EE 569 3D Vision: Post-Lab Report 1

Furkan Gül, Student ID: 28795

November 3, 2020

1 Experiments in Lab

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

In some cases, there exists inefficient or excessive lighting in images. This will make the images not only inefficient but also unfavorable to be used later on. There exist several reasons for this undesirable severe effect on the images. A possible couple of reasons for that low quality of recorded digital images may arise from the poor background light in the camera location, taking a picture at the night without an advanced night mode option in the camera settings. The images taken under poor lighting conditions considered as a part of ”noisy” images. In Lab, two methods were introduced to overcome this lighting issue. These are just two different point operator scaling approaches for the pixel values in monochrome images, which are linear scaling and conditional scaling.

Before diving into the first method for unwanted lighting issues, which is linear scaling, it will be better to mention the main script that calls all the required functions benefited in Lab. It is named ”Lab1Main”, which can be found in the Appendix. Since there are mainly 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. They all start with checking the input images if it is RGB or monochrome. 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.

The first section in the main script is dedicated to the linear scaling solution for lighting problems in images. In this section, after getting the input image, ”lab1linscale” function is called by this input image and returns a linearly scaled output image. This function can be found in the Appendix. In this function, the pixel values of the original image will be linearly

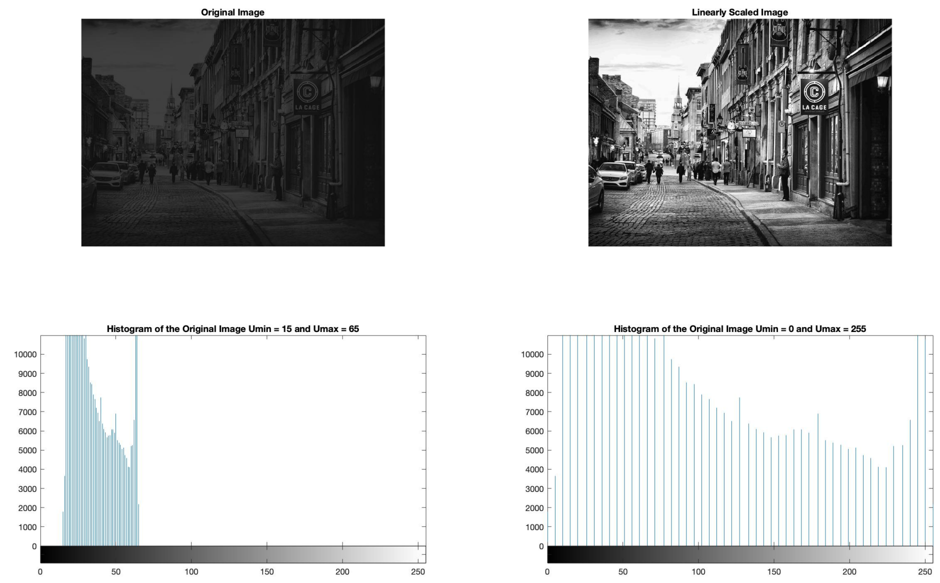


Figure 1: Linear Scaling of Image

scaled by making use of the gradation function discussed in one of the lectures. So that the returned image's pixel values will be between 0 and 255. Thus, the resulting image becomes clear in terms of understanding the context and details of the image. In the last part of function, both original and linearly scaled images are plotted along with their histograms. It can be seen in Figure 1.

In the second section of the main script, there is another method to deal with the lighting issue. That is called conditional scaling. The corresponding function name for that method is "lab1condscale" which can lie under the Appendix. It takes the original input image and reference image and returns the conditionally scaled output image. In this specific function, the input image will be conditionally scaled by the reference image in terms of the mean and standard deviation by using a gradation function for conditional scaling mentioned in the class. It means that the output image will have the same semantic context as the input image, but it will have the same mean and standard deviation of pixel values as the reference image, not the input image. Finally, the current input image, the reference image, and the returned image will be displayed in a Matlab figure together with their pixel values' mean and standard deviation respectively under each image. It is seen in Figure 2.

Apart from the poor and insufficient lighting problems, there occurs other severe issues in images such as motion blur and sudden scene changes. Both of them will also reduce the quality of the images taken. The images with those issues are also considered as "noisy" images. In order to get rid of these undesired effects on images, two local operator methods introduced in Lab are box filtering and local max and local min filtering.

The third part of the main script is written for the problem of motion blur and sudden movement in objects and cameras. The approach is named the local mean filter or box filter.

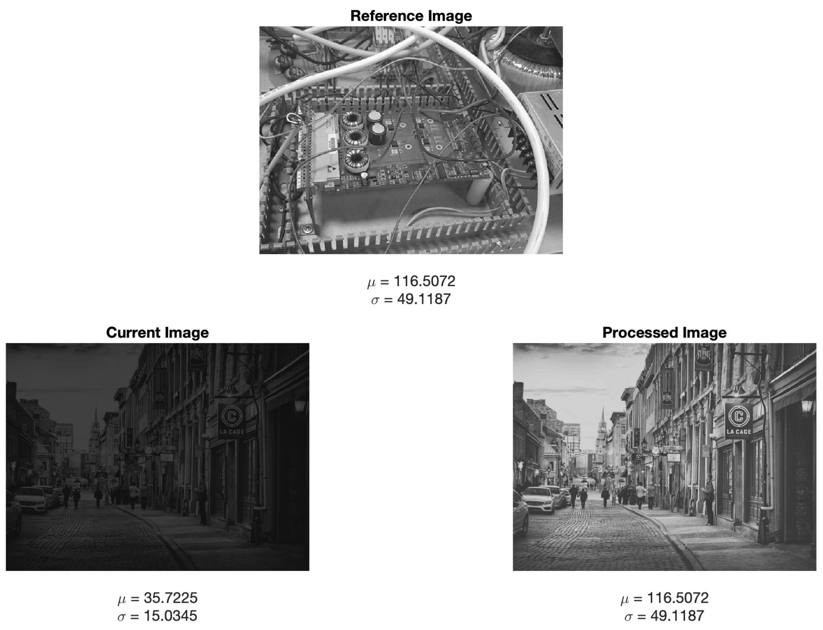


Figure 2: Conditional Scaling of Image

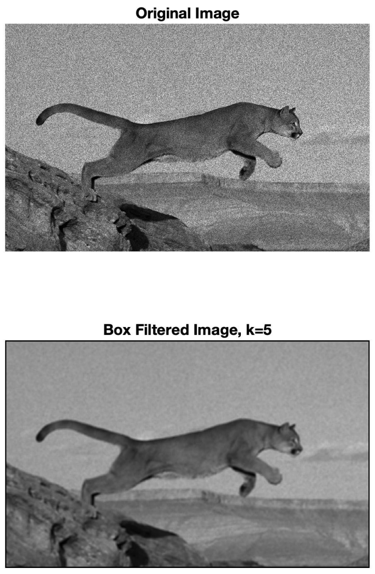


Figure 3: Box Filtering of Image

The name of the function applies this method is "lab1locbox", which can be seen in detail in the Appendix. This function takes the input image and window/filter size parameter as inputs and returns the box filtered output image. The windows 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 mean of the pixel values in the intersection of images will be calculated. Those local mean values will be put into another new matrix. This new image representation will be the local mean filtered version of the input image. Finally, both the original input image and the box filtered image with the corresponding window size parameter will be plotted in Matlab figure, which can be seen in Figure 3.

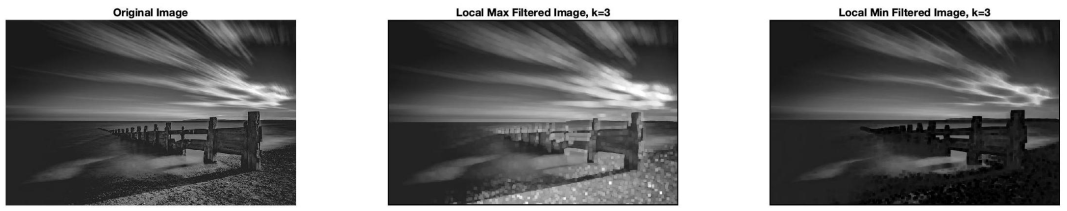


Figure 4: Local Maximum & Local Minimum Filtering of Image

In the final fourth section of the main script, local max and local min filters will be addressed to remove the dilation and erosion of pixels in images. "lab1locmaxmin" named function will be called with the input image and window size parameter. It will return the output local max filtered image and local min filtered image. This function can be found in the Appendix. In this function, the same sliding operation as it is in the "lab1locbox" will be processed. But this time, instead of using the mean of pixel values, the max and the min pixel value corresponding to each window slide will form two different matrices. Lastly, the local max and local min filtered images with the given window size along with the original input image will be displayed in the Matlab figure. This can be see in Figure 4.

2 Results & Discussions

To apply four fundamental image processing techniques learned in Lab, two astonishing images were taken from B-9 Dormitory in Sabanci University. The semantic context of both images is the same. They cover the lake, the performing art center, the sports center, and finally some household lights. One is just a clear shot without any sudden movement in the camera. The other one is a blurred version of that picture due to some shaking in the hand that holds the camera. Both colored versions of images can be seen in Figure 4.

In the first part, let's investigate the images with the linear scaling function. To be able to apply the linear scaling operation to any image, the pixel value range of the image must

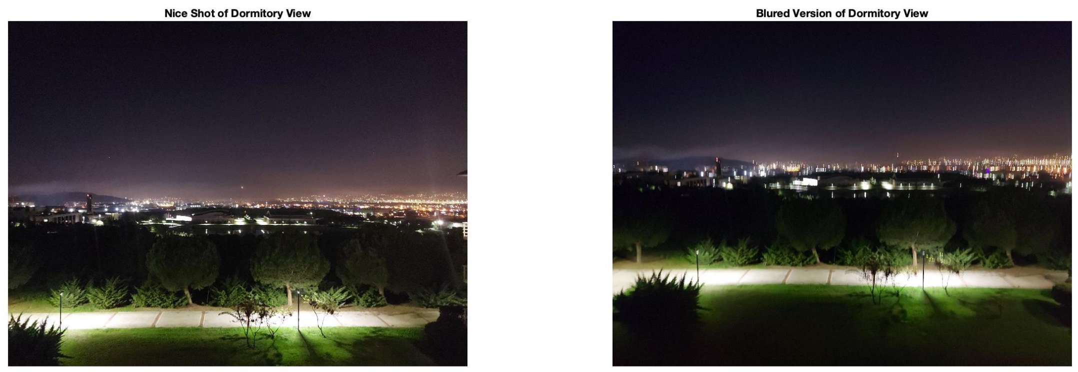


Figure 5: Color Image of Dormitory View

be different than [0, 255]. If that is not the case, the function will return the same image since the pixel values are already in between 0 and 255. In Figure 6, the original image's pixel values lie between 0 and 228, which is almost equal to [0, 255] pixel value range. That's why the linear scaling method didn't make any observable change to the original image as it can be also inferred from Figure 6. Since the dormitory image has also the pixel range of [0, 255], to see the linear scaling effect clearly on the dormitory image, the dorm image was pre-processed by reducing its pixel values to the fifth of the original values. Now, as can be seen in Figure 7, the resulting processed dormitory image is acceptable for the linear scaling.

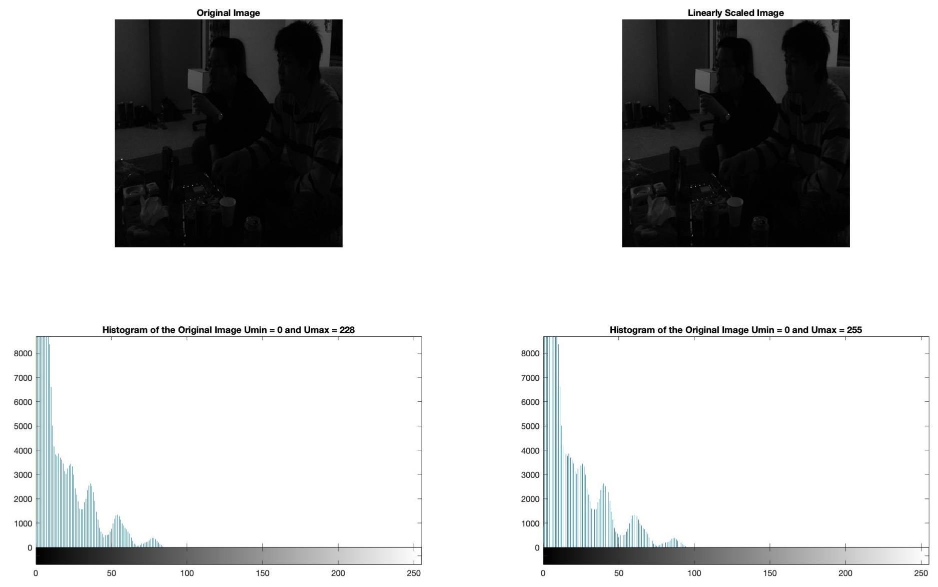


Figure 6: Linear Scaling of Image with Higher Range of Pixel Values

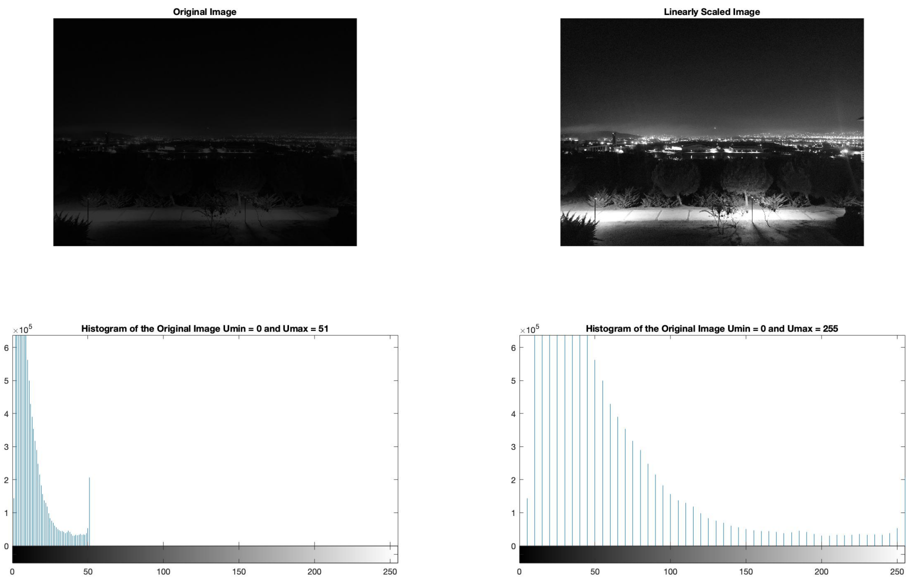


Figure 7: Linear Scaling of Modified Dorm Image

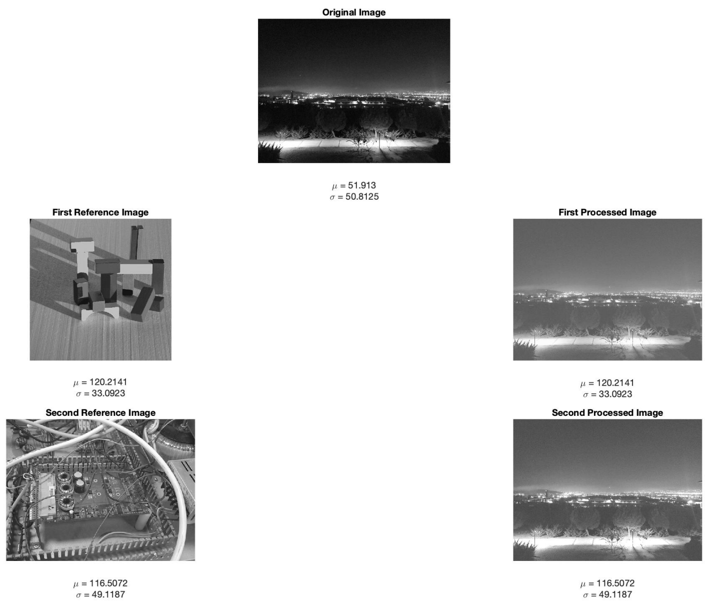


Figure 8: Conditional Scaling of Dorm Image wrt Block and Board Images

Secondly, the dorm image will be processed by the conditional scaling point operator. Two reference images are conditioned on the dorm image separately to explain why the two resulting images are different and where this difference comes from. The whole process for these two conditional scalings can be seen in Figure 8. It is also verified from Figure 8 that the conditional scaling will make the mean and standard deviation of dorm image's pixel values the same as those of reference images' pixel values. The mean of the first and second reference images' pixel values are very close to each other and it is near 120. Thus, the first deduction is that the average pixel intensity within two processed images will be very similar. However, when the standard deviations of those two reference images' pixel values are considered, there is a slight difference, which is about 20. The second deduction means that the pixel intensity variation in the second processed image will be more than that in the first processed image. Those two deductions from the mean and standard deviation of the reference images can be also observed nicely by just look at both the preprocessed images in Figure 8.

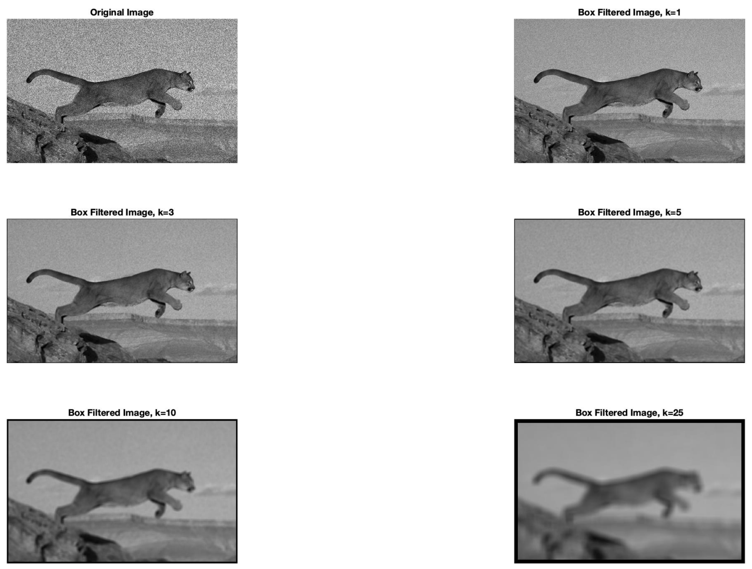


Figure 9: Box Filtering of Jump Image with Different Kernel Sizes

In the third part, the jump image used in Lab will be the discussion of the local mean filtering with different window size parameters which are 1, 3, 5, 10, 25. The resulting filtered images with those filter size parameters can be seen in detail in Figure 9. There are two fundamental takeaways from this experiment. The first one is that when the filter size increased too much, the image will be blurred more and more even it can't be seen clearly. The best filter size parameters k equals 1 and 2, as it is obvious from Figure 9. When k is 1, the filter will be 3 by 3. When it is 2, it is 5 by 5. For both of those two cases, the tiger jump scene can be seen very clearly. However, when k becomes larger as it is 10 and 25, the blur in images due to larger filter size will not allow us to identify even the context in the image. The second obvious result is that the black window frame will become more visible when k is larger. This is because at first, the resulting image's matrix initiated as a zero matrix. From the top, bottom, left, and right edges of the image to the k pixel values in the corresponding direction will be 0, which represents the black color. However, it can be avoided using methods like copy edge (replicate) and reflect across edge for boundary issues. These methods are kind of padding effectively so that the distribution of color or intensity values across edges of the image is not likely to change much .

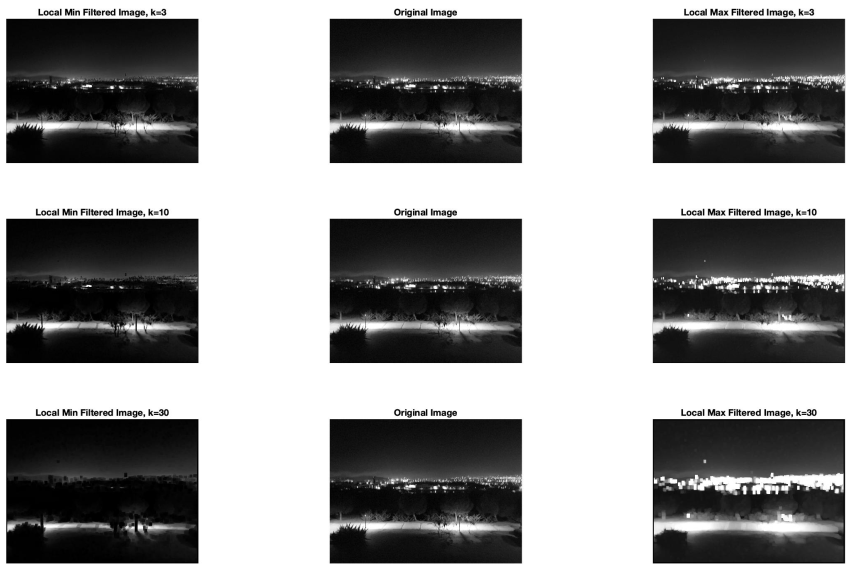

Local Min Filtered Image, k=3
Original Image
Local Max Filtered Image, k=3
Local Min Filtered Image, k=10
Original Image
Local Max Filtered Image, k=10
Local Min Filtered Image, k=30
Original Image
Local Max Filtered Image, k=30

Figure 10: Local Minimum and Local Maximum Filtered Images with Different Kernel Sizes

Lastly, the amazingly perfect view of B9-dorm will be processed via the local min and local max filters. Three different window size parameters, k = 3, 10, 30, are used to specify how the original image will change with respect to those different filter sizes due to different k values. The original dorm image occupied the middle region of Figure 10. As it is observed from the three local min filtered images in the left portion of Figure 10, if the window size parameter, k, increases, the processed local min images will become more black locally. When k is 30, the lights of households in the original image will dissappear. Conversely, when three local max filtered images in the right section of Figure 10 considered, the white will be locally more dominant in the processed local max images as k gets bigger. This time when k is 30, the lights of households distributed over the image locally.

A Appendix

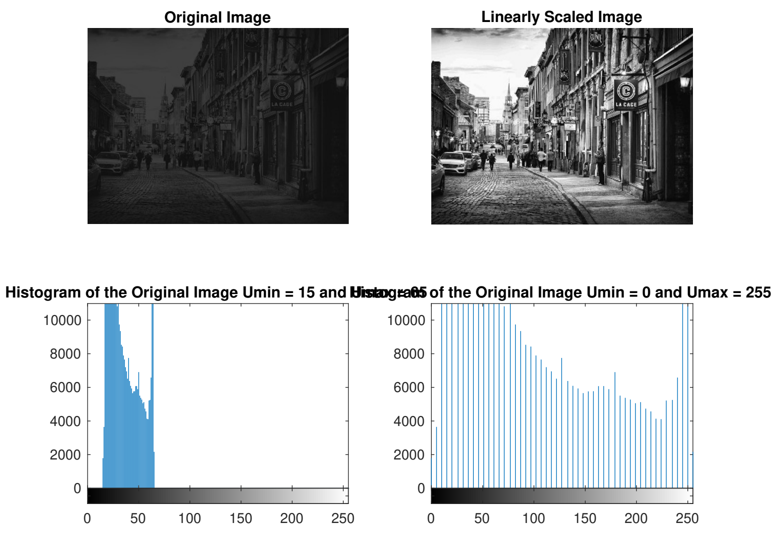
All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

Contents

```
% Below four sections can be runned seperately by using the "Run Section" comment
% under "Editor" Tab
```

Linear Scaling

```
clc;clear;close;

% Let's get the city image
Im = imread("city.png");
% Call the lab1linscale function
[~] = lab1linscale(Im);
% In order to remove orange warning, replace Inew with ~ since we'll not use Inew.
```

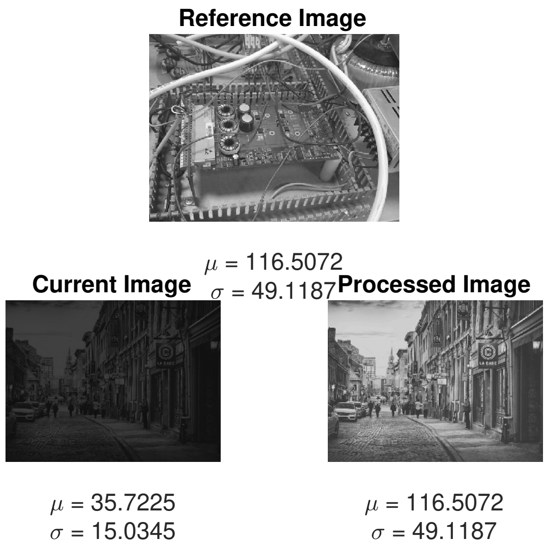


```
function [Inew] = lab1linscale(Im)
% Chech the image if it is RGB or monocolor
[~, ~, c] = size(Im);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    Im = rgb2gray(Im);
end
```

```
% To make math operations on the monocolor image, change type of image matrix to double
I = double(Im);

%%Let’s linearly scale the original image using below gradation function (No need to
% make this line Section, think of it as a heading)

% It will make I matrix to column vector and take min element
a = -min(I(:));

Gmax = 255;

b = Gmax/(max(I(:))- min(I(:)));

Inew = b*(I+a);

% Change the datatype of image matrix from double to uint8 so that it can be displayed
% without any problem.
Inew = uint8(Inew);

% Plot the original and scaled images with their corresponding histograms via subplot
% function
figure()

subplot(2,2,1);
imshow(Im);
title("Original Image");

subplot(2,2,2);
imshow(Inew);
title("Linearly Scaled Image");

subplot(2,2,3);
histogram(Im);
imhist(Im); % imhist is better than histogram function. It will give the same result
% in the pdf.
title("Histogram of the Original Image Umin = "+min(I(:))+" and Umax = "+ max(I(:)));

subplot(2,2,4);
imhist(Inew);
title("Histogram of the Original Image Umin = 0 and Umax = 255")

end
```

Conditional Scaling

```
clc;clear;close;

% Let's get the city and board images
Im = imread("city.png");
Iref = imread("board.jpg");
% Call the lab1condscale function
[Jnew] = lab1condscale(Im,Iref);
```



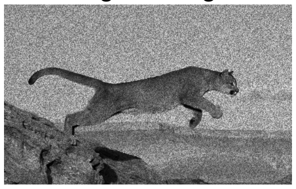
```
function [Jnew] = lab1condscale(Im,Iref)
% Chech the original image if it is RGB or monocolor
[~, ~, c] = size(Im);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    Im = rgb2gray(Im);
end
% Chech the reference image if it is RGB or monocolor
[~, ~, c] = size(Iref);

% If the refrence image is RGB (colored), then convert it to grayscale image
if c == 3
    Iref = rgb2gray(Iref);
end

% To be able to make math operations on the monocolor images, change theirs' datatype
% to double
J = double(Im);
I = double(Iref);
```

```
%%Let's condition the mean and variance of the original image with the mean and
% variance of the reference image using below gradation function
%(No need to make this line Section, think of it as a heading)

b = std(I(:))/std(J(:));
a = mean(I(:))/b - mean(J(:));

Jnew = b*(J+a);

% Let's check if the conditioned new image(Jnew) has the same mean and variance of the
% reference image(J) or not.
%disp([mean(I(:)), mean(J(:)), mean(Jnew(:))]);
%disp([std(I(:)), std(J(:)), std(Jnew(:))]);

% Change the datatype of conditioned image matrix from double to uint8 so that
% it can be displayed without any problem.
% Since I will use Im and Iref labels for the purpose of plotting, there is
% no need to convert their datatypes since they are already in uint8.

Jnew_uint = uint8(Jnew); % Make Jnew image

% Plot the reference, current, and processed images via subplot function
figure()

subplot(2,2,[1, 2]);
imshow(Iref);
title("Reference Image", 'FontSize',18);
% Let's make xlabel as similar as it is in the pdf document
xlabel(["\mu = " + mean(I(:));"\sigma = " + std(I(:))], 'FontSize',18)

subplot(2,2,3);
imshow(Im);
title("Current Image", 'FontSize',18);
xlabel(["\mu = " + mean(J(:));"\sigma = " + std(J(:))], 'FontSize',18)

subplot(2,2,4);
imshow(Jnew_uint);
title("Processed Image", 'FontSize',18);
xlabel(["\mu = " + mean(Jnew(:));"\sigma = " + std(Jnew(:))], 'FontSize',18)
end
```

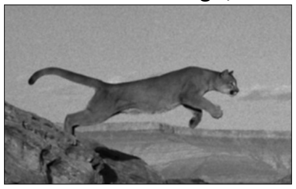
Box Filter

```
clc;clear;close;

% Let’s get the jump image
Im = imread("jump.png");
% Number for window(filter) size
k = 5;
[Inew] = lab1locbox(Im, k);
```

Original Image



Box Filtered Image, k=5

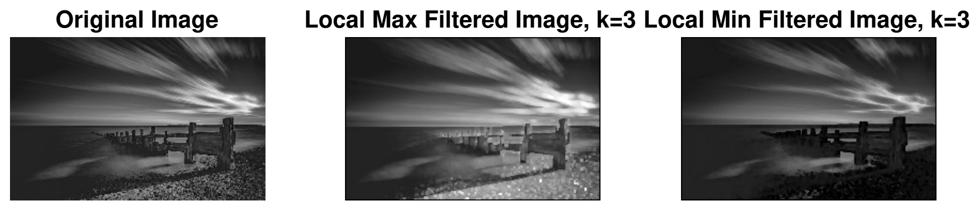


```
function [Inew] = lab1locbox(Im, k)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(Im);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    Im = rgb2gray(Im);
end

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

% Initialize matrix of the box filtered image as 0s with the same dimension
% of the original matrix
Inew = zeros(h,w);

% 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
```

```
        wp = I(i-k:i+k, j-k:j+k);
        % Calculate the mean of pixel values of the substracted part of
        % image after making it column vector
        Inew(i,j)= mean(wp(:));
    end
end

% Change the datatype of image matrix from double to uint8 so that it can be displayed
% without any problem.
Inew = uint8(Inew);

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

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

subplot(2,1,2);
imshow(Inew);
title("Box Filtered Image, k="+int2str(k),'FontSize',18);
end
```

Local Min Max Filter

```
clc;clear;close;

% Let's get the current image
Im = imread("currentImage.png");
% Number for window(filter) size
k = 3;
[Imax, Imin] = lab1locmaxmin(Im, k);
```



```
function [Imax, Imin] = lab1locmaxmin(Im, k)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(Im);

% If the image is RGB (colored), then convert it to grayscale image
```

```
if c == 3
    Im = rgb2gray(Im);
end

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

% Initialize matrices of the local min & max filtered images as 0s with the same
% dimension of the original matrix
Imax = zeros(h,w);
Imin = zeros(h,w);

% 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
        wp = I(i-k:i+k, j-k:j+k);
        % Calculate the max & min of pixel values of the substracted part
        % of image after making it column vector
        Imax(i,j)= max(wp(:));
        Imin(i,j)= min(wp(:));
    end
end

% Change the datatype of the local min & max filtered images from double to uint8
% so that it can be displayed without any problem.
Imax = uint8(Imax);
Imin = uint8(Imin);

% Plot the original, the local min & max filtered images via subplot function
figure()

subplot(1,3,1);
imshow(Im);
title("Original Image");

subplot(1,3,2);
imshow(Imax);
title("Local Max Filtered Image, k="+int2str(k));

subplot(1,3,3);
imshow(Imin);
title("Local Min Filtered Image, k="+int2str(k));
```

```
end
```