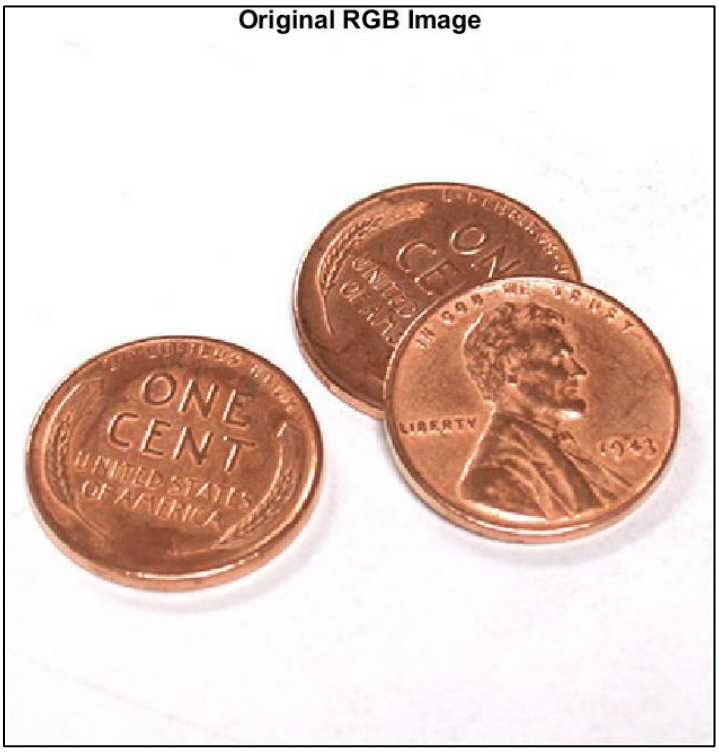
Sobel filtered image

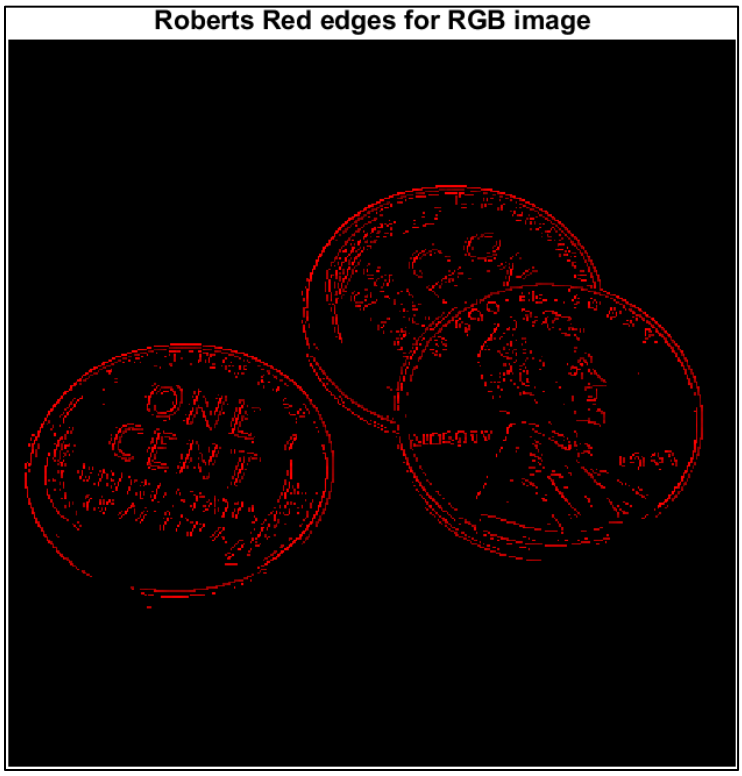


Sobel filtered image

After applying a Gaussian filter over our image and proceeding to filter it even more, we can say that that sharpness has dramatically decreased. There is also noticeably more noise in our filtered images.

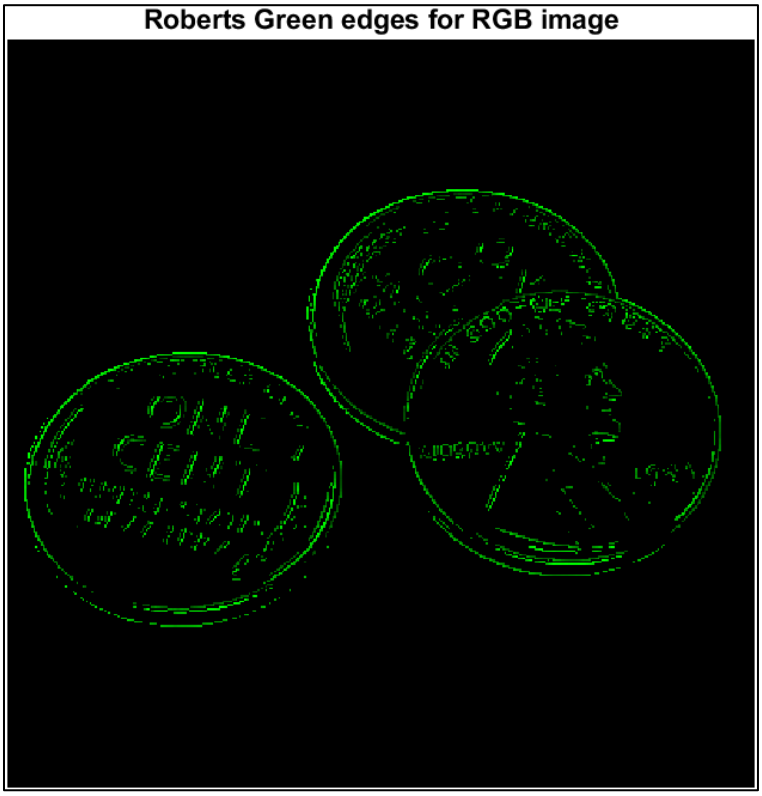
c) Output of our RGB image's edges for each colour pallet

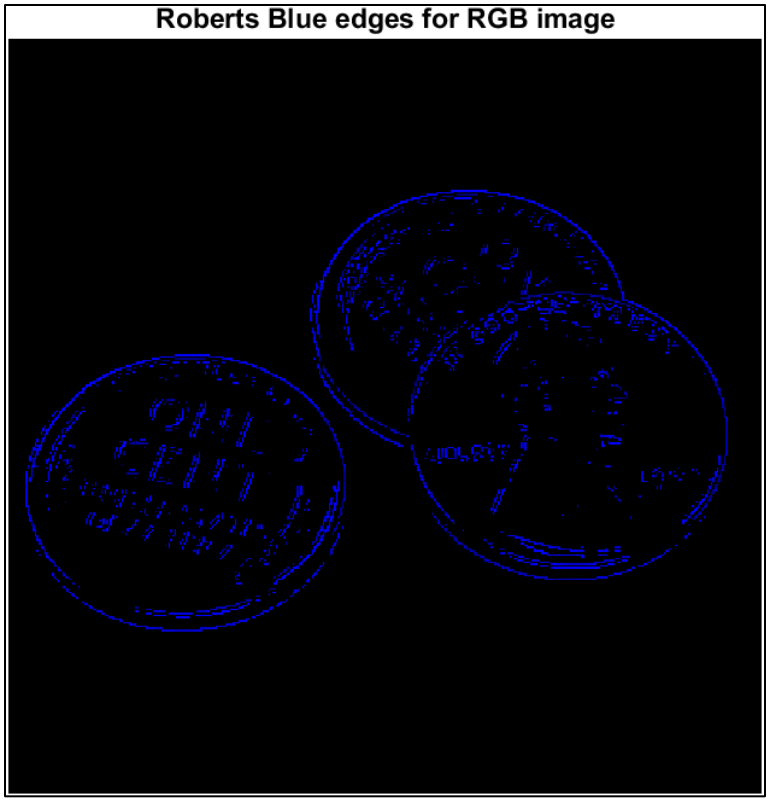


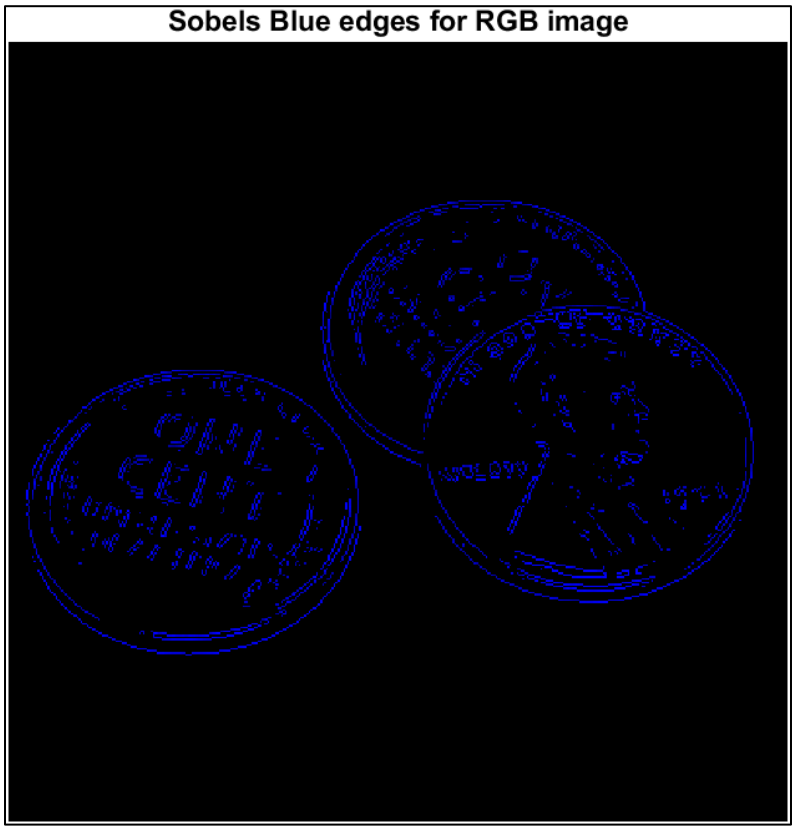


Sobels Red edges for RGB image

Roberts Green edges for RGB image

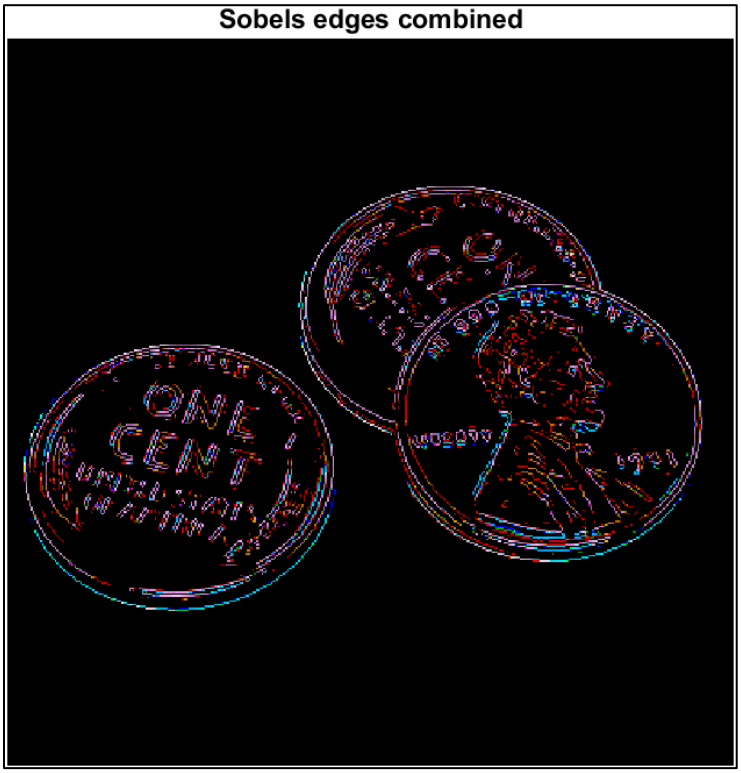



Roberts Blue edges for RGB image


Sobels Blue edges for RGB image

Roberts edges combined

Sobels edges combined

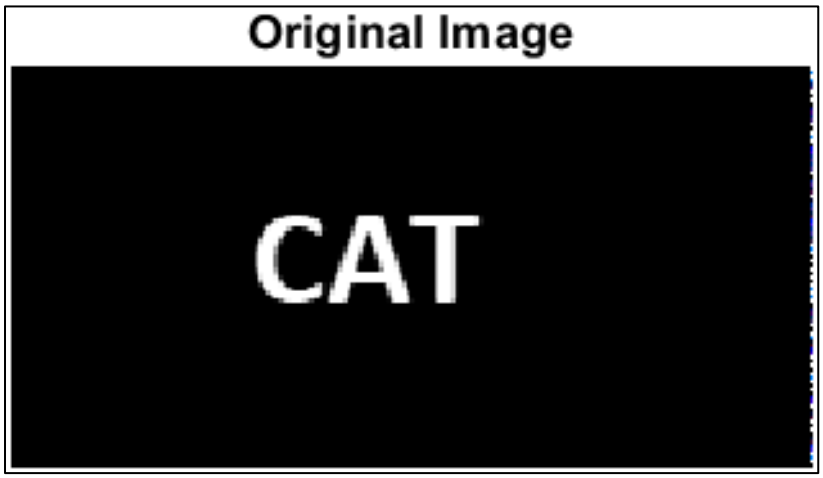


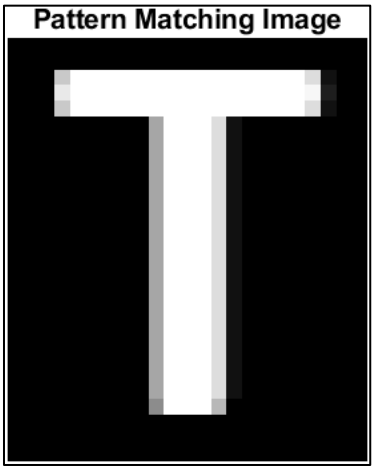
4a-b)

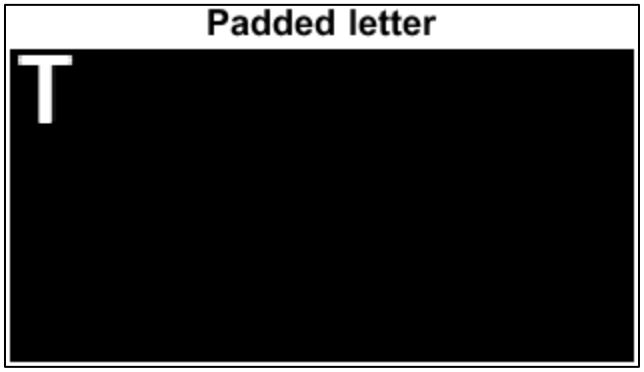
Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 4.4

%Clearing previous results
close all
clear all
clc

%%Part A

%Retreiving our image and displaying it
cat = rgb2gray(imread('cat.png'));
figure
imshow(cat);
title('Original Image');

%Retreiving character template
t = rgb2gray(imread('t.png'));
figure
imshow(t);
title('Pattern Matching Image');

[m n] = size(t);

%padding our letter
paddedt = uint8(zeros(100, 200));
paddedt(1:m, 1:n) = paddedt(1:m, 1:n) + t;
figure
imshow(paddedt);
title('Padded letter');

cor = normxcorr2(cat, paddedt);

%%Part B
disp('Template position in the image:')
[x y] =  find(cor ==(max(max(cor))))
```

Output Results







Command Window giving coordinates

```
Command Window
Template position in the image:

x =

    65


y =

   103
```

4c)

Code:

```
%Emmanuel Mati
%Summer 2020
%Sensors and Vision Systems
%Assignment 4.4 C
%Clearing previous results
close all
clear all
clc

%Retreiving our image and displaying it
cat = rgb2gray(imread('cat.png'));
figure
imshow(cat);
title('Original Image');

%Retreiving character template
t = rgb2gray(rot90(imread('t.png')));
figure
imshow(t);
title('Pattern Matching Image');

[m n] = size(t);

%padding our letter
paddedt = uint8(zeros(100, 200));
paddedt(1:m, 1:n) = paddedt(1:m, 1:n) + t;
figure
imshow(paddedt);
title('Padded letter');

cor = normxcorr2(cat, paddedt);

disp('Template position in the image:')
[x y] =  find(cor ==(max(max(cor))))
```

Output results when we rotate our template