INTRODUCTION

We implemented popular examples of different classes of Local Operators in this lab. These classes are **Smoothing**, **Sharpening**, and **Basic Edge Detecting.**

Image smoothing's main goal is to eliminate noise which are denoted as "outliers" of pixel values from the image. We implemented one *Linear Filtering* of Smoothing which is **Gaussian Filtering,** and one *Non-Linear Filtering* which is **Median Filtering** during the lab.

Sharpening's main goal is to increase contrast level over input image's edges to produce improved image. We implemented the **Unsharp Masking** method during the lab.

Basic Edge Detecting's main goal is to approximate image's either first-order derivatives or second-order derivatives. We implemented the *first-order derivative* version of Basic Edge Detecting by using **Sobel Operator** during the lab.

1. LINEAR FILTERING

Gauss Filter is one of the *Linear Filtering* operators implemented during the lab. Gauss Filter is used for *Smoothing* the image by using special filter kernel which is coming from the 2D Gauss function. When we convolve this filter kernel with our image, we are eliminating the outliers in the input image. In other words, we are reducing the noise in the image.

Gaussian Function and **Filter Kernel** (when k = 2, sigma = 1) can be seen below:

$$G_{\sigma,\mu_x,\mu_y}(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}\right)$$
$$= \frac{1}{2\pi\sigma^2}\cdot e^{-\frac{(x-\mu_x)^2}{2\sigma^2}}\cdot e^{-\frac{(y-\mu_y)^2}{2\sigma^2}}$$

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

/273

We can see the implemented operation as the following formula:

$$L(x,y,\sigma) = [I * G_\sigma](x,y)$$

1.1 My Gaussian Filtering Function Explanation

In my implementation, I started as always by ensuring that the image is grey scale.

```
% Ensuring grey scale image
[row,col,ch] = size(img);
if(ch == 3)
    img = rgb2gray(img);
end
```

Then, I defined my Filter Kernel which is shown in the upper part. Since we have 5x5 kernel size, our k value is 2. Then, I converted my image pixel data to double format to be able to perform mathematical operations. I created a matrix with the same size of input image with zeros.

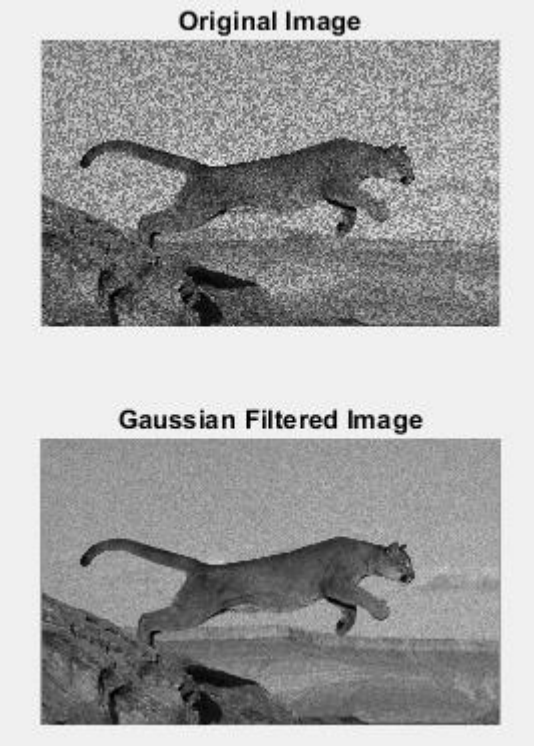
Then, I used double for loop to iterate on the image while doing the convolution operation.

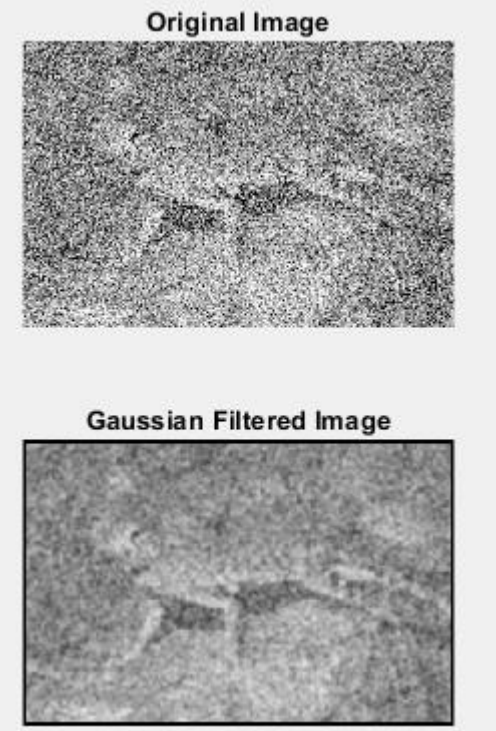
```
20 -  new_img_data = zeros(row, col);
21 -  for i = window_size+1:row-window_size
22 -      for j = window_size+1:col-window_size
23 -          my_window = data((i-window_size):(i+window_size), (j-window_size):(j+window_size));
24 -          smoothed_value = sum(sum(my_window .* kernel));
25    %         temp = my_window .* kernel;
26    %         smoothed_value = sum(temp(:));
27 -          new_img_data(i, j) = smoothed_value;
28 -      end
29 -  end
30 -  new_img = uint8(new_img_data);
```

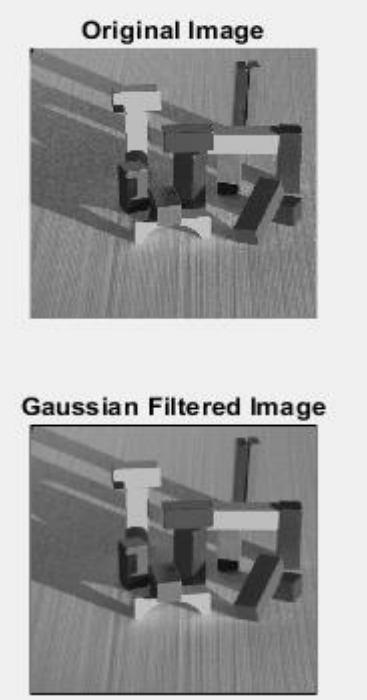
In the for loop part, I decided to leave borders as zero. I could also do zero padding as I did in the previous lab. Since I decided to leave borders as zero, I started to iterate from the index (k+1, k+1) in the image. In each iteration, I took the 5x5 image window to convolve with my filter kernel (this step corresponds to *code line 23*). Then, I did the convolution operation (this step corresponds to *code line 24*). After convolution, I put the result to the center of my window. After loops, I converted my image to uint8 format back again to visually show it.

1.2 Gaussian Filtering Results

I found the second image from google images which is a lizard image with added noise.







1.3 Gaussian Filtering Discussion

We can clearly see that Gaussian Filter can reduce, eliminate the noise in the input image by using a 5x5 filter kernel. Even though we have high noise in the lizard image, Gaussian filter manage to smoothe the input image. We can also see that even we have less noise in the image (Blocks image), Gaussian filter did not make it worse. Therefore, we can say that Gaussian filter is a useful filtering operation for smoothing.

2. Non Linear Filtering

Median Operator one of the *Non Linear Filter* is implemented during the lab. Median operator's main goal is to reduce the noise especially the *"Salt and Pepper noise"* which is strong disturbances mainly black and white pixels in the image. Median Operator slides a window and for each window it replaces the center pixel with the median of the window pixels.

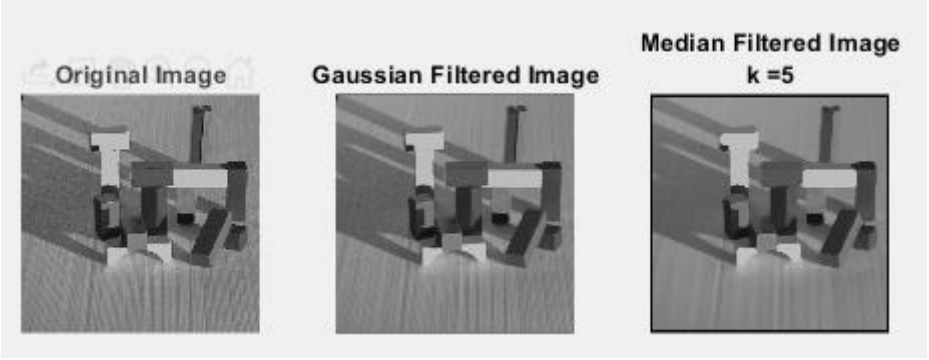
2.1 My Median Operator Function Explanation

In my implementation, I started as always by ensuring that the image is grey scale. Then, I converted my image pixel data to double format to be able to perform mathematical operations. I created a matrix with the same size of input image with zeros. Then, I used double for loop to iterate on the image while replacing the center pixel with the median of the window.

```
18 -  window_size = k;
19 -  for i = window_size+1:row-window_size
20 -      for j = window_size+1:col-window_size
21 -          my_window = data((i-window_size):(i+window_size), (j-window_size):(j+window_size));
22 -          median_value = myMedian(my_window);
23 -          new_img_data(i, j) = median_value;
24 -      end
25 -  end
26 -  new_img = uint8(new_img_data);
```

In each iteration, I took the "k x k" image window to find its median (this step corresponds to *code line 21*). Then, I found the median of the window by using our previously implemented myMedian function (this step corresponds to *code line 22*). After finding median, I put the result to the center of my window. After loops, I converted my image to uint8 format back again to visually show it. myMedian function converts matrix format to vector format. Then, it sorts this vector, and find its median by looking the middle point based on the size of vector. If it is even size, it takes the average of two values in the middle. Otherwise, it takes the middle point directly.

2.2 Median Operator Results



2.3 Median Operator Discussion

We can see that Median Filtering is useful for reducing the “salt and pepper” noise. I think Median Filtering is more effective for dealing with “salt and pepper” noise than the Gaussian Filtering if we compare their results on tiger images. Median Filter’s results are clearer than the Gaussian Filter.

We can also observe that when we increase the window size, results are getting more blurry. Therefore, we should not pick the window size too large. We need to find optimal value by testing different sizes.

In the last image, we can see that when we do not have obvious “salt and pepper” noise, Median Filter did not affect the image too much in both positive and negative ways. On the other hand, Gaussian Filter added some little blur to the image.

3. Sharpening

Sharpening, one of the local operators is implemented during the lab. Sharpening’s main goal is to increase contrast level over input image’s edges to produce enhanced image. While increasing contrast over edges, it tries to minimize additional noise in other regions of the image. This operation can be summarized as following formula:

$$J(p) = I(p) + \lambda\Big[I(p) - S(p)\Big]$$

J is our output image, I is our input image, S is smoothed image, and lambda corresponds to scaling factor and it’s a positive value. Lambda affects the level of the correction part (input – smoothed input) on the final output.

3.1 My Sharpening Function Explanation

In my implementation, I started as always by ensuring that the image is grey scale. Then, I choose the smoothing operator based on the input parameter (M). In my implementation, there are 3 possible smoothing operators: Box Filtering, Gaussian Filtering and Median Filtering.

Then, I applied the selected smoothing filter and hold the returned value (this corresponds to S in the formula).

```
 9 -     k = 3;
10 -     if M == 0
11 -         S = lab1locbox(img, k);
12 -     elseif M == 1
13 -         S = lab2gaussfilt(img);
14 -     else
15 -         S = lab2medfilt(img, k);
16 -     end
```

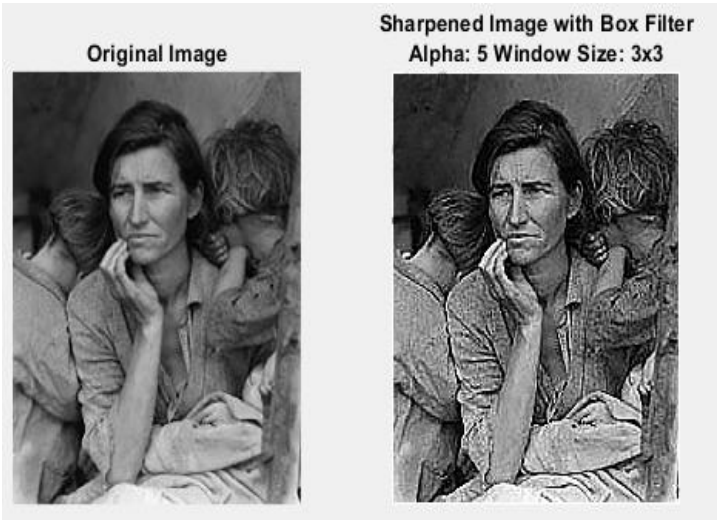
At the end, I applied the sharpening formula with the input parameter lambda after converting both input and smoothed input images to double format. Lastly, I converted to the uint8 format.

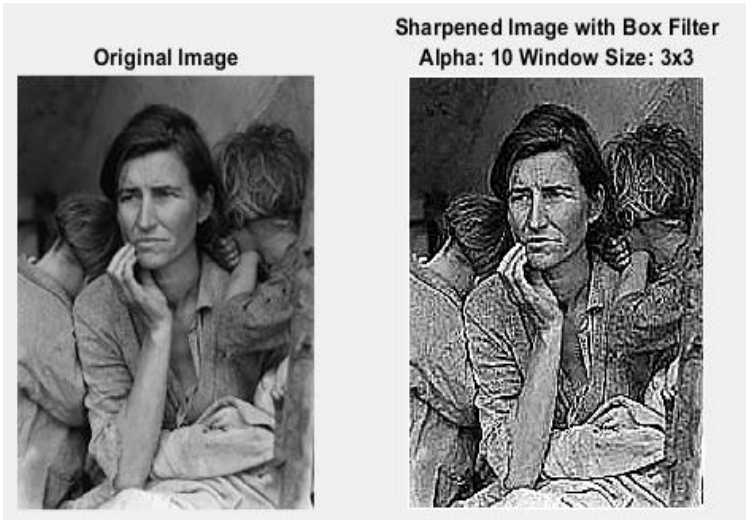
```
18 -     data = double(img);
19 -     data_S = double(S);
20
21 -     new_img_data = data + alpha*(data - data_S);
22 -     new_img = uint8(new_img_data);
```

Ege Erdoğan 25331

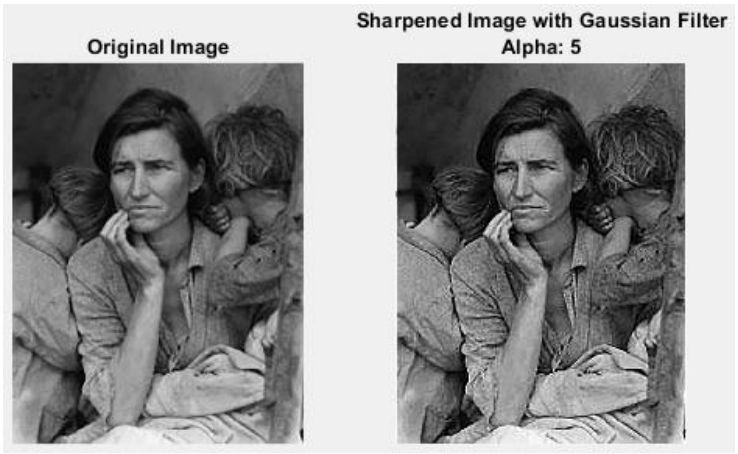
3.2 Sharpening Results

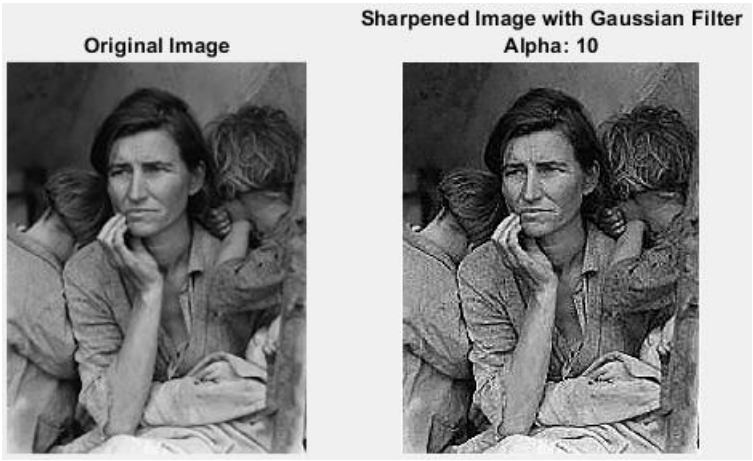
Chosen Smoothing Operator: Box Filter



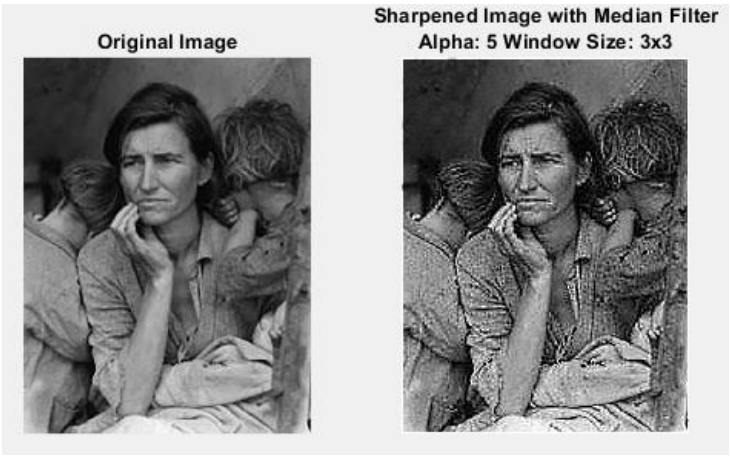


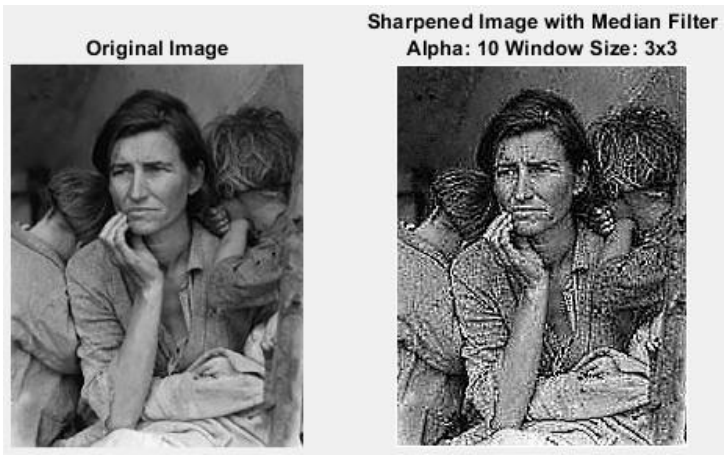
Chosen Smoothing Operator: Gaussian Filtering





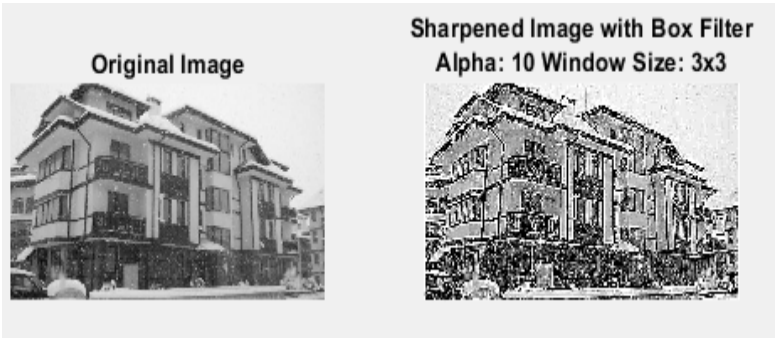
Chosen Smoothing Operator: Median Filtering



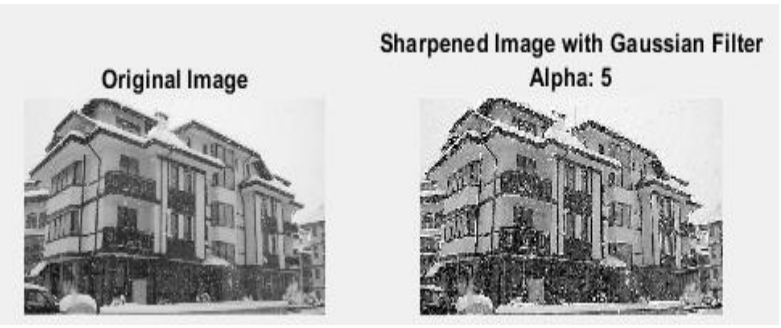


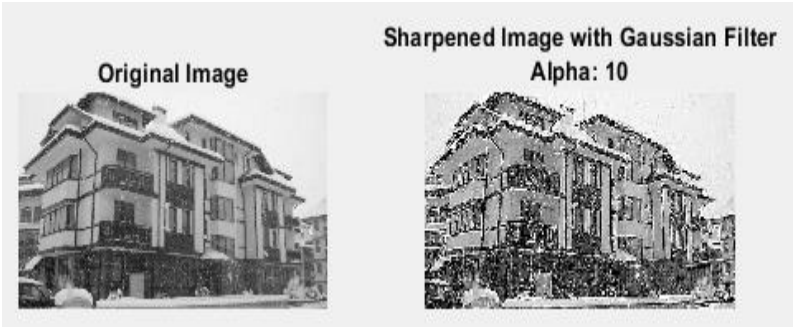
Ege Erdoğan 25331

Chosen Smoothing Operator: Box Filter

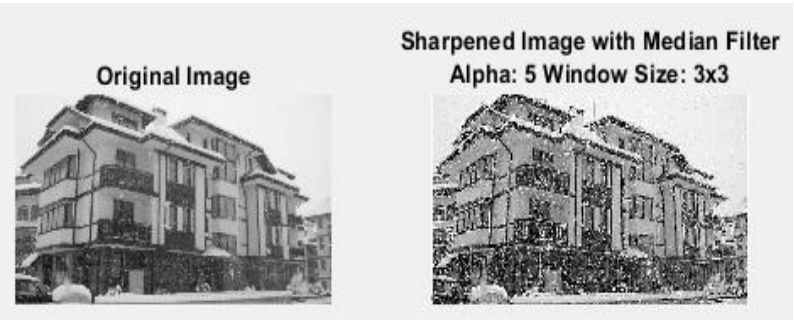


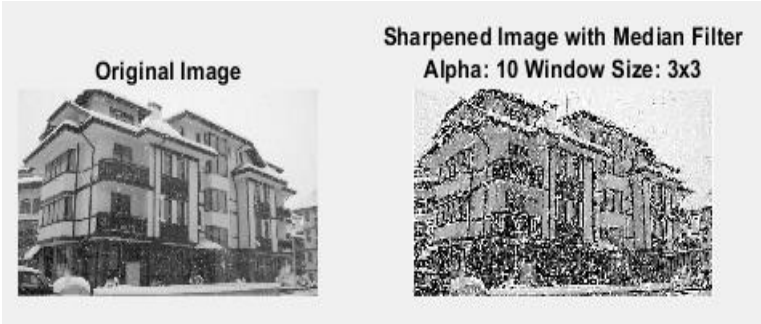
Chosen Smoothing Operator: Gaussian Filter





Chosen Smoothing Operator: Median Filter





3.3 Sharpening Discussion

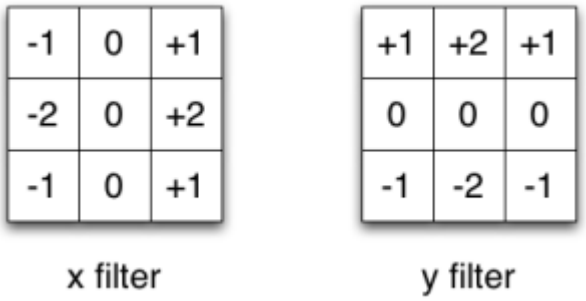
We can see that when we apply sharpening the edges in the image become more obvious as we increase the contrast level of edges. If we look for the same scaling factor with different smoothing operators, I think Gaussian filter is less effective for showing the edges. In other words, increase level of contrast in the edges is lesser than the other two smoothing operators.

We can also observe that when we increase the scaling factor, edges getting more and more contrast, visibility. However, if we increase the scaling factor too much, we are starting to lose homogeneous part of the image. This phenomenon can be seen below:

We can observe clearer in the House image, the edges of the house are getting much more sharpened.

4. Sobel Filtering

Sobel Filtering which is used for edge detection is implemented during the lab. Sobel Filtering uses two different filter kernels to approximate partial derivatives of the input image. Sobel Filtering convolves the following kernels with the image:



If we want to see vertical edges, we will convolve the input image with x filter.

If we want to see horizontal edges, we will convolve the input image with y filter.

4.1 My Sobel Filtering Function Explanation

In my implementation, I started as always by ensuring that the image is grey scale. Then, I defined my two filters which are shown above. Then, I created two matrices with the same size of input image with zeros. Then, I used double for loop to iterate on the image while replacing the center pixel with the convolution result of image window and kernel filter.

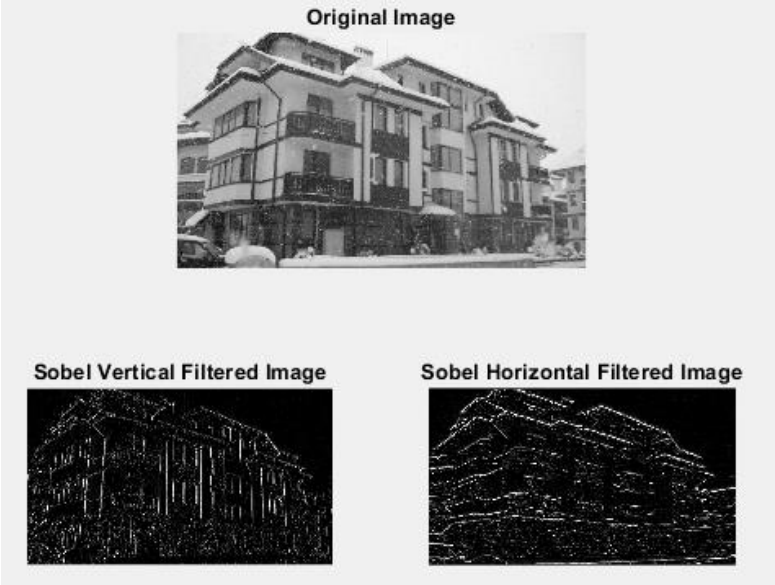
```
23 -  window_size = 1;
24 -  for i = window_size+1:row-window_size
25 -      for j = window_size+1:col-window_size
26 -          my_window = data((i-window_size):(i+window_size), (j-window_size):(j+window_size));
27 -          x_value = sum(sum(my_window .* xfilter));
28 -          y_value = sum(sum(my_window .* yfilter));
29 -          new_img_data(i, j) = x_value;
30 -          new_img_data2(i, j) = y_value;
31 -      end
32 -  end
33 -  new_img = uint8(new_img_data);
34 -  new_img2 = uint8(new_img_data2);
```

In each iteration, I took the “3x 3” image window (this step corresponds to *code line 26*). Then, I convolved this window with the filters (this step corresponds to *code line 27&28*). After convolutions, I put the results to my initial zeros matrices. After loops, I converted my images to uint8 format back again to visually show them.

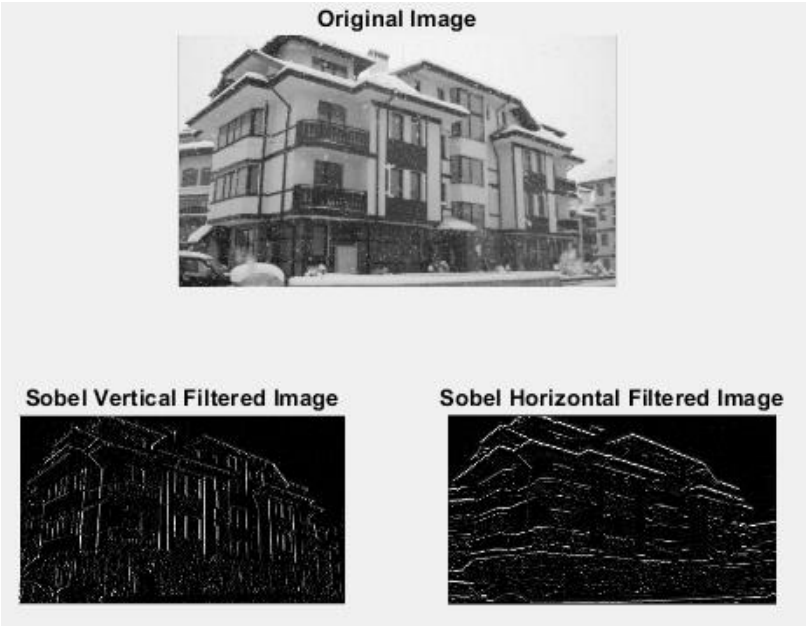
4.2 Sobel Filtering Results

I will show the results of Sobel Filtering without Smoothing, with Gaussian Smoothing, with Gaussian Smoothing and Sharpening.

Without any Smoothing:

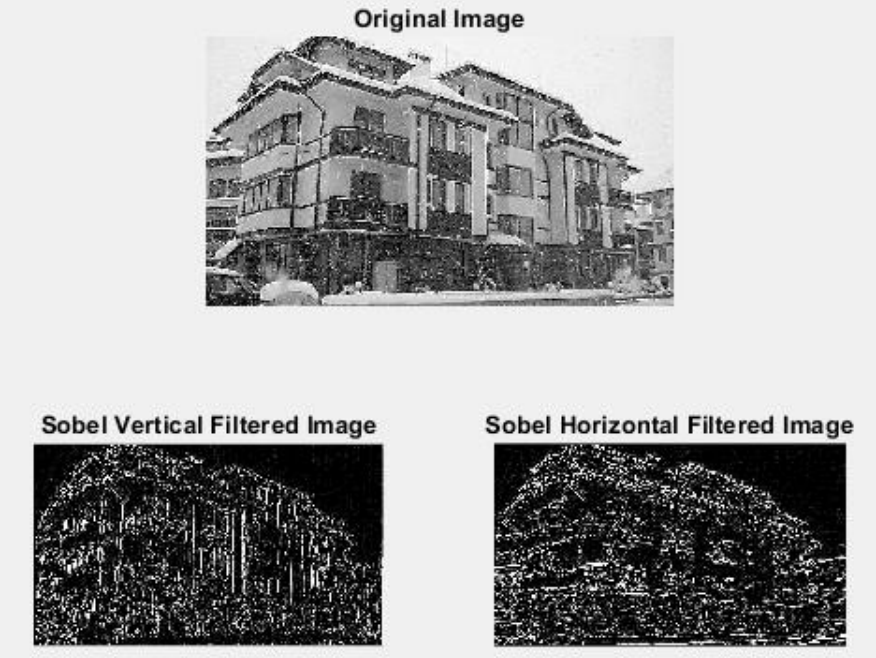


With just Smoothing (Gaussian):

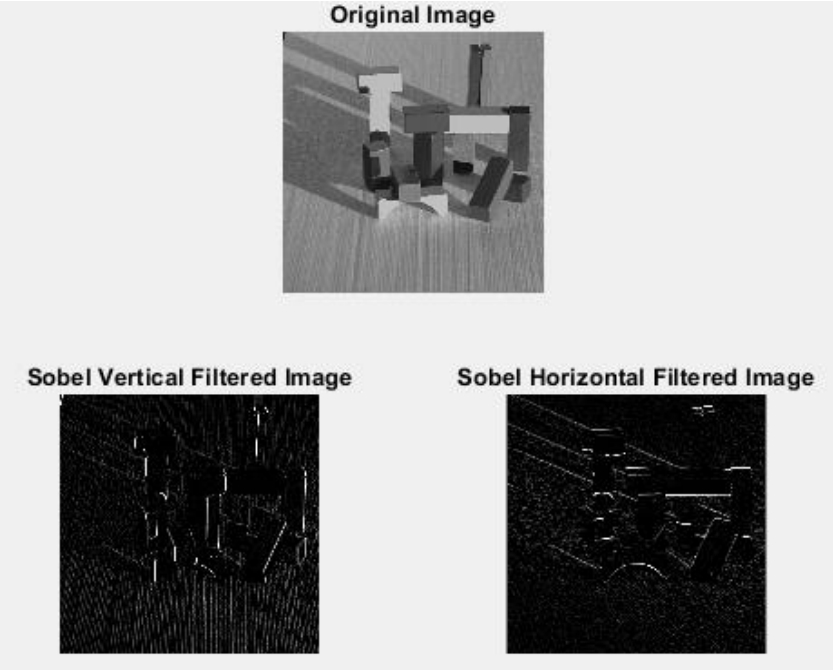


Ege Erdoğan 25331

With Smoothing and Sharpening:

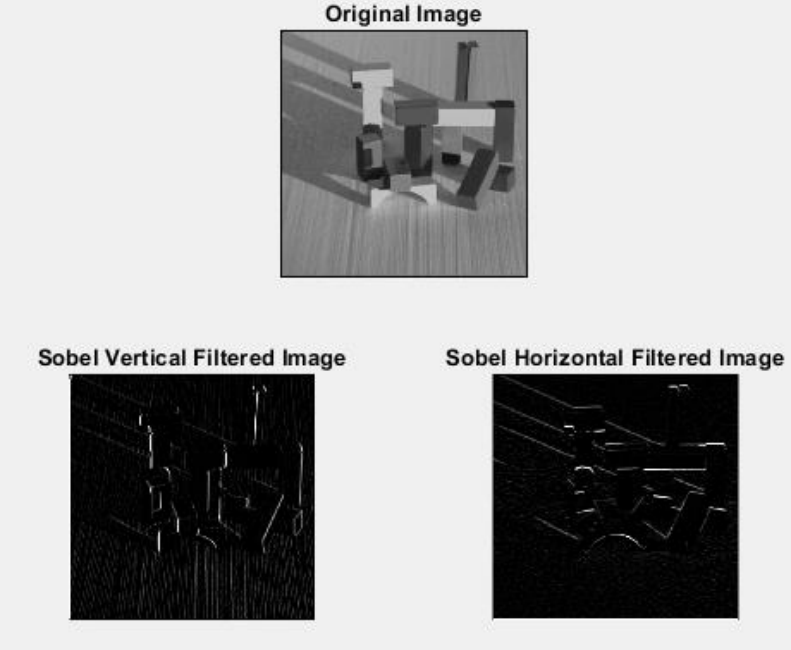


Without any Smoothing:

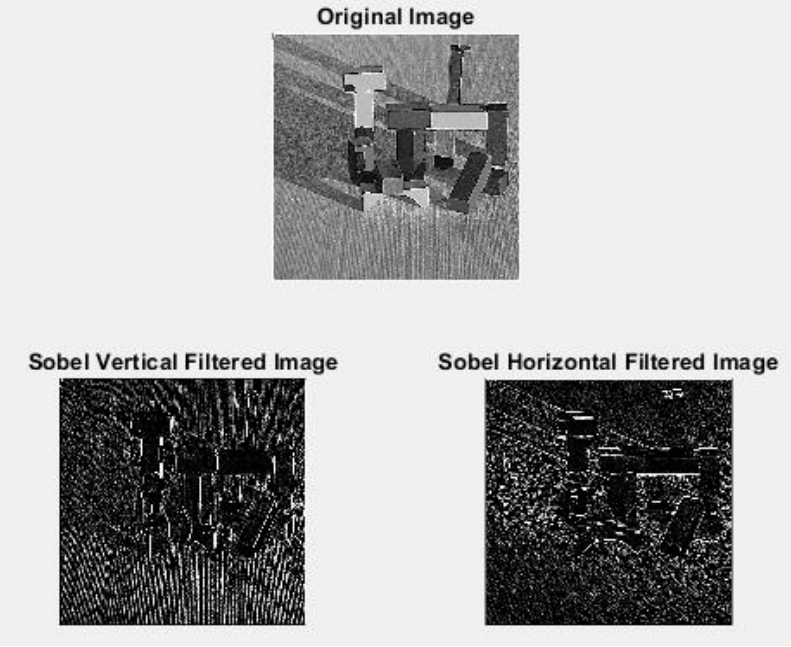


Ege Erdoğan 25331

With just Smoothing:



With Smoothing and Sharpening:



4.3 Sobel Filtering Discussion

We can observe that Sobel Filtering allows us to detect edges in the images and we can choose which kind of edges (horizontal or vertical) we want to see it by convolving with x or y filters.

We also observed that when we apply smoothing and sharpening before edge detecting, they are affecting the result of the edge detection. For instance, when we applied Sharpening to the House image, we got edges with higher contrasts, and it affected the output result.

We observed that we can detect some edges in the Blocks image. Its performance lower compared to House image as the Blocks image's edges are not clear as much as the House image. However, we managed to detect some edges even though this disadvantage.

5. Sigma Filtering

Sigma Filtering, one of the local operators that is used for smoothing operation, is implemented during the lab. Sigma Filtering is smoothing the image by using sliding window while taking the mean of the neighbours' pixels which have the values in the given range.

This range (interval) can be seen below:

$$[I(p) - \sigma, I(p) + \sigma]$$

Calculation of mean by using histogram can be seen below:

$$\mu = \frac{1}{S} \cdot \sum_{u=I(p)-\sigma}^{I(p)+\sigma} u \cdot H(u)$$

5.1 My Sigma Filtering Function Explanation

In my implementation, I started as always by ensuring that the image is grey scale. Then, I created two matrices with the same size of input image with zeros. Then, I used double for loop to slide a window on the image.

```
13    % % Double for loop for sliding window
14 -  window_size = k;
15 -  for i = window_size+1:row-window_size
16 -      for j = window_size+1:col-window_size
17 -          my_window = data((i-window_size):(i+window_size), (j-window_size):(j+window_size)); %Taking window
18 -          [H, bin] = imhist(uint8(my_window)); %Histogram of the taken window
19 -          upper_bound = data(i, j) + var; %Defining interval's upper bound
20 -          lower_bound = data(i, j) - var; %Defining interval's lower bound
21 -          if lower_bound < 0  %Lower bound cannot be lower than zero
22 -              lower_bound = 0;
23 -          end
24 -          if upper_bound > 255 %Upper bound cannot be larger than 255
25 -              upper_bound = 255;
26 -          end
27 -          S = 0;
28 -          summation = 0;
29 -          for u = lower_bound:upper_bound   %Calculating mean of the pixels that are in the interval
30 -              summation = summation + (u * H(u+1)); %Summation part of the formula
31 -              S = S + H(u+1);  %Number of elements that are in the interval
32 -          end
33 -          my_mean = summation / S; % Mean of the interval
34 -          new_img_data(i, j) = my_mean;
35 -      end
36 -  end
```

In each iteration, we are taking a window from our image, calculating histogram of this window and we are defining our lower and upper bounds for our interval. *(These steps correspond to from line 14 to 20)*

Since we can not have pixel values lower than 0 and higher than 255 in our histogram, I used if statements to be ensure these bounds are correct. *(This step corresponds to line 21 to 26)*

Then, I implemented the algorithm to calculate following Sigma Filter Formula: *(line 27 to 33)*

$$\mu = \frac{1}{S} \cdot \sum_{u=I(p)-\sigma}^{I(p)+\sigma} u \cdot H(u)$$

S equals to our Histogram counts for given interval.

Summation equals to multiplication of pixel value and its Histogram counts.

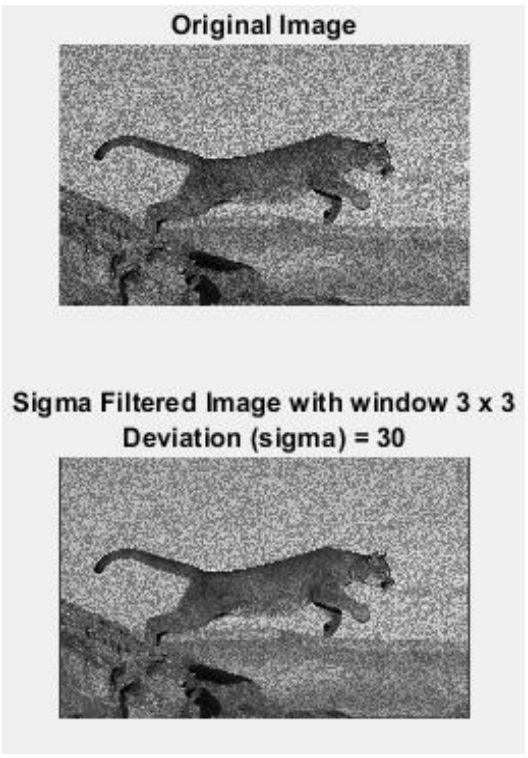
When we divide summation with S, we obtain the mean value for the given interval, and we are putting this value to our center of window.

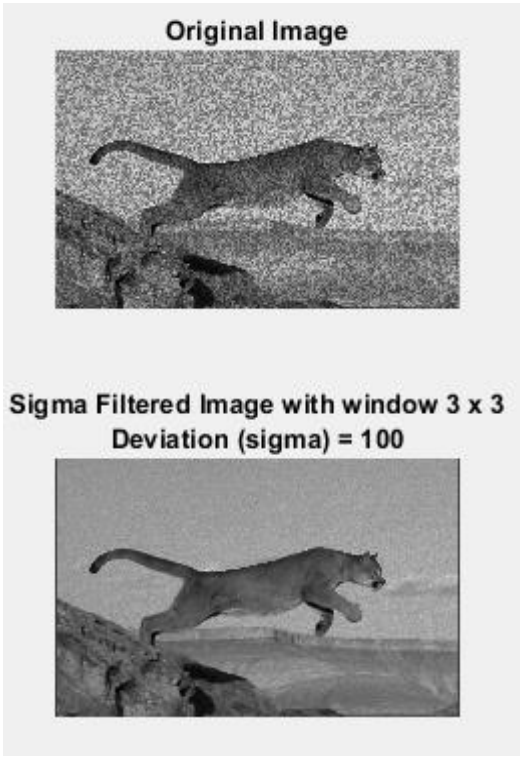
At the end, new image is formatted back to uint8 to see it visually.

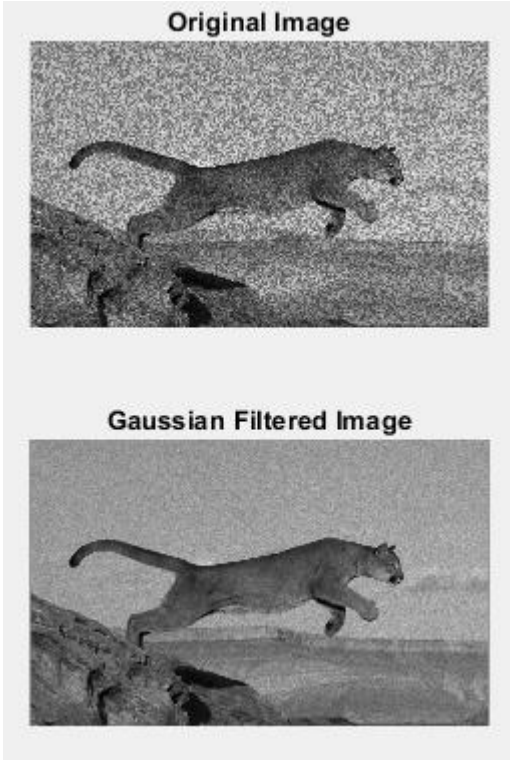
5.2 Sigma Filtering Results

I will present the results of filter with different sigma values and comparison with Gauss Filter.

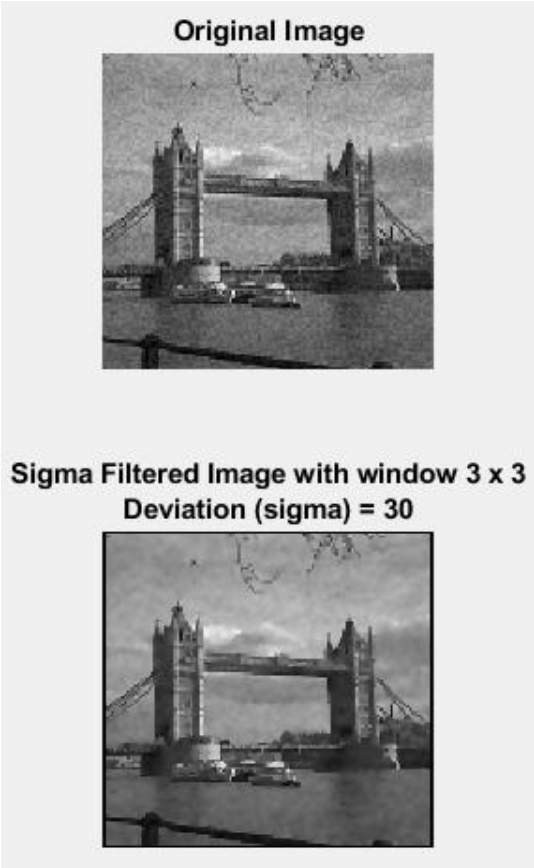
JUMPING PUMA IMAGE

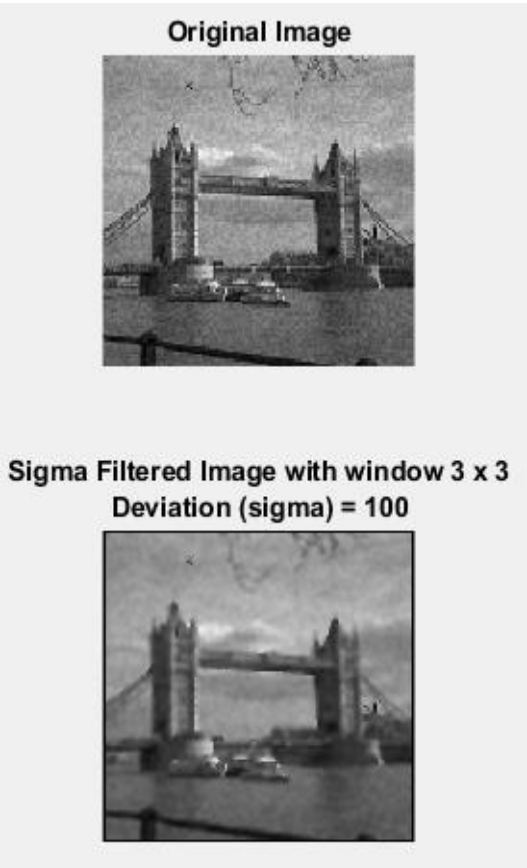






BRIDGE IMAGE (taken from book's web page)





5.3 Sigma Filtering Discussion

We can see that Sigma Filter can be used for noise removal from the input image. In other words, it can be used for smoothing.

We can observe that sigma which is used for defining the histogram interval is an important parameter for the Sigma Filter. For instance, in the first example (Jumping Puma Image), when we have sigma as 30, Sigma Filter could not remove the noise properly. However, when we increased the sigma value to 100, Sigma Filter managed to remove noise better than previous case. We can also observe that if do not choose sigma value carefully, we can end up with blurry image as it can be seen in the second example of Bridge Image with sigma as 100.

5.4 Sigma Filter Comparisons with Different Filters

When we choose sigma value properly, result of Sigma Filter is compatible with other filters like Gaussian Filter. However, choosing sigma is not easy task, so I can say that other filters like Gaussian and Box Filter are more preferable than Sigma Filter.

I also observed that Sigma Filter's computational time is larger than other filters. For instance, filtering the Jumping Puma image with sigma value 100 is taking too much time compared to Gaussian Filter. The reason of extra time is that we have another loop inside the algorithm to choose only pixel values that are in the interval. This loop adding additional computational time for our algorithm.

Consequently, I think that Gaussian Filter is better than Sigma Filter in both computation time and overall quality of output image.