EE 569 3D Vision: Post-Lab Report 2

Furkan Gül, Student ID: 28795

November 12, 2020

1 Experiments in Lab

This first part of the report is an introductory step to understand scripts, and functions implemented in Lab deeply.

Before diving into the first filtering method discussed in Lab, it will be better to mention the main script that calls all the required functions benefited in Lab. It is named "main", which can be found in the Appendix. Because there exist fundamentally four problems that will be addressed in Lab, the main script is divided into four sections by starting the code line with two comment characters (%%). Each section refers correspondingly to four distinct functions made for every four problems in Lab. At the beginning of all those four functions, there are a couple of lines of codes in common to make them generic. They all start with checking the input images if it is RGB or gray scale. If they are RGB, they will convert it into a grayscale format. To avoid arithmetic errors due to using image matrices of uint8 data type, which is the default option for image matrices in Matlab, those four functions will also convert data types of image matrices to double format. Finally, right before those four functions show returned images in the figure, the data type of the returned image's matrix form must be converted back into uint8. So that the returned image can be displayed without any issues.

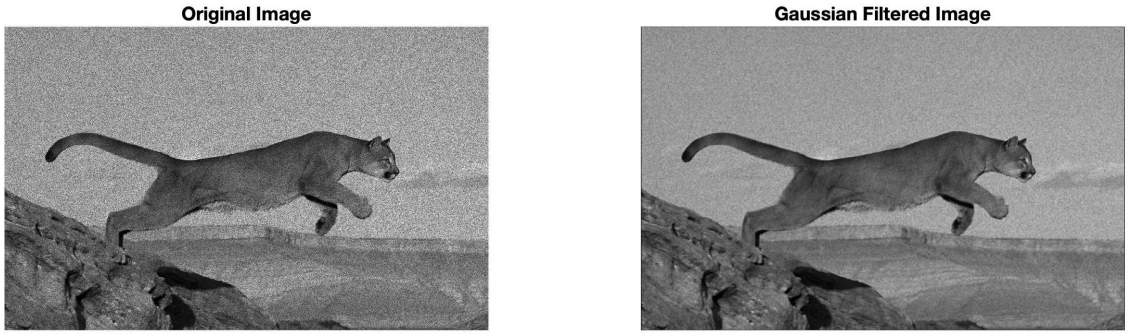


Figure 1: Gaussian Filtering of Image

The first section in the main script is dedicated to the Gaussian filtering for smoothing an image by removing "outliers" in pixel values of the image. These so-called "outlier pixels" can

be also considered to be noise in a given image. In this section, after getting the input image, "lab2gaussfilt" function is called by this input image and returns the Gaussian filtered output image. This function can be found in the Appendix. Gaussian filtering is a type of linear local operation over images by a convolution kernel. In contrast to the box filter which takes the mean of neighboring pixels, in the Gaussian filter, nearest neighboring pixels have the most influence. In this function, the 5 by 5 Gaussian filter is given. So that the input image will be convolved locally with this given Gaussian filter where the kernel size parameter, k, is 2. The kernel size parameter will decide the size of the window as the parameter name suggests. In this particular function, the window/kernel/filter will slide from the upper left corner of the input image to the right by one pixel until the upper right corner of the image. Then, the window slides back to the left edge of the image. But this time it will also slide one pixel above. It will maintain this process until the window comes to the right bottom corner of the image. At every sliding, the element-wise multiplication between the Gaussian filter and pixel values in the intersection of images will be calculated. Then, all the values in the result of this multiplication are summed together so that there would be one scalar pixel value for each sliding operation. Those scalar pixel values will be put into another new matrix. The image representation of this new matrix will be a Gaussian smoothed version of the input image. Finally, both the original input image and Gaussian filtered image with the window size as 2 will be plotted in Matlab figure, which can be seen in Figure 1.

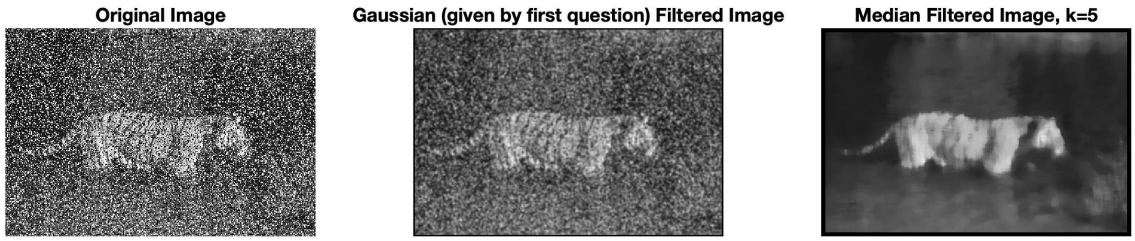


Figure 2: Median & Gaussian Filtered Images

In the second section of the main script, there is another method to deal with noise issues in images. That is called median filtering. It sorts pixel elements in a given local part of an image and picks the median pixel value among sorted pixel elements. That's why it is a type of nonlinear operation. Because of that nonlinearity, convolution operation can not be applied as it is in the Gaussian Filtering. The corresponding function name for that method is "lab2medfilt" which lies under the Appendix. It takes both the original input image and the window size parameter and returns the median filtered output image. In this specific function, local operations over the whole input image by sliding will be implemented without convolution. At every sliding, there will be a local part of the image with corresponding pixel values. The median function written in the first week of Lab will be called to calculate the median pixel element of a local image window at each sliding. Those scalar median pixel values will be put into another new matrix. The image representation of this new matrix will be the median filtered version of the input image. Finally, the input image, the Gaussian smoothed image by calling "lab2gaussfilt" function with the same input, and the median filtered image with the given window size parameter will be displayed in a Matlab figure together which can be seen in Figure 2.

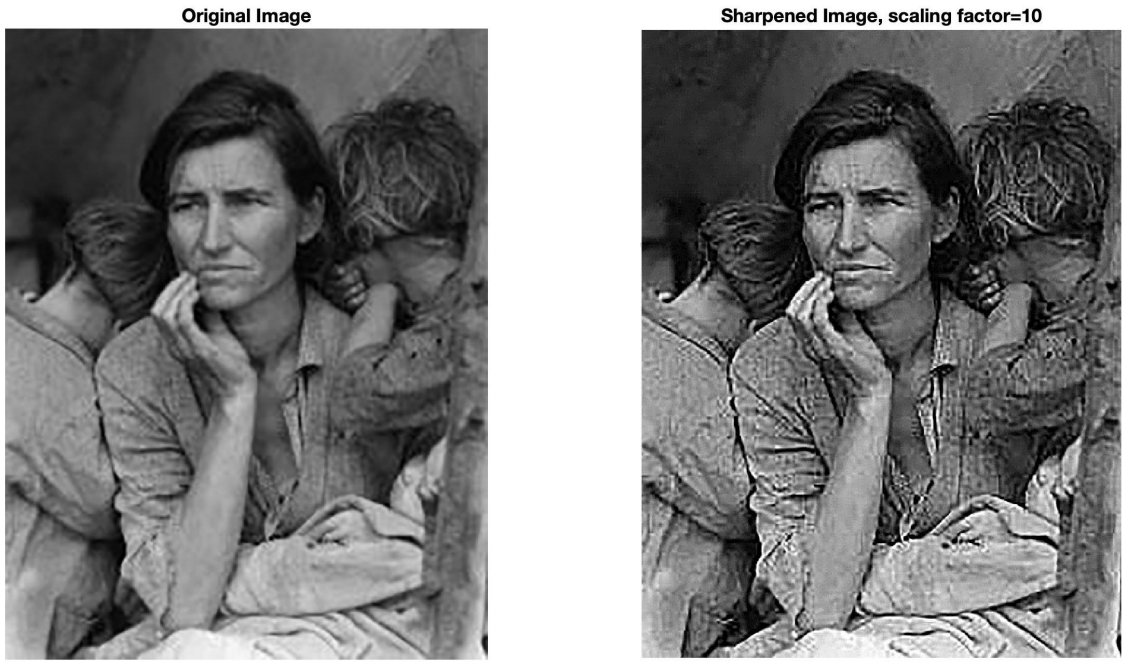


Figure 3: Sharpening Filtering of Image

The third part of the main script is written for image sharpening to generate an enhanced output image by increasing the contrast of input image along its edges. Sharpening method details can be found in the equation below:

$$J(p) = I(p) + \lambda[I(p) - S(p)] \tag{1}$$

where J denotes the sharpened output image, I is the given input image, S is the smoothed version of I image, and λ represents the scaling factor. This scaling factor controls the effect of the correction signal. The name of the function applies this method is "lab2sharpen", which can be seen in detail in the Appendix. This function takes an image, constant λ, and integer M as inputs and returns the sharpened output image. M is used to select the type of smoothing. For the box filtering, $M = 1$. For the Gaussian filtering, $M = 2$. Finally, for the median filtering, $M = 3$. By using the equation 1, the sharpened image array can be calculated easily. Finally, both the original input image and the sharpened image with the corresponding scaling, λ, parameter will be plotted in Matlab figure, which can be seen in Figure 3.

In the final fourth section of the main script, Sobel filtering will be addressed to calculate the first derivative of an image in x and y direction. Different 3 by 3 Sobel kernel for x and y are discussed in the class. Then, they are used in Lab for obtaining the first derivatives. "lab2sobelfilt" named function will be called with just the input image. It will return the first derivatives of the input image in x and y directions. This function can be found in the Appendix. In this function, the same sliding operation as it is in the "lab2gaussfilt" will be processed. But this time, instead of convolving the input image with the gaussian filter, the input image will be convolved separately with two Sobel derivative filters in each x and

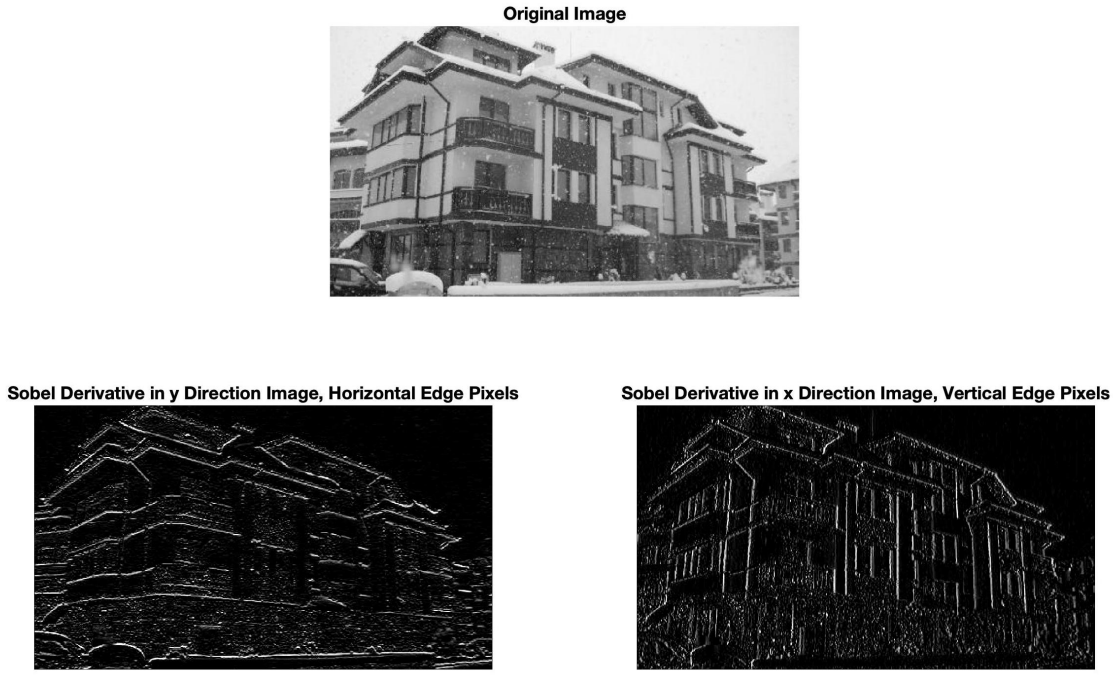


Figure 4: Sobel Filtered Image

y directions. Lastly, the images of two first derivative arrays in x and y direction will be displayed in the Matlab figure. This can be seen in Figure 4.

2 Results & Discussions

An image can be represented by a function which maps every location of pixels to the intensity at this given position. It is also called the image intensity function. Noise is just another function that is combined with the original function to create a new function. In other words, a noise image will be generated by adding noise function to the function of the given image. There exist many types of noise in images such as salt & pepper noise, impulse noise, and Gaussian noise. Salt & pepper noise is random occurrences of white and black pixels/spots in images. On the other hand, impulse noise is random occurrences of just white pixels/spots. In Gaussian noise, variations in intensity drawn from a Gaussian normal distribution.

To remove noise in images, there are several filters such as Gaussian filter, box filter, and median filter. It is crucial to understand in which case of noise to apply those filters. Gaussian filters are reasonable to apply if the noise is independent at each pixel in an image and centered about zero pixel value. In Gaussian and box filtering, pixel values in an image are replaced by taking local averaging of nearby pixel values. So, when the noise is not huge and tends to go to 0 intensity(in the case that the standard deviation of noise is small and the mean of noise is about 0), the noise can be averaged away and the right intensity value for pixels can be found by taking local average as it is in gaussian and local filtering.

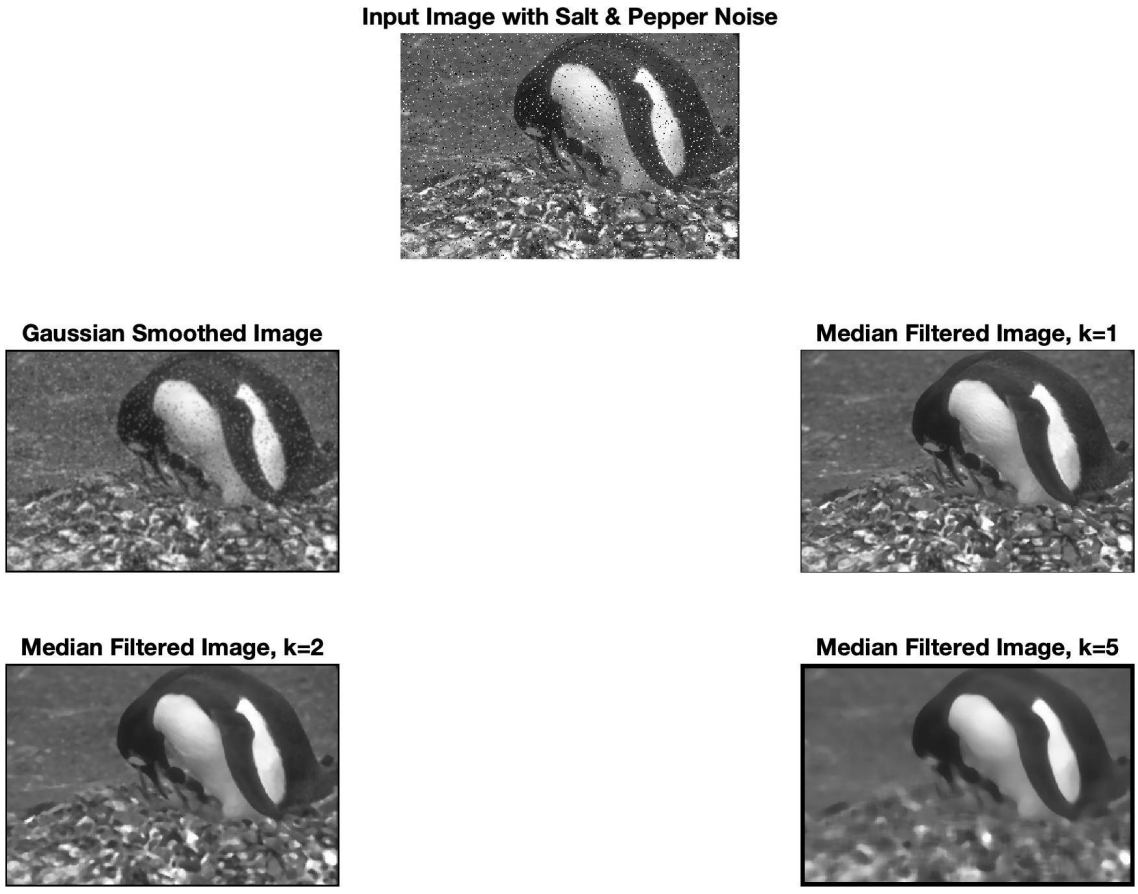


Figure 5: Gaussian and Median Filtering Comparison

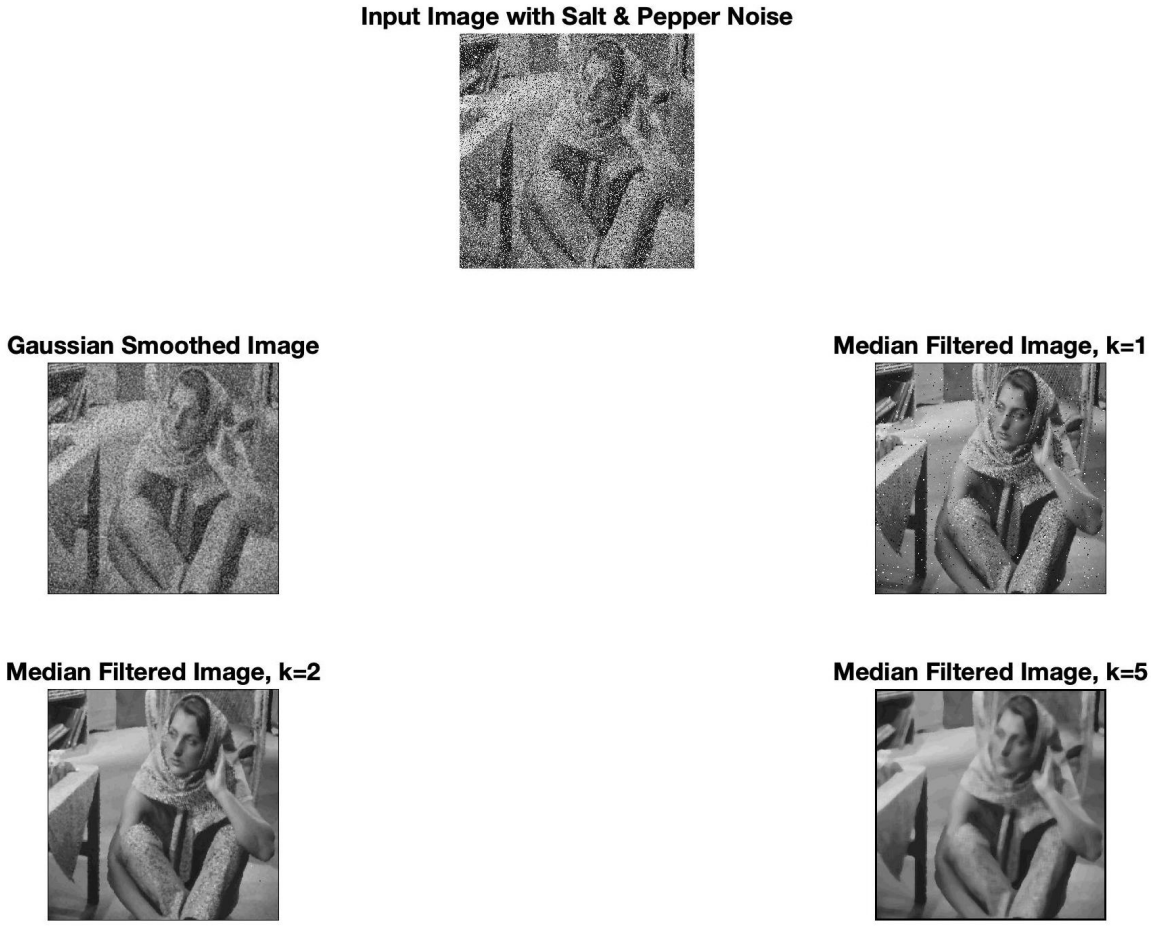


Figure 6: Gaussian and Median Filtering Comparison

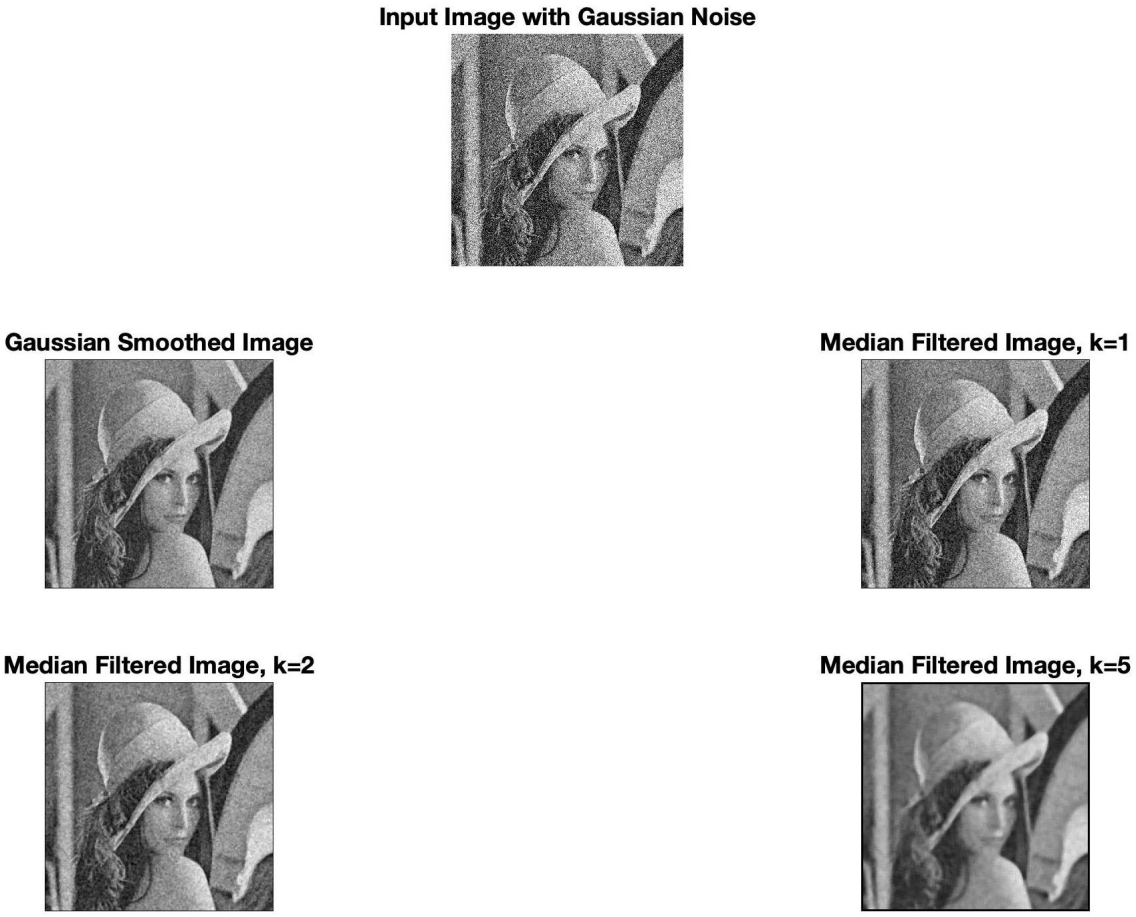


Figure 7: Gaussian and Median Filtering Comparison

Since salt & pepper noise is a totally random noise in an image, averaging will not work well. Instead of averaging, the median filter should be applied in such random noises to get the locally middle pixel value so that outlier noise pixel values can be removed simply. Even though the median filter is non-linear, it is perfect for removing huge random noises in an image. In Figures 5 and 6, input images have salt & pepper noise due to the random

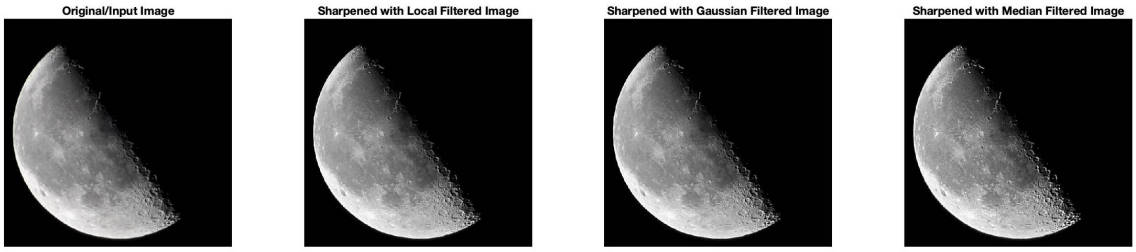


Figure 8: Sharpened Images with Scaling Factor of 1

occurrences of white and black pixels. The median filter is the best option for that type of huge random noise. The input image in Figure 6 has much more salt & pepper noise than the input image in Figure 5. That's why the filter size parameter, k, must be bigger for the image in figure 6 to clean up all those salt & pepper noises. In Figure 5, the best k value for removing those noise spikes would be 1, which corresponds to 3 by 3 sized median filter operation. However, in Figure 6, when k is 1, there is still a considerable amount of salt & pepper noises. That's why the best value for k would be 3 for the image in Figure 6. When k value goes beyond its optimum, even though edges in images will be preserved, the image will be blurred more. So, when k is large, small details in images can be lost. Due to the zero initialization of the filtered image array, there will be a black frame around it. As the filter size or equivalently k increased, the zero initialization effect can become more obvious to observe. There is a 5 by 5 gaussian filter given by Lab instructions. The effect of this gaussian filter on an input image can also be seen in Figures 5 and 6 as a Gaussian smoothed image. Compared to the median filtering, this Gaussian filter doesn't make any progress to clear the noise in both input images in Figures 5 and 6. In Figure 7, the input image has gaussian noise this time. The median filter is not doing a great job for this type of noise. However, it seems the Gaussian smoothed image and the median filtered with k equals 1 and 2 are quite similar. If there is a chance to change the sigma of this given 5 by 5 gaussian filter, gaussian filtering will be the obvious winner in terms of removing small zero centered noise.

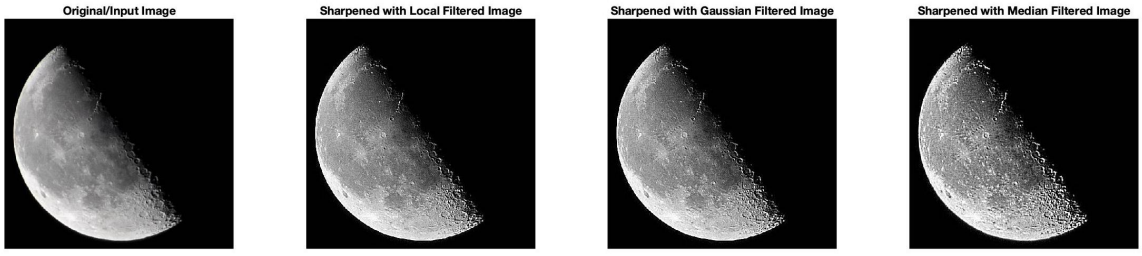


Figure 9: Sharpened Images with Scaling Factor of 5

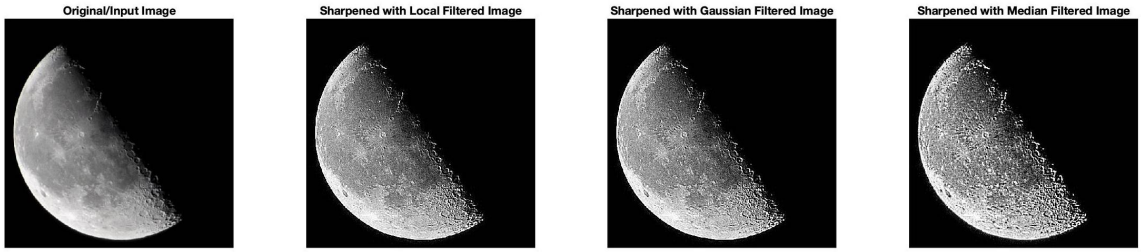


Figure 10: Sharpened Images with Scaling Factor of 10

In Figures 8, 9, and 10, the moon image will be sharpened using the scaling factor of 1, 5, 10 respectively. In each of those three Figures, there are three sharpened images with respect to three different types of the smoothed version of the input moon image; local filtering, gaussian filtering, and median filtering. In Figure 8, the sharpening effect is very small for all three different sharpened images due to the small scaling factor. In Figure 9, when the scaling factor increased to 5, the sharpening effect will become noticeable for all those three sharpened images. As the scaling factor increased to 10 in Figure 10, sharpening will be more noticeable than the sharpening in Figures 9 and 10. In Figures 9 and 10, compared to two sharpened images with local filtered and gaussian filtered versions of the moon image, the sharpened images with the median filtered version seem a bit exaggerated/magnified in terms of sharpening. The first takeaway is that when the scaling factor increased, the sharpening will be more visible. The second one is that when the median smoothed version is used for sharpening operation keeping the scaling factor the same, the sharpening effect would be magnified.

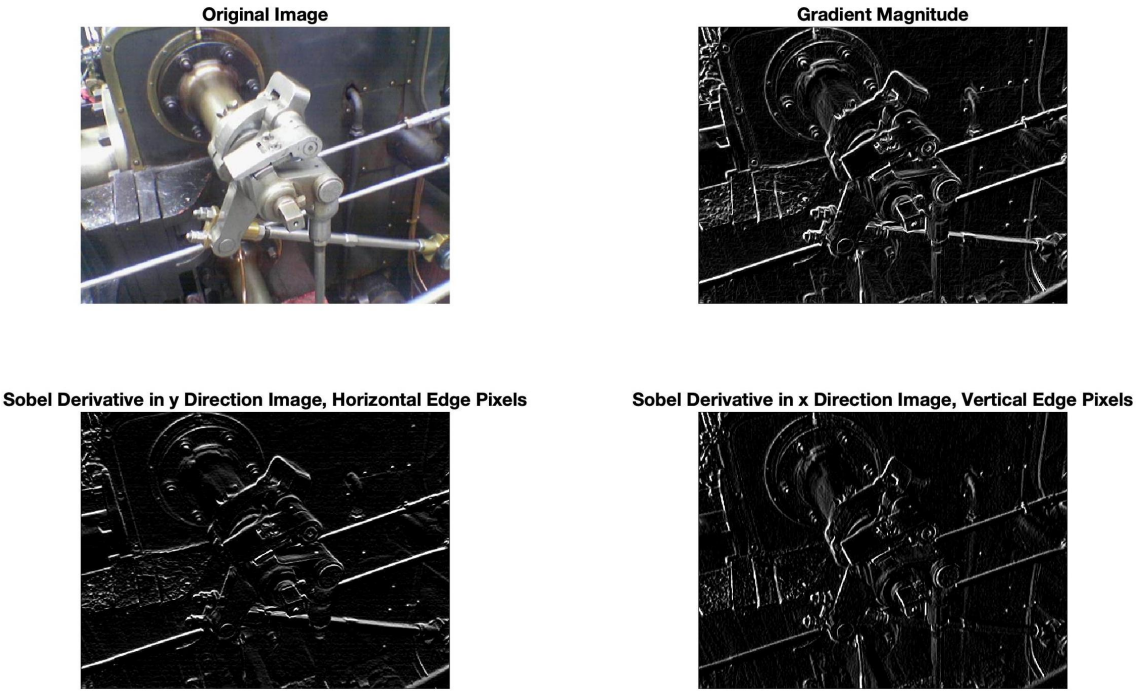


Figure 11: Sobel Edge Filtered Image

The input image in Figure 11 is the commonly used image for showing the result of the first derivative operation in x and y directions. In Lab, 3 by 3 Sobel filters are used to take the derivative of an image in x and y directions. To observe horizontal edges in an image, the input image must derivated with respect to y direction by using the corresponding Sobel filter. By looking at the bottom left image in Figure 11, it can be also observed that when the Sobel y derivative filter is applied, horizontal edges become more visible. To see vertical edges in an image, the first derivative of the input image in x direction must be calculated by using the Sobel x derivative filter. As can be seen in the bottom right image in Figure 11, the Sobel x derivative filter makes vertical edges brighter. Think of those two first derivatives in x and y directions as the elements of the gradient of the input image. When the magnitude of this gradient vector is calculated, the corresponding image for that magnitude is the upper right image in Figure 11. It shows both horizontal and vertical edges in the input image in a clear and magnified manner.

A Appendix

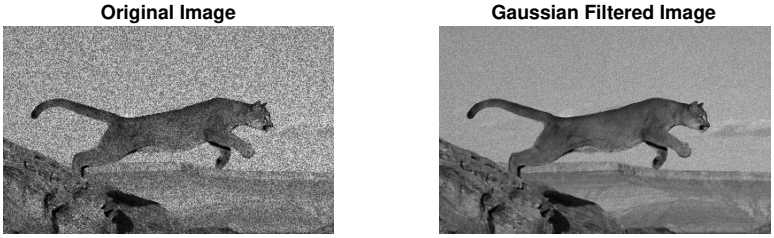
All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

Contents of Main Script

- Part 1: Linar Filtering: Gaussian Filtering
- Part 2: Non-Linear Filtering: Median Filtering
- Part 3: Sharpening
- Part 4: First Derivative, Sobel Filtering

Part 1: Linar Filtering: Gaussian Filtering

```
clc;clear;close;
% Let's get jump image
img = imread("jump.png");
% In order to remove orange warning, replace function output with ~ since
% we'll not use it.
[~] = lab2gaussfilt(img);
```

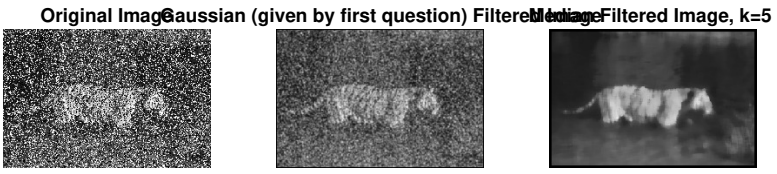


```
function [Filtered_img] = lab2gaussfilt(img)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = double(img); % In order to have proper matrix operations, convert datatype
% from uint8 to double

Gaussian_Matrix =  (1/273.)*[    1   4   7   4    1  ;
                                 4   16  26  16   4  ;
                                 7   26  41  26   7  ;
                                 4   16  26  16   4  ;
                                 1   4   7   4    1  ];


% Initialize matrices of the filtered image as 0s with the same dimension
% of the original matrix
Filtered_img = zeros(size(img));
k = 2; % since we have 5*5 filter

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.

%Method1
for i = k+1:h-k
    for j = k+1:w-k
        Window = img(i-k:i+k, j-k:j+k);
        % Calculate the gaussian weighted image pixels by multiplying
        % element-wise and adding all elements together to get single value
        value = sum(sum(Window.*Gaussian_Matrix));
        Filtered_img(i,j) = value;
    end
end

%Method2
% Filtered_img = conv2(img, Gaussian_Matrix, "same");


% Change the datatype of the original and gaussian filtered image matrix
```

```
% from double to uint8 so that it can be displayed without any problem.
img = uint8(img);
Filtered_img = uint8(Filtered_img);

% Plot the original, gaussian filtered images via subplot function
figure()

subplot(1,2,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(1,2,2);
imshow(Filtered_img);
title("Gaussian Filtered Image", 'FontSize',18);
end
```

Part 2: Non-Linear Filtering: Median Filtering

```
clc;clear;close;
% Let's get tiger image
img = imread("Tiger.png");
% Filter size parameter
k=5;
[resImg] = lab2medfilt(img, k);
```



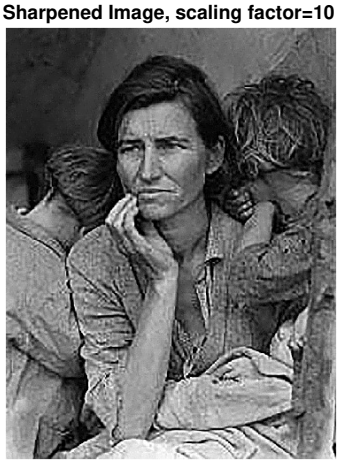

```
function median = myMedian(img)

[row,col,ch] = size(img);
Card = row*col; % Cardinality
    if (ch==3)
        img = rgb2gray(img);
    end
    img = double(img);
    Data = reshape(img, [1 Card]); % I reshaped my photo to 1D array
    Data = sort(Data);
    if (mod(Card,2) == 0)
        ind = Card/2;
        median = (Data(ind) + Data(ind+1))/2;
```

```
    else
        ind = (Card+1)/2;
        median = Data(ind);
    end
end

function [Filtered_img] = lab2medfilt(img, k)
% Chech the original image if it is RGB or monocolor
[row,col,ch] = size(img);
% If the image is RGB (colored), then convert it to grayscale image
if ch == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = double(img); % In order to have proper matrix operations, convert datatype
% from uint8 to double

% Call lab2gaussfilt function to get the gaussian filtered image
GaussianFiltered = lab2gaussfilt(img);


% Initialize matrices of the filtered image as 0s with the same dimension of
% the original matrix
Filtered_img = zeros(size(img));

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.

for i = k+1:row-k
    for j = k+1:col-k
        Window = img(i-k:i+k, j-k:j+k);
        % Calculate the median of pixel values of the substracted part of image
        % by calling earlier function myMedian
        value = myMedian(Window);
        Filtered_img(i,j) = value;
    end
end


% Change the datatype of the original image and filtered image from double to
% uint8 so that it can be displayed without any problem.
img = uint8(img);
Filtered_img = uint8(Filtered_img);
```

```
% Plot the original, gaussian filtered, and median filtered images via
% subplot function
figure()

subplot(1,3,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(1,3,2);
imshow(GaussianFiltered);
title("Gaussian (given by first question) Filtered Image", 'FontSize',18);

subplot(1,3,3);
imshow(Filtered_img);
title("Median Filtered Image, k="+int2str(k), 'FontSize',18);
end
```

Part 3: Sharpening

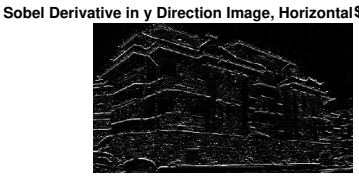
```
clc;clear;close;
% Let's get mother image
img = imread("mother.png");
% Scaling factor that controls the influence of the correction signal
limbda = 10;
% For selecting the method of smoothing. 1 for the 3 by 3 local filtering,
% 2 for the gaussian (given in the first part) filtering,
% 3 for the 7 by 7 median filtering
Mode = 2;
[Simg] = lab2sharpen(img, limbda, Mode);
```



```
function [Sharpened_img] = lab2sharpen(img, limbda, M)
% Chech the original image if it is RGB or monocolor
[~,~,ch] = size(img);
% If the image is RGB (colored), then convert it to grayscale image
if ch == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

% Let's get smoothed image according to the mode provided from their
% corresponding functions
if M == 1
    Smoothed = lab1locbox(img, 1);
elseif M == 2
    Smoothed = lab2gaussfilt(img);
elseif M == 3
    Smoothed = lab2medfilt(img, 3);
end

img = double(img);
Smoothed = double(Smoothed); % In order to have proper matrix operations,
% convert datatype from uint8 to double

% Sharpening function
Sharpened_img = img + limbda*(img-Smoothed);

% Change the datatype of the original image and sharpened image from double
% to uint8 so that it can be displayed without any problem.
img = uint8(img);
Sharpened_img = uint8(Sharpened_img);

% Plot the original, the original and sharpened images via subplot function
figure("NumberTitle", 'off', "Name", 'Sharpening')

subplot(1,2,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(1,2,2);
imshow(Sharpened_img);
title("Sharpened Image, scaling factor="+int2str(limbda), 'FontSize',18);
end
```

Part 4: First Derivative, Sobel Filtering

```
clc;clear;close;
% Let's get house image
img = imread("house.png");
[x_filtered, y_filtered] = lab2sobelfilt(img);
```




```
function [x_filtered, y_filtered] = lab2sobelfilt(img)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = double(img); % In order to have proper matrix operations, convert
% datatype from uint8 to double

% Define sobel filters for x and y
x_Filter = [-1 0 1;-2 0 2; -1 0 1];
y_Filter = [-1 -2 -1; 0 0 0 ; 1 2 1];

k = 1; % Since we have 3 by 3 sobel filters

% Initialize X and Y sobel derivatives as 0s with the same dimension of
% the original matrix
x_filtered =  zeros(size(img));
y_filtered =  zeros(size(img));

% Go over center pixels. We have total 2k+1 parameters in every row & column of
```

```
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.
for i = k+1:h-k
    for j = k+1:w-k
        Window = img(i-k:i+k, j-k:j+k);
        % Calculate the first x and y derivative pixel values of the substracted
        % part of image by multiplying each filter element-wise with window
        Xvalue = sum(sum(Window.*x_Filter));
        Yvalue = sum(sum(Window.*y_Filter));
        x_filtered(i,j) = Xvalue;
        y_filtered(i,j) = Yvalue;
    end
end

% Change the datatype of the images from double to uint8
% so that it can be displayed without any problem.
img = uint8(img);
x_filtered = uint8(x_filtered);
y_filtered = uint8(y_filtered);

% Plot the original, sovel derivative images via subplot function
figure()

subplot(2,2,[1,2]);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(2,2,3);
imshow(y_filtered);
title("Sobel Derivative in y Direction Image, Horizontal Edge Pixels", 'FontSize',18);

subplot(2,2,4);
imshow(x_filtered);
title("Sobel Derivative in x Direction Image, Vertical Edge Pixels", 'FontSize',18);
end
```

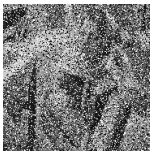
Contents of Examples4Report2 Script

- For Comparison between Gaussian Filtering and Median Filtering
- For Examining Sharpening Filter
- Sobel Filtering Closer Looking

For Comparison between Gaussian Filtering and Median Filtering

```
clc;clear;close;
% Let's get noisyimg image
%img = imread("gaussianNoise.png");
img = imread("noisyimg.png");
set(0,'DefaultFigureVisible','off')
[resImg1] = lab2medfilt(img, 1);
[resImg2] = lab2medfilt(img, 2);
[resImg5] = lab2medfilt(img, 5);
gaussImg = lab2gaussfilt(img);
set(0,'DefaultFigureVisible','on')
figure
subplot(3,2,[1,2])
imshow(img)
%title("Input Image with Gaussian Noise", 'FontSize',18)
title("Input Image with Salt & Pepper Noise", 'FontSize',18)
subplot(3,2,3)
imshow(gaussImg)
title("Gaussian Smoothed Image", 'FontSize',18)
subplot(3,2,4)
imshow(resImg1)
title("Median Filtered Image, k=1", 'FontSize',18)
subplot(3,2,5)
imshow(resImg2)
title("Median Filtered Image, k=2", 'FontSize',18)
subplot(3,2,6)
imshow(resImg5)
title("Median Filtered Image, k=5", 'FontSize',18)
```

Input Image with Salt & Pepper Noise



Gaussian Smoothed Image

Median Filtered Image, k=1

Median Filtered Image, k=2

Median Filtered Image, k=5

For Examining Sharpening Filter

```
clc;clear;close;
% Let's get mother image
img = imread("moon.jpg");
% Scaling factor that controls the influence of the correction signal
limbda = 1;
% For selecting the method of smoothing. 1 for the 3 by 3 local filtering,
% 2 for the gaussian (given in the first part) filtering,
% 3 for the 7 by 7 median filtering
Mode = 1;
set(0,'DefaultFigureVisible','off')
[sharp1_1] = lab2sharpen(img, limbda, Mode);
[sharp1_2] = lab2sharpen(img, limbda, 2);
[sharp1_3] = lab2sharpen(img, limbda, 3);
limbda = 5;
[sharp2_1] = lab2sharpen(img, limbda, Mode);
[sharp2_2] = lab2sharpen(img, limbda, 2);
[sharp2_3] = lab2sharpen(img, limbda, 3);
limbda = 10;
[sharp3_1] = lab2sharpen(img, limbda, Mode);
[sharp3_2] = lab2sharpen(img, limbda, 2);
[sharp3_3] = lab2sharpen(img, limbda, 3);
set(0,'DefaultFigureVisible','on')
% figure
% subplot(1,4,1)
% imshow(img)
% title("Original/Input Image")
% subplot(1,4,2)
% imshow(sharp1_1)
% title("Sharpened with Local Filtered Image")
% subplot(1,4,3)
% imshow(sharp1_2)
% title("Sharpened with Gaussian Filtered Image")
% subplot(1,4,4)
% imshow(sharp1_3)
% title("Sharpened with Median Filtered Image")
% figure
% subplot(1,4,1)
% imshow(img)
% title("Original/Input Image")
% subplot(1,4,2)
% imshow(sharp2_1)
% title("Sharpened with Local Filtered Image")
% subplot(1,4,3)
```

```
% imshow(sharp2_2)
% title("Sharpened with Gaussian Filtered Image")
% subplot(1,4,4)
% imshow(sharp2_3)
% title("Sharpened with Median Filtered Image")
figure
subplot(1,4,1)
imshow(img)
title("Original/Input Image")
subplot(1,4,2)
imshow(sharp3_1)
title("Sharpened with Local Filtered Image")
subplot(1,4,3)
imshow(sharp3_2)
title("Sharpened with Gaussian Filtered Image")
subplot(1,4,4)
imshow(sharp3_3)
title("Sharpened with Median Filtered Image")
```

Sobel Filtering Closer Looking

```
clc;clear;close;
% Let's get fruit_color image
img = imread("Valve_original.PNG");
set(0,'DefaultFigureVisible','off')
[x_filtered, y_filtered] = lab2sobelfilt(img);
set(0,'DefaultFigureVisible','on')

figure
subplot(2,2,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(2,2,2);
mag = sqrt(double(x_filtered).^2 + double(y_filtered).^2);
imshow(uint8(mag));
title("Gradient Magnitude", 'FontSize',18);

subplot(2,2,3);
```

```
imshow(y_filtered);
title("Sobel Derivative in y Direction Image, Horizontal Edge Pixels", 'FontSize',18);

subplot(2,2,4);
imshow(x_filtered);
title("Sobel Derivative in x Direction Image, Vertical Edge Pixels", 'FontSize',18);
```