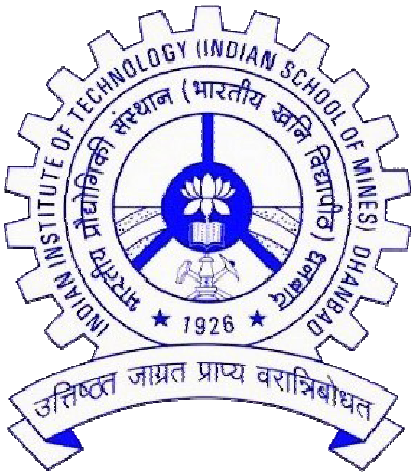


Computer Science & Engineering Department

Project on

Image Processing

7 November

Rajnish Kumar Ranjan(15JE001619)

Dual Degree Computer Science &Engineering

Indian Institute of Technology (Indian School of Mines), Dhanbad-826004

Session 2016-2017

CONTENTS

Acknowledgement

I would like to express the deepest appreciation to my project mentor Mr. Rajendra Pamula, who has the attributes. He continually and convincingly conveyed a spirit of adventure in regard to the project, and an excitement while guiding. Finally, I must say that no height is ever achieved without some sacrifices made at some end and it is here where I owe our special debt to our parents and our friends for showing their generous love and care throughout the entire period of time.

November 7

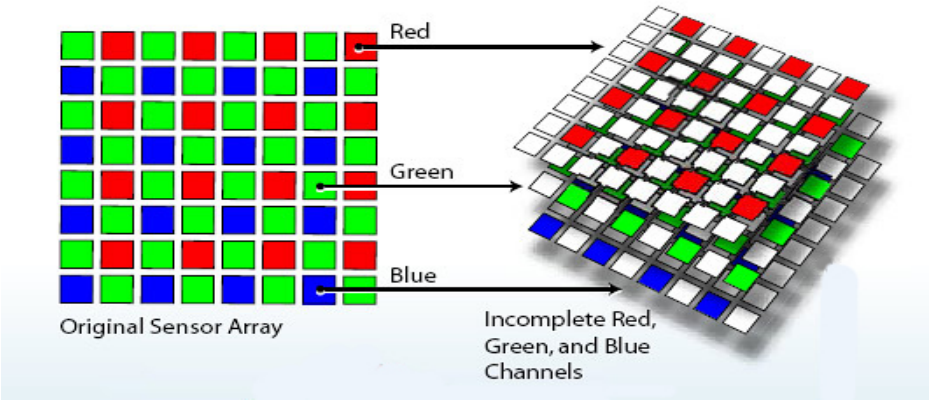
Rajnish Kumar Ranjan

1. Introduction

Processing visual information by computer has been drawing a very significant attention of the researchers over the last few decades. In imaging science, Image Processing is processing of images using mathematical operations by using any form of signal processing for which the input is an image, a series of images, or a video, such as a photograph or video frame. Image processing usually refers to digital image processing, but optical and analog image processing also are possible. The process of receiving and analyzing visual information by digital computer is known by digital image processing. Processing of an image includes improvement in its appearance and efficient representation. So the field consists of not only feature extraction, analysis and recognition of images, but also coding, filtering, enhancement and restoration.

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 (x, y) is called the intensity or gray level of the image at that point. When x, y and the amplitude value of f are all finite, discrete quantities, we call the image a digital image. A digital image is composed of finite number of elements, each of which has a particular location and value. These elements are referred as picture elements and *pixels*.

A multispectral image is a vector-valued function having number of components equal to that of spectral bands and is represented by [g1(x, y), g2(x, y), g3(x, y), ….. , gn(x, y)] at each (x, y). The color image is a special case of multispectral image; and if we consider intensity measured in three wavelengths corresponding to red, green and blue, then g(x, y) = [gR(x, y), gG(x, y), gB(x, y)]. An RGB image (having red, green and blue spectral intensities) may be viewed as a “stack” of three gray-scale images that, when fed into the red, green and blue inputs of a color monitor, produce a color image on the screen.



1.1 Different stages of image processing

The entire process of image processing and analysis starting from the receiving visual information to the giving out of description of the scene, may be divided into three major stages which are also considered as major sub-areas, and are given below:

(i) *Discretization and representation*: converting visual information into discrete form, which is suitable for computer processing; approximating visual information to save storage space as well as time requirement in subsequent processing.

(ii) *Sampling and quantization*: The sampling rate determines the spatial resolution of the digitized image, while the quantization level determines the number of grey levels in the digitized image. A magnitude of the sampled image is expressed as a digital value in image processing. The transition between continuous values of the image function and its digital equivalent is called quantization.

(iii) *Processing:* improving image quality by filtering etc, or compressing data to save storage and channel capacity during transmission etc.

1.2 Image I/O and Display

Images are read into the MATLAB environment using function `imread`. Images are displayed on the MATLAB desktop using function `imshow`. They have the basic syntax:

imread('filename') imshow(f)

Images are written to the Current Directory using function `imwrite,` which has the following syntax:

Imwrite(f, 'filename')

Variable f(an array or likewise) is written as an image known by '*filename*'.

1.3 Classes and Image Types

There are various classes supported by MATLAB and the Image Processing Toolbox for representing pixel values. Some of them are:

(i) uint8 – Unsigned 8-bit integers in the range [0,255](1 byte per element).

(ii) double-Double precision floating point numbers in the approximate range [-10^{308} 10^{308}]

(iii) logical- Values are 0 or 1(1 bit per element).

Gray-scale, Binary, Indexed, RGB are the four types of images supported by toolbox.

1.4 Applications

Digital image processing techniques have variety of applications. The following are a few major application areas:

1. *Industrial automation:* automation inspection system, non-destructive testing, automatic assembling, process related to VLSI manufacturing, robotics etc.
2. *Bio medical:* ECG, EEG, EMG analysis, X-ray image analysis cytological, histological and stereological applications, mass screening of medical images such as chromosome slides for detection of various diseases etc.
3. *Remote Sensing:* natural resources survey and management, estimation related to agriculture, hydrology, forestry, mineralogy, urban planning, environment and pollution control etc.
4. *Scientific applications:* bubble chamber and other forms of track analysis.
5. *Criminology:* finger print identification, human face registration and matching, forensic investigation etc.
6. *Office automation:* optical character recognition, document processing, cursive script recognition, identification of address area of envelope etc.
7. *Astronomy and space applications:* restoration of image suffering from geometric and photometric distortions, computing close-up picture of planetary surfaces etc.
8. *Security measure*

2. Intensity Transformation and Spatial Filtering

Intensity transformation and spatial filtering are two important categories of spatial domain processing.

Spatial domain processes are denoted by the expression

$$g(x, y) = T[f(x, y)]$$

where f(x, y) is the input image, g(x, y) is the output image, and T is an operator on f defined over a specified neighborhood about point (x, y).

In case of intensity transformation, output value only depends on the intensity value at a point, and not on a neighborhood of points, intensity transformation functions frequently are written in simplified form as

$$s = T(r)$$

where r denotes the intensity of f and s the intensity of g, both at the same coordinates (x, y) in the images. Inbuilt function imadjust is well known in the matlab for intensity transformation.

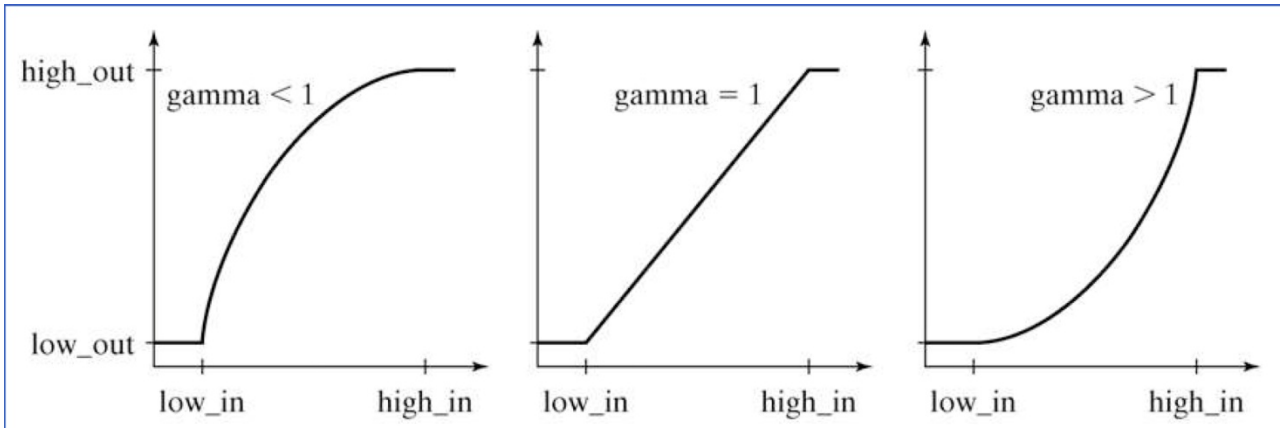
2.1 Function imadjust and stretchlim

2.1.1Function imadjust

It has the general syntax

g = imadgust (f, [low_in high_in], [low_out high_out], gamma)

It maps the intensity values in image f to new values in g, such that values between low_in and high_in map to values between low_out and high_out. Parameter gamma specifies the shape of the curve that maps the intensity values to create g.



Mapping the values between low_out and high_out depending on the value of gamma

2.1.2 Implementing imadjust in .m file

Let r be the input pixel and s be the output pixel at a coordinate (x, y) in an image

From the above graph we notice that

(s-low_out)/(high_out-low_out) = ((r-low_in) / (high_in-low_in))^{gamma}

Using this formula our matlab function is like

```
function f=imadj(li,hi,lo,ho,g)
s=imread('in.gif');
 a=im2double(s);
[r c]=size(a);
f=zeros(r,c);
for x=1:r
    for y=1:c
        f(x,y)=a(x,y);
        if(a(x,y)>li&&a(x,y)<hi)
            f(x,y)=lo+(ho-lo)*((a(x,y)-li)/(hi-li))^g;
        end
    end
end
end
```

where,

li=low_in hi=high_in lo=low_out ho=high_out

In run command of matlab

>> f=imadj(0,1,1,0,g);

>>figure; imshow(a); imshow(f);

Before adjustment

After applying adjustment from [0 1] to [1 0]

The same process is being done if we run the command

>>f=imadjust(a,[0 1],[1 0],1);

This type of processing is useful for highlighting an intensity band of interest

2.1.3 Function stretchlim

Sometime it is of interest to be able to use function `imadjust` "automatically" without having to be concerned about the low and high parameters discussed above. Function `strechlim` is useful in this regard.

Low_High = stretchlim(f)

Where Low_High is a two element vector of a lower and upper limit that can be used to achieve contrast stretching.

2.1.3 Image Histogram

An **image histogram** is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer will be able to judge the entire tonal distribution at a glance.

For an 8-bit gray-scale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those gray-scale values.

2.1.4 Implement stretchlim in .m file

For contrast stretching manually, we take a view on histogram and the tones, where the frequency is found to be low are denied, while those with high intensities are tried to adjust to different intensity level (Gray level).

HISTOGRAM EQUALISATION

Gray level	nk	PDF	CDF	(L-1)*CDF	ROUND OFF
0	790	0.19	0.19	1.33	1
1	1023	0.25	0.44	3.08	3
2	850	0.21	0.65	4.55	5
3	656	0.16	0.81	5.67	6
4	329	0.08	0.89	6.23	6
5	245	0.06	0.95	6.65	7
6	122	0.03	0.98	6.86	7
7	81	0.0.2	1	7	7
	4096				

Here, nk=number of pixels for a definite Gray level PDF=probability density function

CDF=cumulative density function L=Total values of Gray level

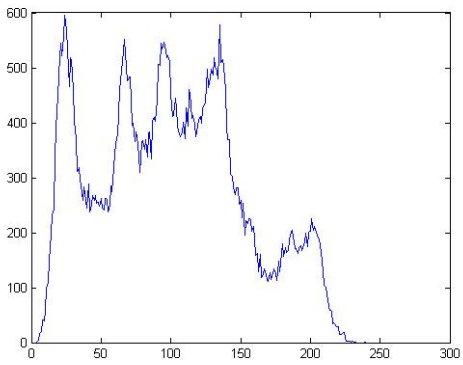
PDF=nk/n

Where, n = total number of pixels in the image (herein graph, n =4096) Finally, rounding off (L-1)*CDF gives output Gray level based on the frequency values distribution for various Gray level.

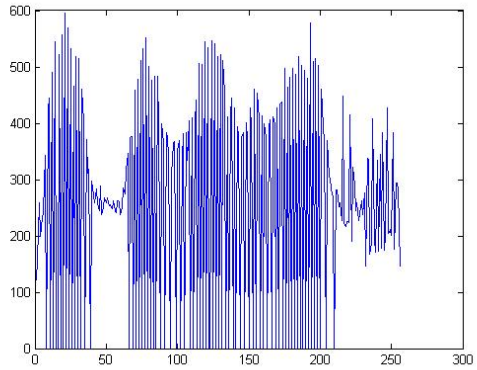
```
g=imread('in.gif');
[r c]=size(g);
ai=uint8(zeros(r,c));
n=r*c;
sum=0;
nk=zeros(256);
pdf=zeros(256);
cum=zeros(256);
cdf=zeros(256);
out=zeros(256);
for i=1:r
   for j=1:c
      nk(g(i,j)+1)=nk(g(i,j)+1)+1;
   end
end
for i=1:256
   sum=sum+nk(i); pdf(i)=nk(i)/n;
   cum(i)=sum;
   cdf(i)=cum(i)/n;
   out(i)=round(cdf(i)*255);
end
for i=1:r
   for j=1:c
      ai(i,j)=out(g(i,j)+1);
   end
end
figure; imshow(g), imshow(ai);
```

Image before

Image after contrast stretching



Input image histogram



Output image histogram

Same output will be shown with command

>>ai=imadjust(g,imstretch(g),[0 1]); imshow(ai);

2.2 Spatial Filtering

Filtering techniques are an important part of image processing systems, in particular when it comes to image enhancement and restoration. Here, we only consider linear and spatially invariant systems. We demonstrate that image *g(x, y)* resulting from the passing of image *f(x, y)* through such a system can be computed using a 2D convolution product with the system impulse response *h(x, y)*.

Spatial filtering can be ***Linear*** or ***Non-linear***.

Convolution is a linear spatial filtering technique.

2.2.1 Convolution

In convolution, the calculation performed at a pixel is a weighted sum of grey levels from a neighborhood surrounding a pixel. Grey levels taken from the neighborhood are weighted by coefficients that come from a matrix or convolution kernel. The kernel's dimensions define the size of the neighborhood in which calculation take place. The most common dimension is *3 x 3*. During convolution, each kernel coefficient is taken in turn and it is multiplied by a value from the neighborhood of the image lying under the kernel. The kernel is applied to the image in such a way that the value at the top-left corner of the kernel is multiplied by the value at bottom-right corner of the neighborhood. This can be expressed by following mathematical expression for kernel of size *m x n*.

$$g(x, y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j, k) f(x - j, y - k)$$

Where *h* denotes the convolution kernel, *f* is the input pixel, *g* is the output pixel. Here *n2 = n/2* and *m2 = m/2*. For *3x3* dimension of kernel, the above expression reduces to

$$g(x, y) = \sum_{k=-1}^{1} \sum_{j=-1}^{1} h(j, k) f(x - j, y - k)$$

Lets see an example. Suppose the convolution matrix is

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

and part of the image pixel is

	72	53	60	
	76	56	65	
	88	78	82	

Now applying above expression, the value of a central pixel becomes

g(x, y) = -1 x 82 + 0 x 78 + 1 x 88 + -1 x 65 + 0 x 56 + 1 x 76 + -1 x 60 + 0 x 53 + 1 x 72 = 29

Algorithm for single pixel convolution can be written as (for *3x3*)

Creating a Kernel of size *3x3* and fill the kernel with coefficients

```
sum = 0;
for k = -1:1
   for j = -1:1
      sum = sum + h(j +1, k + 1)*f(x - j, y - k);
   end
end
g(x, y) = sum;
```

The above discussion is mainly focused on single pixel operation. But the image doesn't have only single pixel. To do the convolution operation on whole image, we must perform convolution on all pixels as follows

```
for y = 0:h
   for x = 0:k
      perform single pixel convolution
   end
end
```

If the Matlab code is

```
f=im2double(imread('in.gif'));
h=   [-1 -1 -1;
     -1 8 -1;
     -1 -1 -1];
[r c]=size(f);
g=zeros(r,c);
for y = 3:r-3
   for x = 3:c-3
      sum = 0;
    for k = 1:3
        for j =1:3
            sum=sum+h(j,k)*f(x-j+1, y-k+1);
        end
    end
    g(x, y) = sum;
   end
end

imshow(g);
```

Here the kernel used is

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

This kernel (matrix) used for edge detection using convolution technique

The output image is

2.2.2 Applications of Convolution

Depending on the value of Kernel and the interrelated techniques, some of the important applications of convolution of images are

Edge detection and Handling

Kernel convolution usually requires values from pixels outside of the image boundaries. There are a variety of methods for handling image edges.

Extend

The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

Wrap

The image is conceptually wrapped (or tiled) and values are taken from the opposite edge or corner.

Crop

Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

3. Filtering in the Frequency Domain

Filtering in the frequency domain is a common image and signal processing technique. It can smooth, sharpen, de-blur, and restore some images.

There are three basic steps to frequency domain filtering:

1. The image must be transformed from the spatial domain into the frequency domain using the Fast Fourier transform.
2. The resulting complex image must be multiplied by a filter (that usually has only real values).
3. The filtered image must be transformed back to the spatial domain.

3.1 Discrete Fourier Transform

Let f(x, y) for x=0, 1, 2, …., M-1 and y=0, 1, 2,….,N-1 denote a digital image of size M x N pixels. The 2-D discrete Fourier transform (DFT) of f(x, y), denoted by F(u, v), is given by the equation

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M+vy/N)}$$

For u = 0,1,2,…., M-1 and v=0,1,2,….,N-1.

It can be expanded into sine and cosine functions, with the variables u and v determining their frequencies. The frequency domain is the coordinate system spanned by F(u, v) with u and v as (frequency) variables. The M x N rectangular region defined by u = 0,1,2,….,M-1 and v = 0,1,2,….,N-1 is often referred to as the *frequency rectangle*. Clearly, the frequency rectangle is of the same size as the input image.

The inverse, discrete Fourier transform (IDFT) is given by

$$F(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)}$$

for x = 0,1,2,….., M-1 and y = 0,1,2,….., N-1.

Even if f(x, y) is a real function, its transform is complex in general. The principle method for analyzing a transform visually is to compute its spectrum [i.e., the magnitude of F(u, v), which is a real function] and display it as an image. Letting R(u, v) and I(u, v) represent the real and imaginary components of F(u, v), the Fourier spectrum is defined as

$$|F(u, v)| = [\![R^2(u, v) + I(u, v)]\!]^{1/2}$$

The phase angle of the transform is defined as

$$\phi(u, v) = \arctan\left[\!\left[\frac{I(u,v)}{R(u,v)}\right]\!\right]$$

The complex function F(u, v) in polar form:

$F(u, v) = |F(u, v)|e^{j\phi(u, v)}$

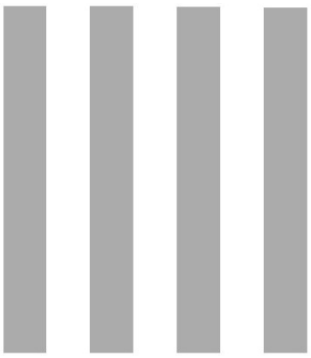
The program for Fourier Transform in matlab

```
f=im2double(imread('after1.gif'));
[r c]=size(f);
n=r*c;
sum=0;
sum1=0;
sum2=0;
g=zeros(r,c);
for y=1:c
   for x=1:r
       sum=0;
       sum1=0;
       sum2=0;
      for v=1:c
        for u =1:r
            sum1 = sum1+f(u,v)*cos(2*pi*(u*x/r+v*y/c));
     sum2 = sum2+f(u,v)*sin(2*pi*(u*x/r+v*y/c));

     sum=sum+(sum1*sum1+sum2*sum2)^(1/2);

        end
      end
    g(x,y)=(sum)/(n);
   end
end

imshow(g);
```



An image before fourier transform

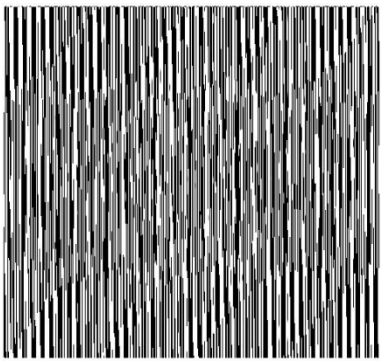


Image after fourier transform

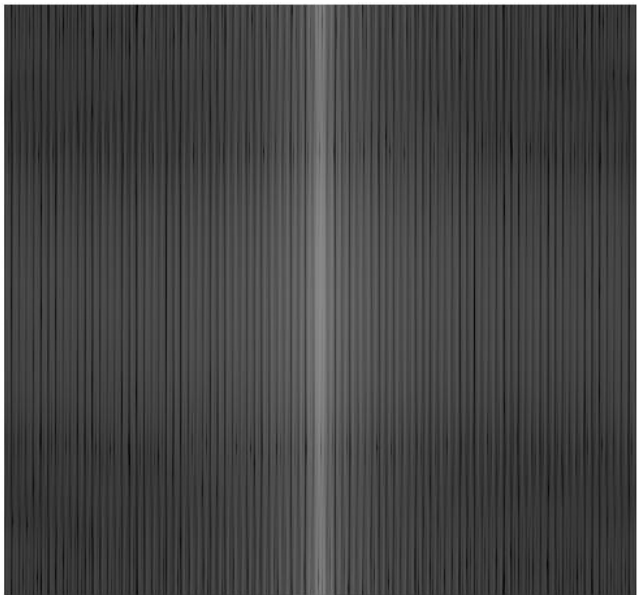
>>g=fft(f); can also be used to get the same result

After applying logarithmic to output image g

>> S=fftshift(log(1+abs(g)));

>> imshow(S,[]);

The fourier spectrum found is



Fourier Transform has its application in image processing as:

- Transformation
- representation and encoding
- smoothing and sharpening images.
- Recognition of objects based on the study of frequency patterns