International Journal of Engineering Sciences Paradigms and Researches (IJESPR)
Volume 48, Issue 03, Quarter 03 (July-Aug-Sept 2019)
An Indexed and Referred Journal with Impact Factor: 2.80
ISSN (Online): 2319-6564
www.ijesonline.com

Implementation for De-noising of Images by Different Filters

Sonal Goyal[1] and Devender Kumar Saini[2]

[1]M. Tech. Student,
Department of ECE, GITM, Gurgaon, Haryana, India

[2]Assistant Professor,
Department of ECE, GITM, Gurgaon, Haryana, India

Publishing Date: July 18, 2019

Abstract

The paper is based on digital image processing. In imaging science, image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Image processing usually refers to digital image processing, but optical and analog image processing also are possible. Digital image processing is the use of computer algorithms to perform image processing on digital images. Digital image processing deals with manipulation of digital images through a digital computer, it is a subfield of signals and systems but focus particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image. The input of that system is a digital image and the system process that image using efficient algorithms, and gives an image as an output. The basic aim is to study the various types of noise that can cause blurring of an image. And also the various types of filters that are used to reduce the noise to de-blur the image will be studied. All the effects of various types of noise involved are shown using MATLAB. Also we will be writing the MATLAB code for various types of filters involved.

Keywords: *Image Processing, MATLAB, Noise, Filters.*

Introduction

Digital image processing deals with manipulation of digital images through a digital computer. It is a subfield of signals and systems but focuses particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image. The input of that system is a digital image and the system process that image using efficient algorithms, and gives an image as an output. The most common example is Adobe Photoshop. It is one of the widely used applications for processing digital images.

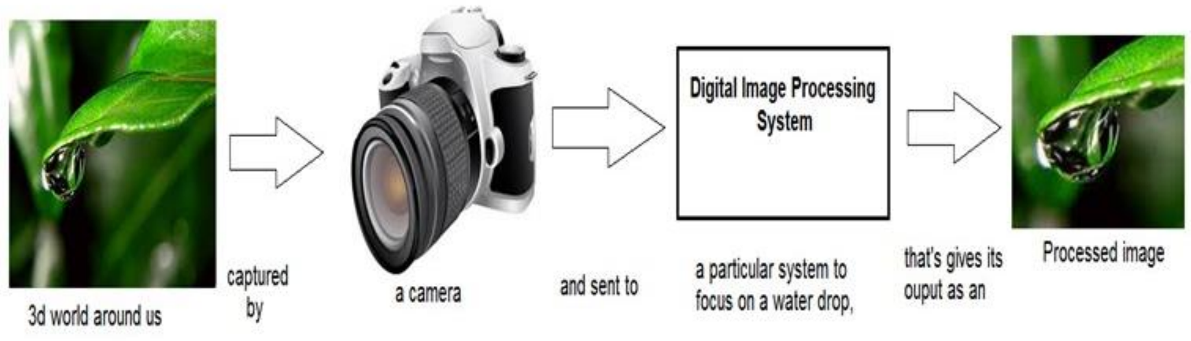

3d world around us
captured by
a camera
and sent to
Digital Image Processing System
a particular system to focus on a water drop,
that's gives its ouput as an
Processed image

Figure 1: Illustration of Digital Image Processing

In the above Figure 1 an image has been captured by a camera and has been sent to a digital system to remove all the other details, and just focus on the water drop by zooming it in such a way that the quality of the image remains the same.

In our major project we have also attempted to implement some of the widely used methods and procedures for interpreting digital images for image enhancement and restoration and performing operations on images such as (blurring, zooming, sharpening, edge detection, etc).

Our approach is simple and in demonstrating image processing operations we have used the software platform of MATLAB. There are many subareas under the area of image processing. Among it we have dealt with image denoising.

Initially, we started with simple MATLAB functions and we have demonstrated how they manipulate images. In our project we have worked with grayscale images of size 256*256. Then we have implemented our objective of image denoising. There are many types of noise and we have majorly worked with salt and pepper noise. Apart, we have also demonstrated techniques of image noising. In image noising we have corrupted our image with noises such as salt and pepper, salt, pepper, Gaussian, speckle, Poisson etc.

Under image denoising portion of our project we have concentrated only on techniques used to denoise images corrupted by salt and pepper noise. Many filters are implemented. Two approaches are followed for both - implementation of filter as well as for demonstration of results obtained. In implementation, the two approaches are:

i. Use of inbuilt functions of MATLAB
ii. Use of user defined functions implemented in MATLAB

For demonstration of results, the two approaches used are:

i. Use of histograms
ii. Use of Grayscale images

Also, the advantages and shortcomings of all the implemented filters are discussed. We have worked with the following filters:

i. Smoothing Spatial Filters (Linear)
ii. Order Statistics Filter
iii. Sharpening Spatial Filters

At last, we have concluded with the practical applications of image processing in today's world.

Digital Image Processing

The field of digital image processing refers to processing digital images by means of a digital computer. A digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term used most widely to donate the elements of a digital image. An image maybe defined as two dimensional function, *f(x,y)*, where *x* and *y* are spatial co-ordinates, and the amplitude of *f* at any pair of co-ordinates *(x,y)* is called the intensity or gray level of the image at that point. When x, y and the amplitude values of f are all finite, discrete quantities, we call the image a Digital Image. Digital image processing encompasses a wide and varied field of applications.

There are no clear cut boundaries in the continuum from image processing at one end to computer vision at the other. However, a useful paradigm is to consider three types of computerized processes in this continuum: low, mid and high level processes.

a) Low Level Processes involve primitive operations, such as pre-processing to reduce noise, contrast enhancement, and image sharpening. A low level process is characterised by the fact that both its input and outputs typically are images.
b) Mid-Level Processes on images involve tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterised by the fact that its input generally are images, but its output are attributes extracted from those images (example edges, contours and the identity of individual objects).
c) High Level Processing involves "making sense" of an ensemble of recognized objects, as an image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with human vision.

A reasonable area of overlap between image processing and image analysis is the upper end of the scale of mid-level processes. Thus, digital image processing encompasses processes whose inputs and outputs are images, in addition, includes processes that extract attributes from images.

Digital Image Representation

An image may be defined as a two-dimensional function *f(x, y)*, where *x* and *y* are spatial

coordinates, and the amplitude of f at any pair of coordinates is called the intensity of the image at that point. The term 'gray level' is used often to refer to the intensity of monochrome images. Color images are formed by a combination of individual images. For example, in the RGB color system a color image consists of three individual monochrome images, referred to as the *red* (R), *green* (G), and *blue* (B) *primary images.* For this reason, many of the techniques developed for monochrome images can be extended to color images by processing the three component images individually.
An image may be continuous with respect to the x- and y- coordinates, and also in amplitude. Converting such an image to digital form requires that the coordinates, as well as the amplitude, be digitized. Digitizing the coordinate values is called quantization. Thus, when x, y, and the amplitude values of *f* are all finite, we call the image a Digital Image.

Coordinate Conventions

The result of sampling and quantization is a matrix of real numbers. We use two principal ways to represent digital images. Here, we assume that an image *f(x, y)* is sampled so that the resulting image has M x N. the values of the coordinates are discrete quantities. For notational clarity and convenience, we use integer values for these discrete coordinates.
The image origin is defined to be at (x, y) = (0,0). The next coordinate values along the first row of the image are (x, y) = (0, 1). The notation (0, 1) is used to signify the second sample along the first row. It does not mean that these are the actual values of physical coordinates when the image was sampled.

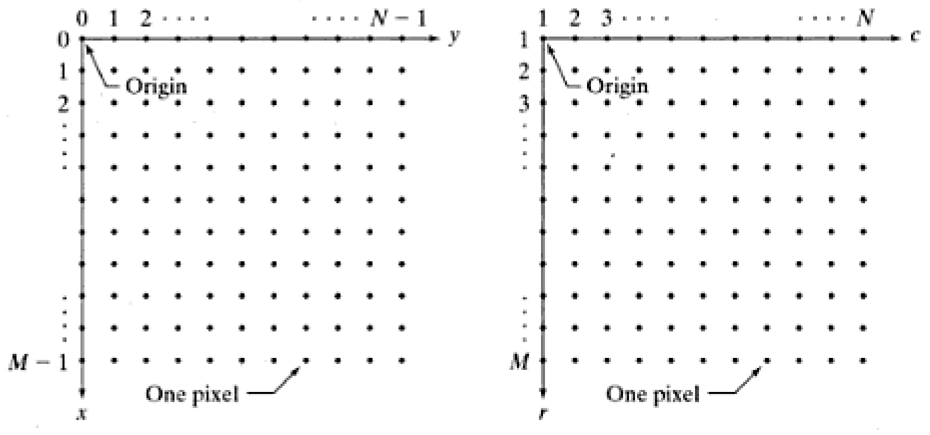


Figure 2: Coordinate Conventions

Here, *x* ranges from 0 to M-1 and *y* from 0 to N-1 in integer increments.

The coordinate convention used in the Image Processing Toolbox to denote arrays is different from the one shown in Figure 1.2 above.

(i) Instead of using (x, y), the toolbox uses the notation (r, c) to indicate rows and columns. However, the order of coordinates is the same as the order discussed above i.e., the forst element of a coordinate tuple, (a, b) , refers to a row and the second is column.

(ii) The origin of the cooedinate system is ar (r, c) = (1, 1); thus, *r* ranges from 1 to M, and *c* from 1 to N.

Image processing Toolbox documentation refers to the coordinates known as pixel coordinates. Less frequently, the toolbox employs another coordinate convention, called spatial coordinates, that uses *x* to refer to columns and *y* to refer to rows. This is opposite of our use of variables *x* and *y*.

The coordinate system in the figure above lead to the following representation for a digitilized iamge:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

The right side of this equation is a digital image by definition. Each element of this array is called an image element, picture element, pixel or pel.

Figure 3: An illustration of loading an image in MATLAB

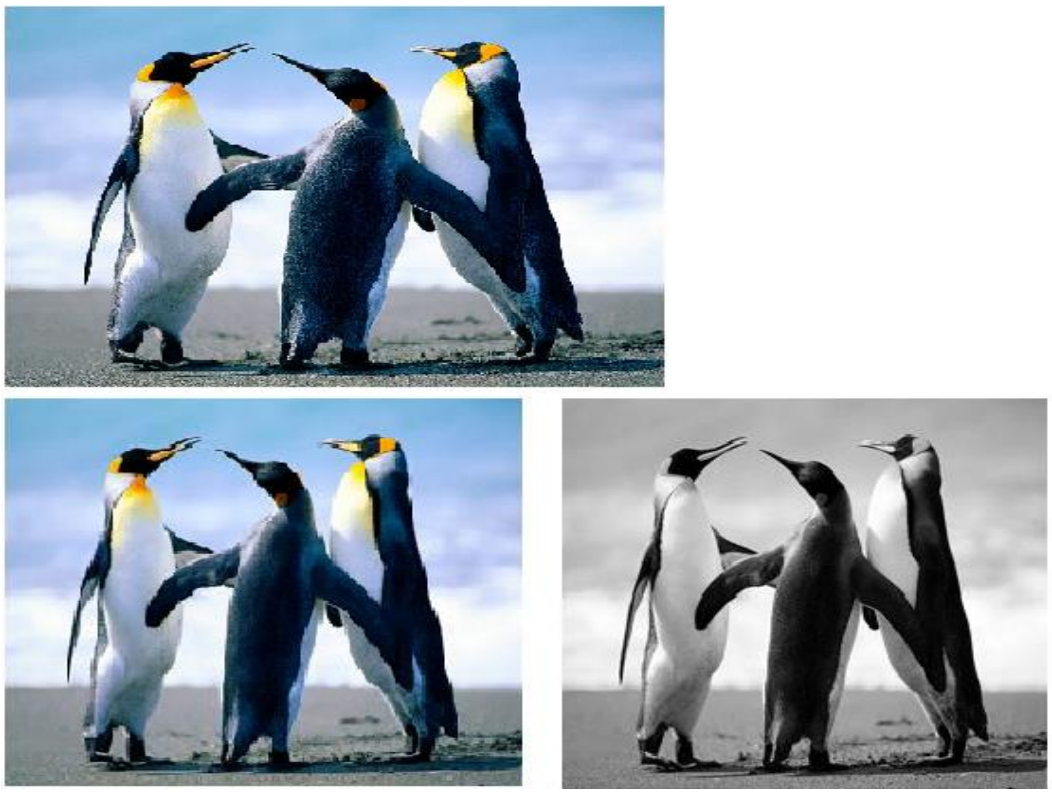


Figure 4: Original image converted into gray scale after compression

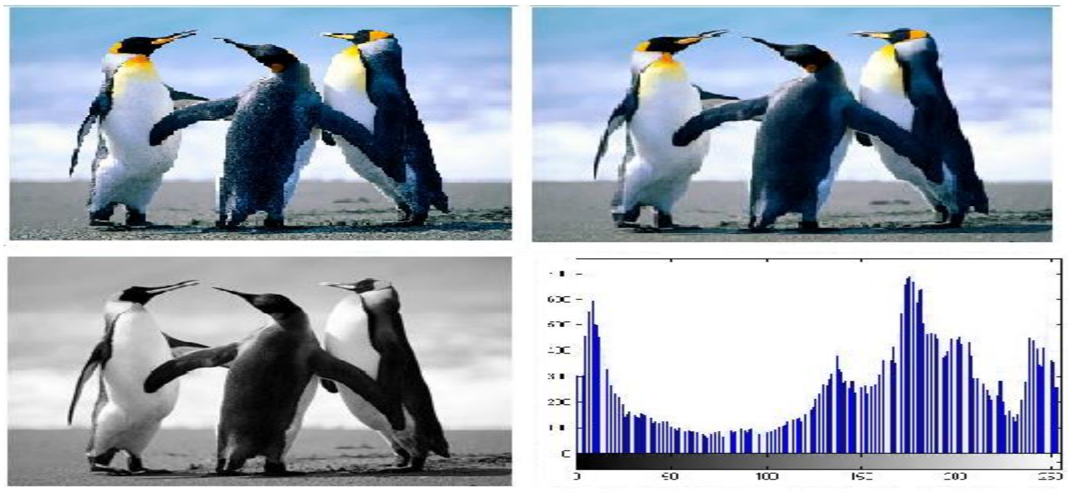


Figure 5: RGB to GRAY conversion with histogram

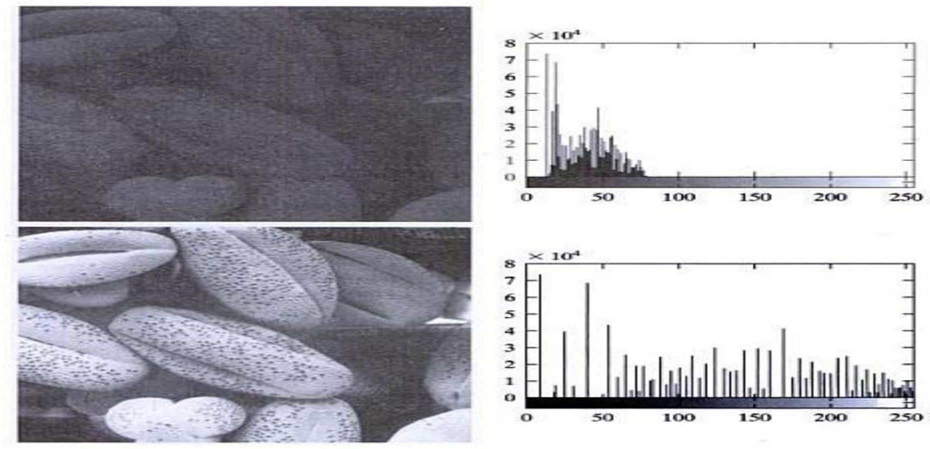


Figure 6: Illustration of histogram equalization

Image Degradation

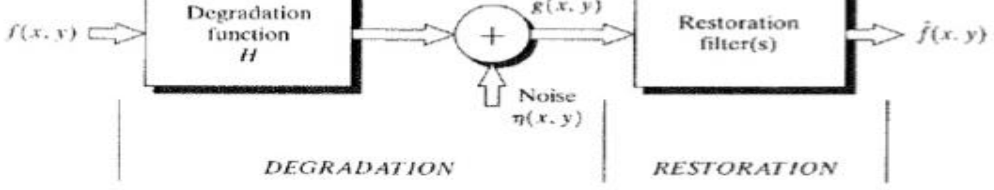


Figure 7: Image Degradation and Restoration Process

The process by which the original image is blurred is usually very complex and often unknown. To simplify the calculations, the degradation is often modelled as a linear function which is often referred as point spread function.

The different causes of image degradation are

- Improper opening and closing of the shutter
- Atmospheric turbulence
- Mis-focus of lens
- Relative motion between camera and object which causes motion blur

Noise Models

Due to limitations of computational and storage capacity, it is practically always impossible to take into account all the available knowledge about minor details which possibly have some impact on the observation. It is then reasonable to ignore the details and model only their net effect, which manifests itself in minor fluctuations not predictable from the things included in the model. These fluctuations are called noise.
For real valued parameters, the standard choice of noise model is the Gaussian distribution. The central limit theorem states, roughly, that if a very large number of very small independent fluctuations are summed linearly, then the distribution of the resulting total fluctuation will be Gaussian. In reality, neither the requirement of all the fluctuations being small nor the linearity of summation are perfectly met, but Gaussian noise models are used anyway because they are mathematically very convenient.

(i) Additive Noise:

One very common model is the additive noise model, where we have our 'true' data vector s[n], (which we are trying to ascertain), being corrupted by a noise vector, v[n]. What we are given, is x[n], where:

$$x[n]=s[n]+v[n]$$

This is called the 'additive' noise model, because as you can see, noise is *added* to our true signal, giving us x[n]. There are many ways in which we can remove this noise in an additive model, such as filtering, (which is a form of averaging if you like). This is a very common type of noise model. If I am talking, and you are talking over me, your voice can be modelled as additive to my voice. You voice would be additive 'noise', that is added to my voice, which in this vain example is the 'true' signal, (although this would be disputed in a heated argument between two people). In a more objective example, thermal noise from a microphone's electronics can also be modelled as an additive noise, which is added to a voice signal that it has received. Many things can be modelled as additive types of noises.

(ii) Multiplicative Noise:

Multiplicative noise on the other hand is still a model, but in this model our true data samples are being *multiplied* by noise samples, like so:

$$x[n]=s[n]v[n]$$

One common way to remove multiplicative noise is to actually transform it into an additive model, and then apply everything we know from the additive noise reduction field. We can do this easily via taking the logarithm of the signal, filtering, then inverse log transform. Thus we can do:

$$x[n]=s[n]v[n]\implies\log(x[n])=\log(s[n]v[n])=\log(s[n])+\log(v[n])$$

At this point we have an additive model once more. Now, we can filter as we normally might, to remove or reduce log(v[n]), and then simply take $\log^{-1}$ of the result, yielding an estimate of s[n], our true signal.One example of multiplicative noise is in illumination differences between images, which is solved in the above way. The non-uniform illumination across an image can be modelled as a pixel-by-pixel multiplication of the image by an illumination mask. This is also known as homomorphic filtering by the way. Anytime you can model a corrupting phenomenon as *multiplying* your clean data signal, you can use this multiplicative model.

(iii) Impulse Noise:

Impulse noise has the property of either leaving a pixel unmodified with the probability 1-p, or replacing it together with a probability p. Restricting η[m,n] to produce only the extreme intensities 0 or 1 results in salt and pepper noise. The source of impulse noise is usually the result of an error in transmission or an atmospheric o man-made disturbance. The MATLAB code that adds salt-and-pepper noise to the input image and then performs the averaging operation on the noisy image.

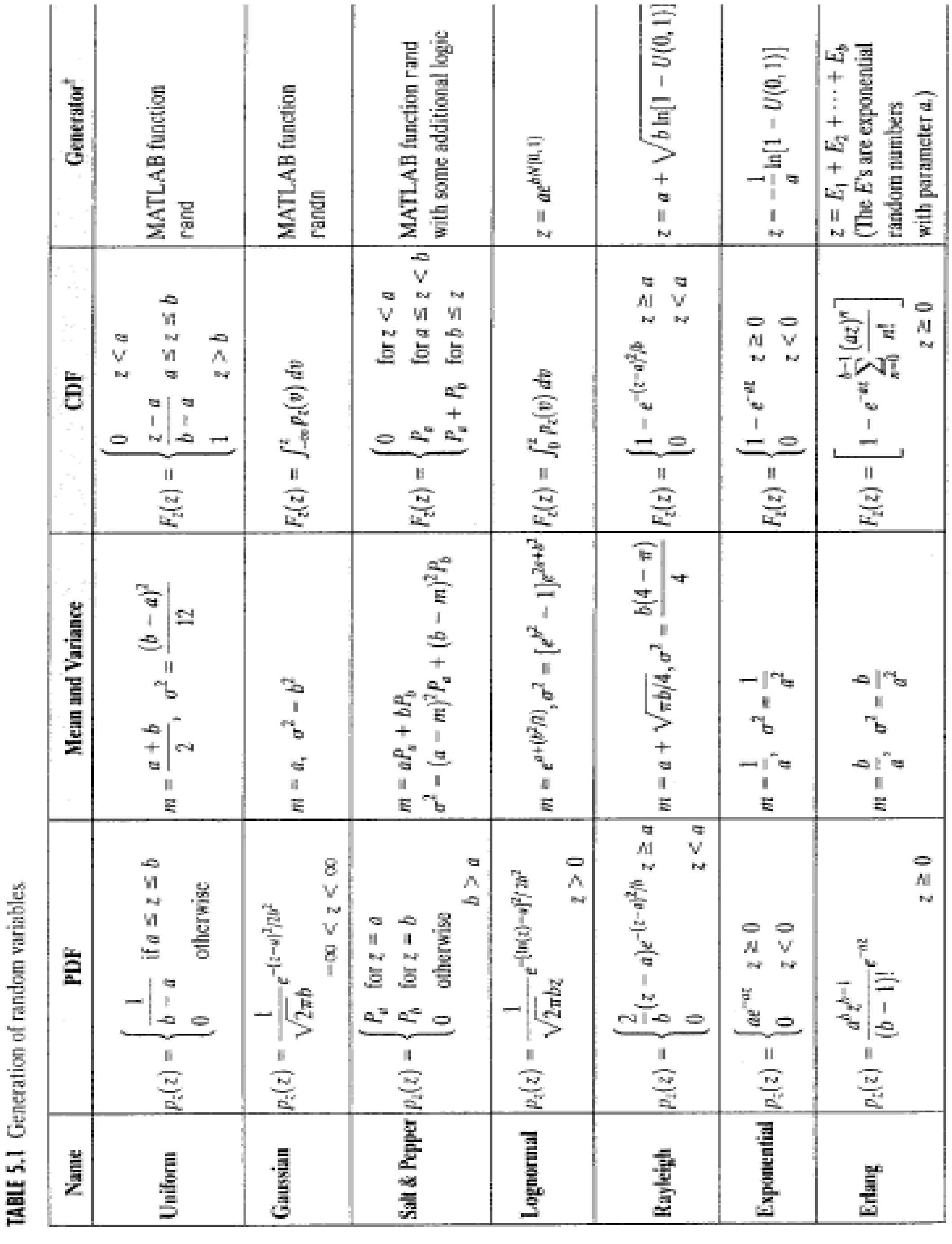


TABLE 5.1 Generation of random variables

Name	PDF	Mean and Variance	CDF	Generator[†]
Uniform	$p_z(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise} \end{cases}$	$m = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$	$F_z(z) = \begin{cases} 0 & z < a \\ \frac{z-a}{b-a} & a \le z \le b \\ 1 & z > b \end{cases}$	MATLAB function rand
Gaussian	$p_z(z) = \frac{1}{\sqrt{2\pi}b} e^{-(z-a)^2/2b^2}$ $-\infty < z < \infty$	$m = a,\ \sigma^2 = b^2$	$F_z(z) = \int_{-\infty}^{z} p_z(v)\,dv$	MATLAB function randn
Salt & Pepper	$p_z(z) = \begin{cases} P_a & \text{for } z = a \\ P_b & \text{for } z = b \\ 0 & \text{otherwise} \end{cases}$ $b > a$	$m = aP_a + bP_b$ $\sigma^2 = (a-m)^2 P_a + (b-m)^2 P_b$	$F_z(z) = \begin{cases} 0 & \text{for } z < a \\ P_a & \text{for } a \le z < b \\ P_a + P_b & \text{for } b \le z \end{cases}$	MATLAB function rand with some additional logic
Lognormal	$p_z(z) = \frac{1}{\sqrt{2\pi}bz} e^{-[\ln(z)-a]^2/2b^2}$ $z > 0$	$m = e^{a+(b^2/2)},\ \sigma^2 = [e^{b^2} - 1]e^{2a+b^2}$	$F_z(z) = \int_{0}^{z} p_z(v)\,dv$	$z = ae^{bN(0,1)}$
Rayleigh	$p_z(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$m = a + \sqrt{\pi b/4},\ \sigma^2 = \frac{b(4-\pi)}{4}$	$F_z(z) = \begin{cases} 1 - e^{-(z-a)^2/b} & z \ge a \\ 0 & z < a \end{cases}$	$z = a + \sqrt{b\ln[1 - U(0,1)]}$
Exponential	$p_z(z) = \begin{cases} ae^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$m = \frac{1}{a},\ \sigma^2 = \frac{1}{a^2}$	$F_z(z) = \begin{cases} 1 - e^{-az} & z \ge 0 \\ 0 & z < 0 \end{cases}$	$z = -\frac{1}{a}\ln[1 - U(0,1)]$
Erlang	$p_z(z) = \frac{a^b z^{b-1}}{(b-1)!} e^{-az}$ $z \ge 0$	$m = \frac{b}{a},\ \sigma^2 = \frac{b}{a^2}$	$F_z(z) = \left[1 - e^{-az}\sum_{n=0}^{b-1}\frac{(az)^n}{n!}\right]$ $z \ge 0$	$z = E_1 + E_2 + \cdots + E_b$ (The E's are exponential random numbers with parameter a.)

Figure 8: Generation of random variables

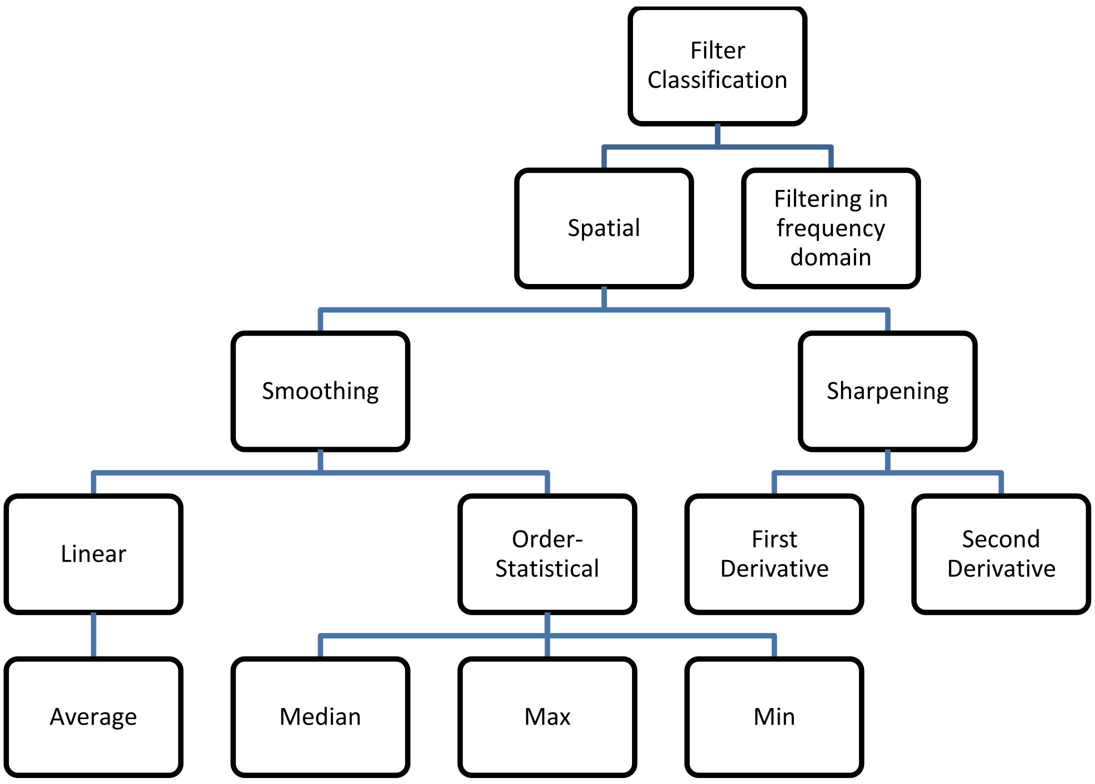
Filters

A filter is a device or process that removes from a signal some unwanted component or feature. Filtering is a class of signal processing, the defining feature of filters being the complete or partial suppression of some aspect of the signal. Most often, this means removing some frequencies and not others in order to suppress interfering signals and reduce background noise. However, filters do not exclusively act in the frequency domain; especially in the field of image processing many other targets for filtering exist. Correlations can be removed for certain frequency components and not for others without having to act in the frequency domain.

Image Denoising

Image denoising is an important image processing task, both as a process itself, and as a component in other processes. Very many ways to denoise an image or a set of data exists. The main properties of a good image denoising model are that it will remove noise while preserving edges. Traditionally, linear models have been used. One common approach is to use a Gaussian filter, or equivalently solving the heat-equation with the noisy image as input-data, i.e. a linear, 2nd order PDE-model. For some purposes this kind of denoising is adequate. One big advantage of linear noise removal models is the speed. But a backdraw of the linear models is that they are not able to preserve edges in a good manner: edges, which are recognized as discontinuities in the image, are smeared out. Nonlinear models on the other hand can handle edges in a much better way than linear models can.

Classification of Filters



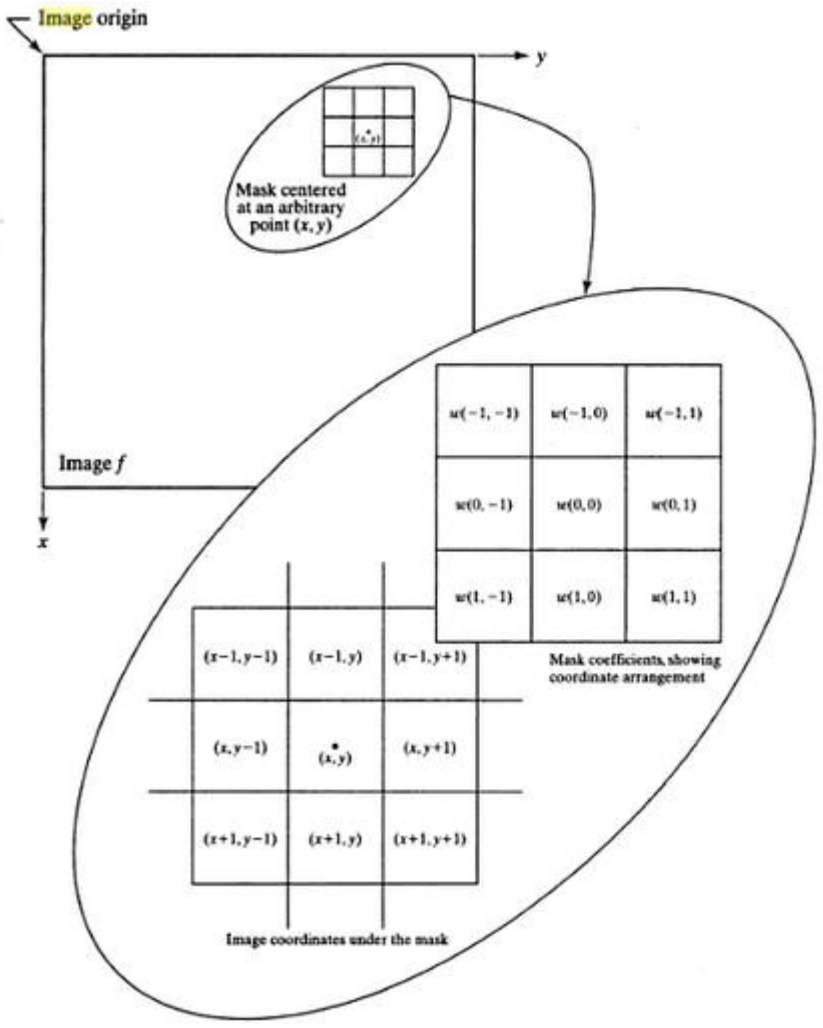


Figure 9: Mechanics of Linear Spatial Filtering

Smoothing Spatial Filtering

Smoothing filters are used for blurring and for noise reduction. Blurring is used in pre-processing tasks, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

Smoothing Linear Filter

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighbourhood of the filter mask. These filters sometimes are called averaging filters. They are also referred to as lowpass filters.

The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the intensity levels in the neighbourhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in intensities. Because random noise typically consists of sharp transitions in intensity levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable feature of an image)also are characterized by sharp intensity transitions, so averaging filters have the undesirable side effect that they blur edges. Another application of this type of process include the smoothing of false contours that result from using an insufficient number of intensity levels. A major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant" we mean pixel regions that are small with respect to the size of filter mask.

Fig shows two 3*3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask. This can best be seen by substituting the coefficients of the mask into eq

$$R = \frac{1}{9}\sum_{i=1}^{9} zi$$

Which is the average of the intensity levels of the pixels in the 3*3neighbourhood defined by the mask. Note that, instead of being $1/_9$, the coefficients of the filter are all 1s. The idea here is that it is computationally more efficient to have coefficients valued 1. At the end of the filtering process the entire image is divided by 9. An *m*n* mask would have a normalizing constant equal to $1/_{mn}$. A spatial averaging filter in which all coefficients are equal sometimes is called a *box filter*.

The second mask yields a so called weighted average, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown in fig the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask. The diagonal terms are further away from the center than the orthogonal neighbors (by a factor of $\sqrt{2}$) and, thus, are weighted less than the immediate neighbors of the center pixel. The basic strategy behind weighing the center point the highest and then reducing the value of the coefficients as a function of increasing distance from the origin is simply an attempt to reduce blurring in the smoothing process. However, the sum of all the coefficients in the mask is equal to 16, an attractive feature for computer implementation because it is an integer power of 2.

$$\frac{1}{9}\times\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\times\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Figure 10: Filter Masks

An important application of spatial averaging is to blur an image for the purpose of getting a gross representation of objects of interest, such that the intensity of smaller objects blends with the background and larger objects become “blob like” and easy to detect. The size of the mask establishes the relative size of the objects that will be blended with the background. Fig , which is an image from the hubble telescope in orbit around the Earth. Fig shows the result of applying a 15*15 averaging mask to this image.

As we can see from the figure above, by increasing the window size from 3 to 5 to 7 in an average filter, its noise is reduced. However, the image starts to blur and the edges are smoothened. A number of objects have either blended with the background or their intensity has diminished considerably. It is typical to follow an operation like this with thresholding to eliminate objects based on their intensity. The result of using the thresholding function with a threshold value equal to 25% of the highest intensity in the blurred image shown.

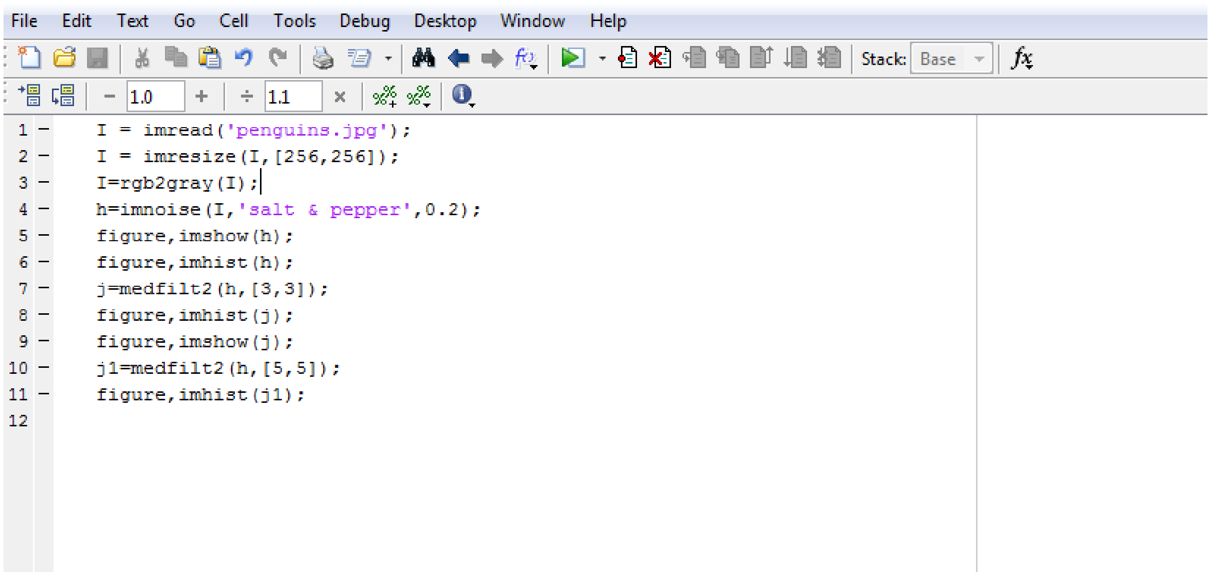
Order Statistic Filters

Order statistic filters are non-linear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the centre pixel with the value determined by the ranking result.

Median Filter

Median filter, as its name implies, replaces the value of a pixel by the median of the intensity values in the neighbourhood of that pixel (the original value of that pixel is included in the computation of the median). Median filters are quite popular because, for certain types of random noise, they provide excellent noise reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of impulse noise, also called *salt-and-pepper noise* because of its appearance as white and black dots superimposed on an image.

The median, ε, of a set of values is such that half the values in the set are less than or equal to ε, and half are greater than or equal to ε. In order to perform median filtering at a point in an image, we first sort the values of the pixel in the neighborhood, determine their median, and assign that value to the corresponding pixel in the filtered image. For example, in a 3*3 neighbourhood the median is the 5th largest value, in a 5*5 neighbourhood it is the 13th largest value, and so on. When several values in a neighbourhood are the same, all equal values are grouped. For example, suppose that a 3*3 neighbourhood has values (10, 20, 20, 20, 15, 20, 20, 25, 100). These values are sorted as (10, 15, 20, 20, 20, 20, 20, 25, 100), which results in a median of 20. Thus, the principal function of median filters is to force points with distinct intensity levels to be more like their neighbours. In fact, isolated clusters of pixels that are light or dark with respect to their neighbours, and whose area is less than m^2 /2 (one half of the filter area), are eliminated by an m*m median filter. In this case “ eliminated” means forced to the median intensity of the neighbours. Larger clusters are affected considerably less.



```
I = imread('penguins.jpg');
I = imresize(I,[256,256]);
I=rgb2gray(I);
h=imnoise(I,'salt & pepper',0.2);
figure,imshow(h);
figure,imhist(h);
j=medfilt2(h,[3,3]);
figure,imhist(j);
figure,imshow(j);
j1=medfilt2(h,[5,5]);
figure,imhist(j1);
```

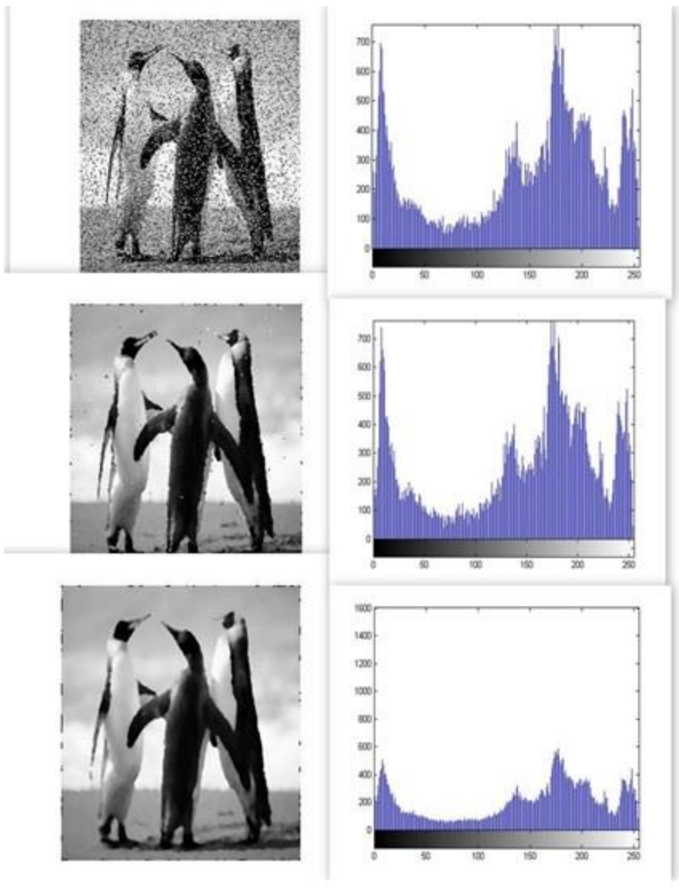
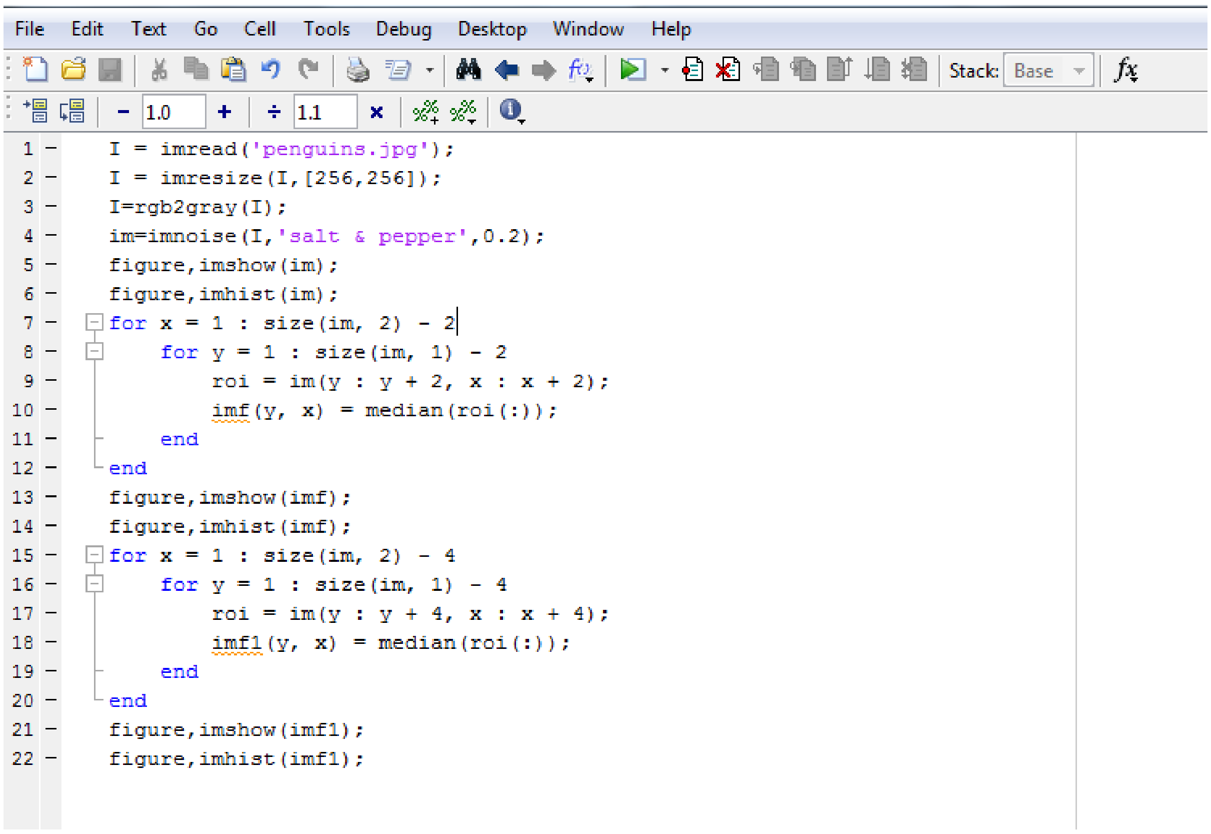


Figure 11: In-built Median Filter



```
I = imread('penguins.jpg');
I = imresize(I,[256,256]);
I=rgb2gray(I);
im=imnoise(I,'salt & pepper',0.2);
figure,imshow(im);
figure,imhist(im);
for x = 1 : size(im, 2) - 2
    for y = 1 : size(im, 1) - 2
        roi = im(y : y + 2, x : x + 2);
        imf(y, x) = median(roi(:));
    end
end
figure,imshow(imf);
figure,imhist(imf);
for x = 1 : size(im, 2) - 4
    for y = 1 : size(im, 1) - 4
        roi = im(y : y + 4, x : x + 4);
        imf1(y, x) = median(roi(:));
    end
end
figure,imshow(imf1);
figure,imhist(imf1);
```

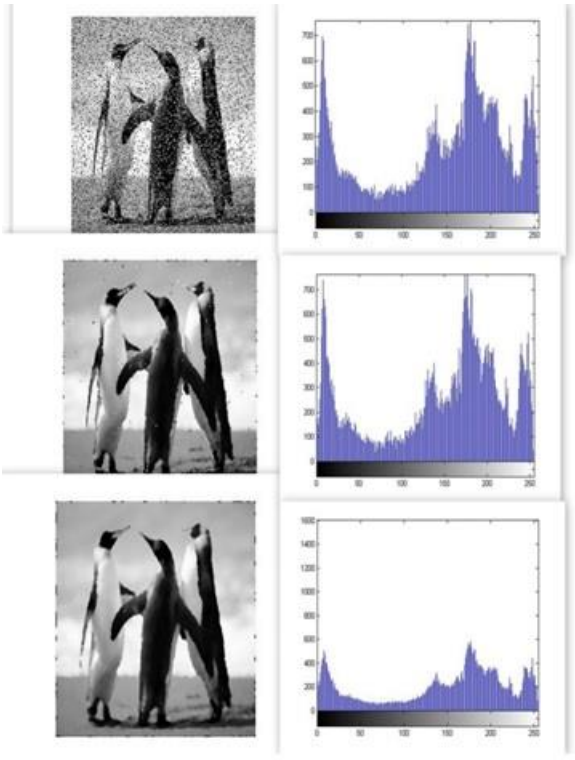
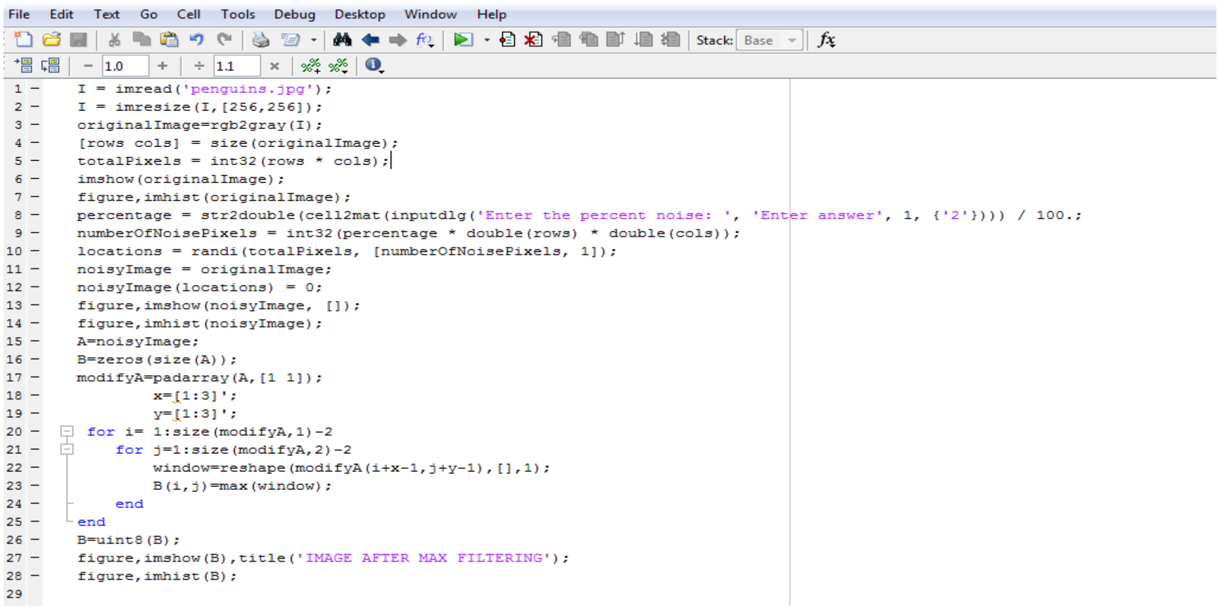


Figure 12: User Defined Median Filter

Max Filter

The filter is almost same as median filter. The only difference between the 2 is the value with which some pixel values of corrupted image are replaced. The value here is the max value out of 256 values in the range 0 to 256. This kind of filter is useful mainly in correcting pepper noise. Since in pepper noise the image is corrupted by some random patches of zero so, using n max filter almost denoises an image.



```
I = imread('penguins.jpg');
I = imresize(I,[256,256]);
originalImage=rgb2gray(I);
[rows cols] = size(originalImage);
totalPixels = int32(rows * cols);
imshow(originalImage);
figure,imhist(originalImage);
percentage = str2double(cell2mat(inputdlg('Enter the percent noise: ', 'Enter answer', 1, {'2'}))) / 100.;
numberOfNoisePixels = int32(percentage * double(rows) * double(cols));
locations = randi(totalPixels, [numberOfNoisePixels, 1]);
noisyImage = originalImage;
noisyImage(locations) = 0;
figure,imshow(noisyImage, []);
figure,imhist(noisyImage);
A=noisyImage;
B=zeros(size(A));
modifyA=padarray(A,[1 1]);
        x=[1:3]';
        y=[1:3]';
 for i= 1:size(modifyA,1)-2
    for j=1:size(modifyA,2)-2
        window=reshape(modifyA(i+x-1,j+y-1),[],1);
        B(i,j)=max(window);
    end
end
B=uint8(B);
figure,imshow(B),title('IMAGE AFTER MAX FILTERING');
figure,imhist(B);
```

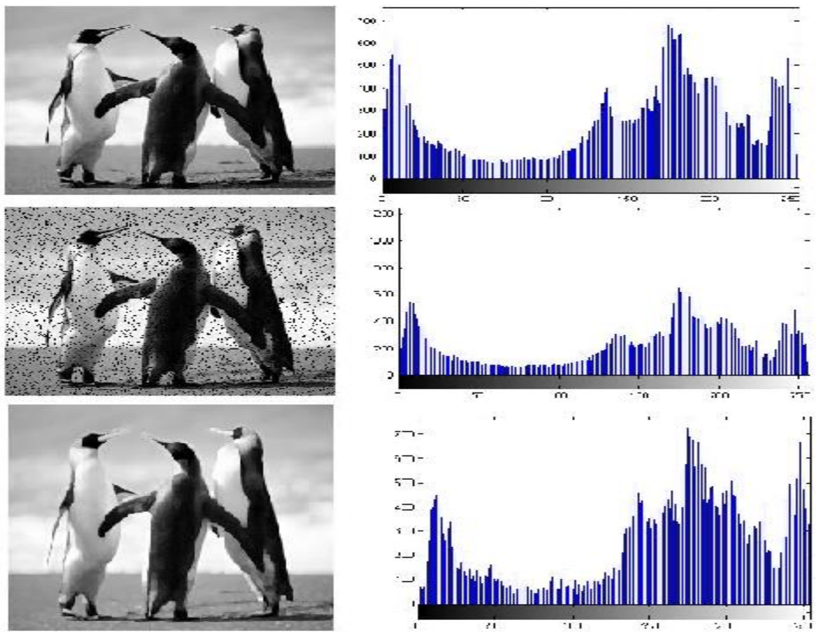
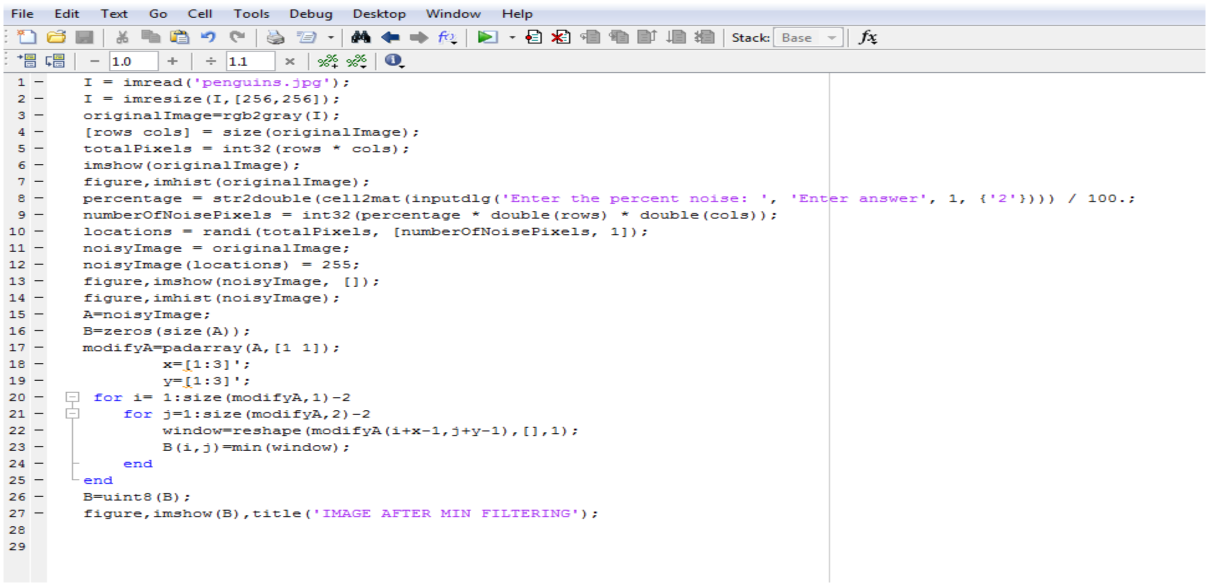


Figure 13: User Defined Max Filter

Min Filter

The filter is almost same as median filter. The only difference between the 2 is the value with which some pixel values of corrupted image are replaced. The value here is the min value out of 256 values in the range 0 to 256. This kind of filter is useful mainly in correcting salt noise. Since in salt noise the image is corrupted by some random patches of one so, using n min filter almost denoises an image.



```
I = imread('penguins.jpg');
I = imresize(I,[256,256]);
originalImage=rgb2gray(I);
[rows cols] = size(originalImage);
totalPixels = int32(rows * cols);
imshow(originalImage);
figure,imhist(originalImage);
percentage = str2double(cell2mat(inputdlg('Enter the percent noise: ', 'Enter answer', 1, {'2'}))) / 100.;
numberOfNoisePixels = int32(percentage * double(rows) * double(cols));
locations = randi(totalPixels, [numberOfNoisePixels, 1]);
noisyImage = originalImage;
noisyImage(locations) = 255;
figure,imshow(noisyImage, []);
figure,imhist(noisyImage);
A=noisyImage;
B=zeros(size(A));
modifyA=padarray(A,[1 1]);
        x=[1:3]';
        y=[1:3]';
 for i= 1:size(modifyA,1)-2
    for j=1:size(modifyA,2)-2
        window=reshape(modifyA(i+x-1,j+y-1),[],1);
        B(i,j)=min(window);
    end
end
B=uint8(B);
figure,imshow(B),title('IMAGE AFTER MIN FILTERING');
```

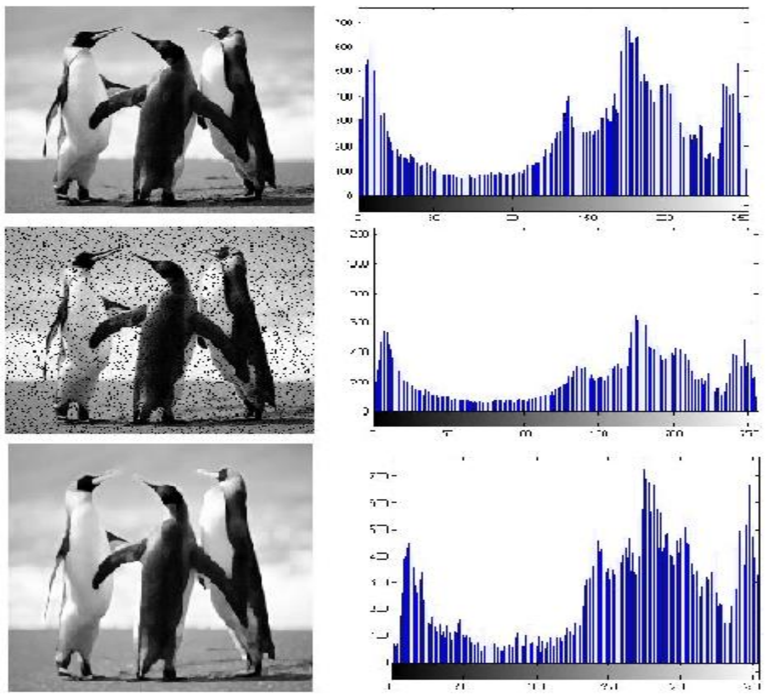


Figure 14: User Defined Min Filter

Sharpening Filter

The principal objective of sharpening is to highlight transition in intensity. Uses of image sharpening vary and include applications ranging from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems. Image blurring could be accomplished in the spatial domain by pixel averaging because averaging is analogous to integration, sharpening can be accomplished by spatial differentiation. The strength of the response of a derivative operator is proportional to the degree of intensity discontinuity of the image at the point at which the operator is applied.

Sharpening filters are based on first and second order derivatives. The derivatives of a digital function are defined in terms of differences. Any definition we use for a *first derivative*

(i) Must be zero in areas of constant intensity
(ii) Must be nonzero at the onset of an intensity step or ramp
(iii) Must be nonzero along ramps.

Any definition of second derivative

(i) Must be zero in constant areas
(ii) Must be nonzero at the onset and end of a gray-level step or ramp
(iii) Must be zero along ramps of constant slope.
Since we are dealing with digital quantities whose values are finite, the maximum possible gray-level change also is finite, and the shortest distance over which that change can occur is between adjacent pixels.

A basic definition of the first-order derivative of a one-dimensional function f(x) is the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

A partial derivative is used here to keep the notation same as when we consider an image function of two variables, f(x, y), at which time we will be dealing with partial derivatives along the two spatial axes. Similarly, we define a second-order derivative as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

It is easily verified that these two definitions satisfy the conditions stated previously regarding derivatives of the first and second order.

Conclusion

There are many factors due to which noise can corrupt an image. The various factors can be lens misfocus, atmospheric turbulence etc. depending upon the way in which noise is introduced into the image and the way the image is altered noise can be classified in many ways. One very popular noise is salt and pepper noise. An image containing salt-and-pepper noise will have dark pixels in bright regions and bright pixels in dark regions. This type of noise can be caused by analog-to-digital converter errors, bit errors in transmission, etc. Depending upon the noise type we have to pick up filter of appropriate kind which can remove that particular noise. In case of salt and pepper noise many filters can be used. Each filter has its own advantages and shortcomings. The most popular filter used is in averaging filter. It is the simplest of all the filters from designing point of view. But its inherent disadvantage is that if the amount of noise is high then using a larger window size averaging filter causes blurring. Blurring here refers to unsharp edges in image. The same is the case with a median filter which is somewhat better than averaging filter in designing but complex in design. But then also both of these filters have an advantage due to which they are into use till now. An important application of spatial averaging is to blur an image for the purpose of getting a gross representation of objects of interest, such that the intensity of smaller objects blends with the background and larger objects become “blob like” and easy to detect. There are instances where only salt type or pepper type noises are introduced in an image. Both the types can be reduced to 100% by using min filter and max filter respectively. The results obtained with both these filters are approximately 100%. There is also one more type of filter Laplacian filter. The main use of this filter is to detect edges. We mentioned that in case of averaging filter the edges are not defined sharply as we increase our window size to eliminate more noise. But if we incorporate a Laplace filter along with averaging filter then the results are very good.

The averaging filter reduces all noise and Laplace filter defines the edge sharply and hence de-blurring is reduced.

References

[1] Digital Image Processing (2e) Using MATLAB by Rafael C. Gonzalez, Richard Eugene Woods, Steven L. Eddins.
[2] Digital Image Processing by Rafael C Gonzalez
[3] Digital Image Processing By Jayaraman
[4] www.mathworks.com/help/matlab/
[5] www.imageprocessingplace.com/