SONA COLLEGE OF TECHNOLOGY
AUTONOMOUS

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGG

REGULATION-2015 R

U15EC604R - DIGITAL IMAGE PROCESSING LAB MANUAL

Manual Created using LaTeX

Name	:	.
Register No.	:	
Degree & Branch	:	
Year / Section	:	
Batch	:	

Sona Nagar, Junction Main Road, Salem - 636005
Ph: (0427) 4099999 Fax: (0427) 4099888
E-mail: info@sonatech.ac.in, Web: www.sonatech.ac.in.

SONA COLLEGE OF TECHNOLOGY
AUTONOMOUS

Programme Outcomes

PROGRAM OUTCOMES (POs)

On completion of the B.E (ECE) degree graduates will be able to

PO1. Utilize the basic knowledge in mathematics, science and engineering in Electronics and Communication Engineering field.

PO2. Identify, formulate and solve complex problems to achieve demonstrated conclusions using mathematical principles and engineering sciences.

PO3. Design system components that meet the requirement of public safety and offer solutions to the societal and environmental concerns.

PO4. Apply research based knowledge to design and conduct experiments, analyze, synthesize and interpret the data pertaining to Electronics and Communication Engineering problems and arrive at valid conclusions.

PO5. Construct, choose and apply the techniques, resources and modern engineering tools required for Electronics and Communication Engineering applications.

PO6. Apply the contextual knowledge to assess societal, health, safety and cultural issues and endure the consequent responsibilities relevant to the professional engineering practice.

PO7. Examine the impact of engineering solutions in global and environmental contexts and utilize the knowledge for sustained development.

PO8. Develop consciousness of professional, ethical and social responsibilities as experts in the field of Electronics and Communication Engineering.

PO9. Perform effectively as a member/leader in multidisciplinary teams.

PO10. Communicate the engineering activities to engineering society for documentation and presentation.

PO11. Demonstrate knowledge and understanding of the engineering and management principles to manage projects in multidisciplinary environment.

PO12. Demonstrate resourcefulness for contemporary issues and lifelong learning.

PROGRAM SPECIFIC OUTCOMES (PSOs)

On the completion of the B.E (ECE) degree the Electronics and Communication graduates will be able to

PSO1 Apply the fundamental concept of Electronics and Communication Engineering to design a variety of components and systems for applications including signal processing, Image processing, Communication, Networking, Embedded systems, VLSI and control system.

PSO2 Select and apply cutting-edge engineering hardware and software tools to solve complex Electronics and Communication Engineering problems.

Guidelines for Laboratory

The laboratory notebook is a record of all work pertaining to the experiment. This record should be sufficiently complete so that you or anyone else of similar technical background can duplicate the experiment and data by simply following your laboratory notebook. Record everything directly into the notebook during the experiment. Do not use scratch paper for recording data. Do not trust your memory to fill in the details at a later time.

Organization in your notebook is important. Descriptive headings should be used to separate and identify the various parts of the experiment. Record data in chronological order. A neat, organized and complete record of an experiment is just as important as the experimental work.

1. **Heading :** The experiment identification (number) should be at the top of each page. Date should be at the top of the first page of each day's experimental work.

2. **Aim :** A brief but complete statement of what you intend to find out or verify in the experiment should be at the beginning of each experiment.

3. **Diagram :** A block diagram should be drawn and labeled so that the actual experiment block diagram could be easily duplicated at any time in the future. Be especially careful to record all changes made in the block diagram during the experiment.

4. **Procedure :** In general, lengthy explanations of procedures are unnecessary. Be brief. Short commentaries alongside the corresponding data may be used. Keep in mind the fact that the experiment must be reproducible from the information given in your notebook.

5. **Results :** The results should be presented in a form which makes the interpretation easy. Tables are generally used for small amounts of results.

DO'S

- Students must maintain *SILENCE* inside the lab.
- All students should enter the lab wearing laboratory coat.
- Girl students should ensure that their hair is properly covered inside the overcoat
- All students must posses the lab manual and record sheets of the completed experiment to the lab.
- All students must submit the record sheets of the completed experiment as soon as they enter the lab.
- While leaving the lab ensure that you have properly soft shutdowned the computer systems, handed over equipments to the laboratory in-charge. Finally arrange chairs properly orderly.
- Do ask the faculty / staff for assistance if you need help.
- Use computer as much as you need for academic purpose only.
- Explore the MATLAB /LabVIEW Help Files to the best.
- Report malfunctioning of any equipment / computer to the technical staff.
- Utilize the lab manuals provided in the laboratory.

DON'Ts

- Students should not wear the loose clothing
- Students should not roam /chat inside the lab.
- Nobody should misuse the systems in any manner.
- Don' t use Networking Resources for playing games, chat etc.,
- Don' t delete, examine, copy or modify file/data belonging to other users without their prior consent.

Contents

List of Figures

Part I

MATLAB

Using MATLAB for Image Processing

Arrays of MATLAB as Images

The basic data structure in MATLAB is the array, an ordered set of real or complex elements. This object is naturally suited to the representation of images, real-valued ordered sets of color or intensity data.

MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. (Pixel is derived from picture element and usually denotes a single dot on a computer display.)

For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as true color images, require a three-dimensional array, where the first plane in the third dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. This convention makes working with images in MATLAB similar to working with any other type of matrix data, and makes the full power of MATLAB available for image processing applications.

Pixel Indices

Often, the most convenient method for expressing locations in an image is to use pixel indices. The image is treated as a grid of discrete elements, ordered from top to bottom and left to right.

For pixel indices, the row increases downward, while the column increases to the right. Pixel indices are integer values, and range from 1 to the length of the row or column.

There is a one-to-one correspondence between pixel indices and subscripts for the first two matrix dimensions in MATLAB. For example, the data for the pixel in the fifth row, second column is stored in the matrix element (5,2). You use normal MATLAB matrix subscripting to access values of individual pixels. For example, the MATLAB code

```
I(2,15)
```

returns the value of the pixel at row 2, column 15 of the image I.

Similarly, the MATLAB code

```
RGB(2,15,:)
```

returns the R, G, B values of the pixel at row 2, column 15 of the image RGB.

The correspondence between pixel indices and subscripts for the first two matrix dimensions in MATLAB makes the relationship between an image's data matrix and the way the image is displayed easy to understand.

0.1 Basic Image Import, Processing, and Export

- **Reading an Image** : Reads a specific image and stores in the workspace. MATLAB has certain inbuilt images for testing. For example, I = imread ('pout.tiff') would load the image of a young girl. Generally, the command is used to allocate the image to a variable, so that it can be used for further processing during the process of coding (Here I is the variable holding the image). The loaded image can be seen in the workspace of MATLAB.

- **Seeing an image** : Shows the image already loaded in the workspace using a separate window. Considering pout.tiff is already loaded to the variable I, it can be displayed as imshow(I).

- **Image Histogram** : This command is used to see the distribution of intensities in the image. The result of the command can be stored to an- other variable and then plotted (since histogram is a 1D signal). With I as the variable, the command for plotting the histogram can be typed as `his = imhist(I); plot(his);`

- **Writing an Image** : Used to store a processed / adjusted image to a file, for further reference. MATLAB by default writes the image in .png format (Portable Network Graphics), however the user is allowed to write in any format. Suppose the image to be written is in variable I2, then the command imwrite (I2, 'pout2.png'); will create the a new file by name pout2.png and store the contents of I2 in that file.

- **Color to Gray Scale Conversion** The command rgb2gray converts the true color image RGB to the grayscale intensity image I. It makes use of the weighted color space conversion formula as $I = 0.299 \times R + 0.587 \times G + 0.114 \times B$ and in turn eliminates the hue and saturation information, while retaining the luminance content.

- **Contrast Adjustment** The inbuilt command imadjust maps the intensity values in gray scale image I to new values in J such that 1% of data is saturated at low and high intensities of I. This increases the contrast of the output image J

Standard Images for Testing

Standard Test Images are generally used for learning / testing of image processing algorithms. We shall briefly see about a few frequently used test images. One among them which is a common choice among many is the Lena Image.

Lena - Lenna

Lena is a picture of a woman that contains a nice mixture of detail, flat regions, shading, and texture that can do a good job of testing various image processing algorithms.

Figure 1: Standard Lena Test Image

The image was taken in the later part of 1973 by a photographer Dwight Hooker for the centerfold of a famous magazine in US. Later on, after years, Lenna was invited to attend the 50th Anniversary IS&T (Imaging Science and Technology) conference in Boston on May 1997. With the assistance of Jeff Seideman (the President of the Boston chapter of the IS&T) arranged Lenna to appear at the IS&T Boston, as part of an overview of the history of digital imaging.

Figure 2: Lena in 1997!

Some of the test images are made available readily for users in MATLAB. The following are the names of images available in MATLAB which can be used for testing purposes.

File Names with .jpeg extension :

football, greens, hands1, hands2, office 1,office 2,office 3,office 4,office 5, office 6 and yellowlily

File Names with .png extension :

bag, blobs, circles, circlesBrightDark, coins, coloredChips, concordareial, concodorthophoto, fabric, ganrycrane, glass, hestain, liftingbody, onion, pears, pillsetc, rice, saturn, snowflakes, tape, testpat1, , text, tissue, toyobjects, toysflash, toysnoflash, westconcordaerial, westconcodorthophoto, AT3 1m4 01, AT3 1m4 02, AT3 1m4 03, AT3 1m4 04, AT3 1m4 05, AT3 1m4 06, AT3 1m4 07, AT3 1m4 08, AT3_1m4_9, AT3_1m4_10, autumn, board, cameraman, canoe, cell, circbw, circuit, eight, forest, kids, logo, m83, mandi, moon, mri, paper1, pout, shadow, spine, tire, trees.

Important Note:

1. First, clear the workspace for any available previous variables using the command clear all. The workspace window can be docked in the MATLAB main window by clicking on Home → Layout → Workspace. (applicable only for 2014R and higher versions)

2. Close any opened figure windows using the command close all.

Experiment 1

Color Image Processing

Objective of the Experiment

1. Identify the color space model of the given image.

2. Identify the various color planes present in the given image.

3. Visualize each plane of the color image as separate gray scale images.

1.1 RGB True Color Model

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

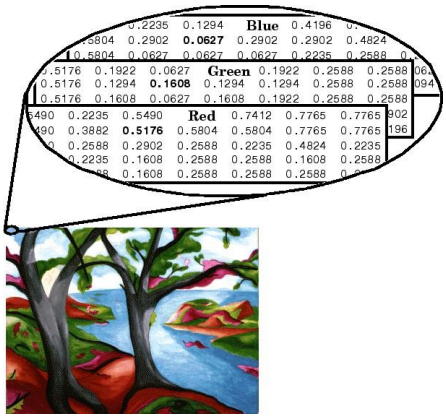

Figure 1.1: Image portraying the RGB color planes of an image

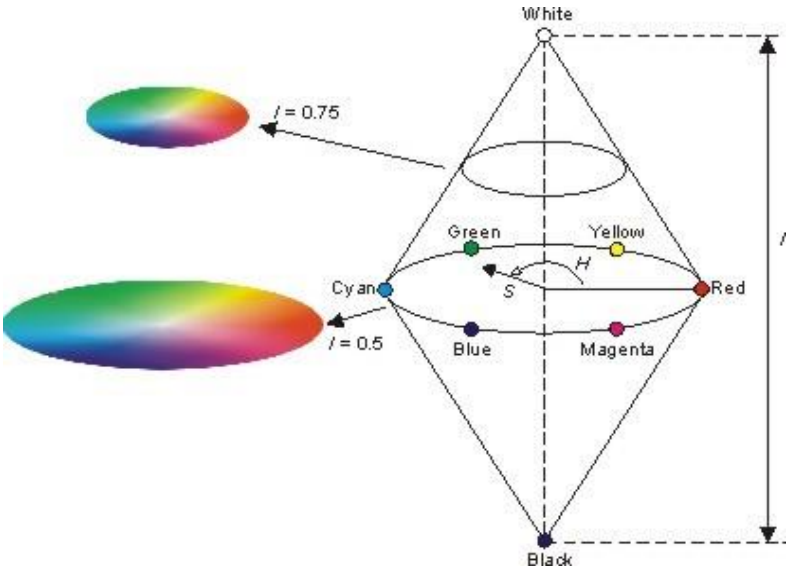

Figure 1.2: HSI Color Model

1.2 The HSI color space

The HSI color space [1] is very important and attractive color model for image processing applications because it represents colors similarly as how the human eye senses colors.

The HSI color model represents every color with three components: hue (H), saturation (S), intensity (I). The Figure 1.2 illustrates how the HIS color space represents colors. The Hue component describes the color itself in the form of an angle between [0,360] degrees. 0 degree mean red, 120 means green, 240 means blue, 60 degrees is yellow & 300 degrees is magenta.

The Saturation component, signals how much the color is diluted with white color. The range of the S component is [0,1]. The Intensity range is between [0,1] and 0 means black, 1 means white.

The Figure 1.2 shows, hue is more meaningful when saturation approaches 1 and less meaningful when saturation approaches 0 or when intensity approaches 0 or 1. Intensity also limits the saturation values.

In the code below, it is necessary to mention the file name of an image or the file name of the image inbuilt in MATLAB. For defining an inbuilt image, it is sufficient to given the file name in single quotes. Alternately, for defining any other image in the computer, the complete path and extension must be specified.

[1] http://www.blackice.com/colorspaceHSI.htm

MATLAB Code

```
clc; % Clearing the Command Window
clear all; % Clearing the Workspace
close all;% Closing all existing display Windows
% Reading an image file from the path and
% storing it to a variable RGB
RGB = imread('FILE NAME WITH PATH');
% Assigning variables R, G and B to RGB
R = RGB;
G = RGB;
B = RGB;
% In the variable R nullify the Green and
% Blue planes.
R(:,:,2) = 0;
R(:,:,3) = 0;
% In the variable G nullify the Red and
% Blue planes.
G(:,:,1) = 0;
G(:,:,3) = 0;
% In the variable B nullify the Red and
% Green planes.
B(:,:,1) = 0;
B(:,:,2) = 0;
% In the same window display the Original Image
% Red Plane, Green Plane and Blue Plane.
subplot(2,2,1),imshow(RGB),title('original image');
subplot(2,2,2),imshow(R),title('Red Component');
subplot(2,2,3),imshow(G),title('Green Component');
subplot(2,2,4),imshow(B),title('Blue Component');
```

The Output of the code is shown in the Figure 1.3.

Beyond the Lab Experiment

1. Display any one of the color plane as a gray scale image.
2. Write the MATLAB code for color-to-gray scale conversion.
3. Try converting the image from RGB color space to other color spaces and hence visualize the planes individually. Make use of MATLAB Help for converting to other color space (Example : rgb2hsv).
4. Why is it necessary to convert a given color image to gray scale image? [2]

[2] Not required for *Record Work*

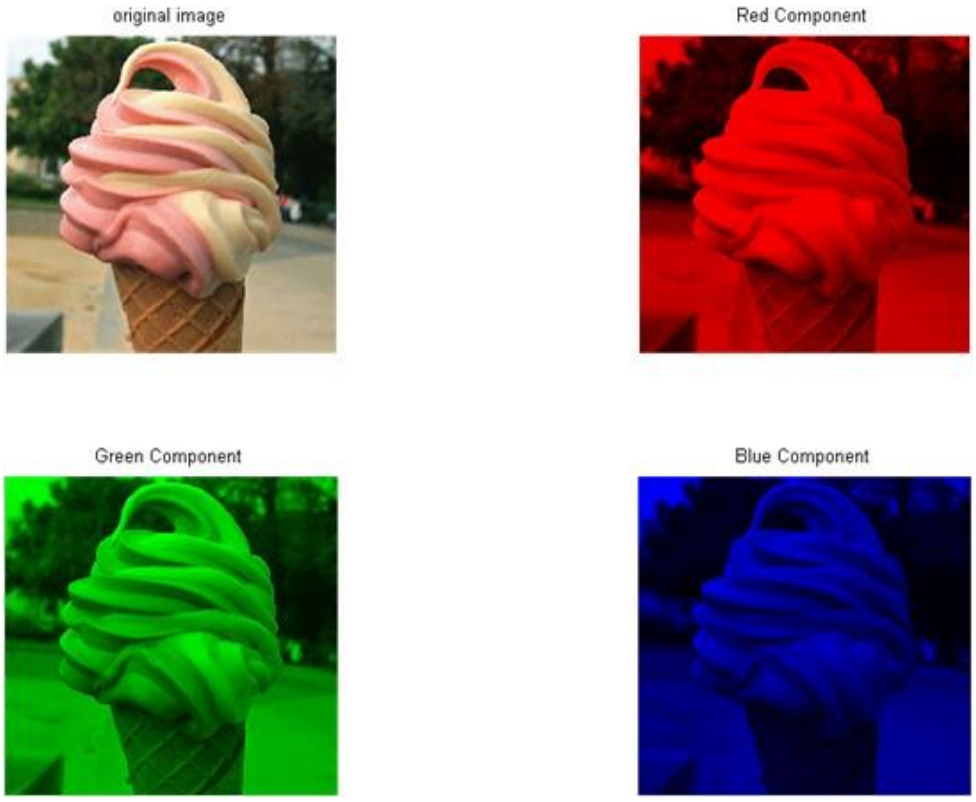

Figure 1.3: Extracting and Displaying Red, Green and Blue Color planes

Experiment 2

Image Quantization

Objective of the Experiment

1. Understand the principle of Image Quantization technique (False Contour Effect).

2.1 False Contour Effect

It is has been realized (both subjectively and objectively) that the quality of a gray-level image is dramatically affected by its gray-level resolution. Other words, increasing the number of bits per pixel has a great effect in improving the quality of gray-level images. This is because that a higher number of gray levels would give a smooth transition along the details of the image and hence improving its quality to the human eye.

MATLAB Code

```
clc; % Clearing the Command Window
clear all; % Clearing the Workspace
close all; % Closing all existing display Windows

% Reading an image file from the path and
% storing it to a variable 'a'
a= imread ('cameraman.tif');

% In the same figure display the Original Image
% on the left side top corner
subplot(321), imshow(a), title('Original image');
%using 128 gray levels
subplot(322), imshow(grayslice(a,128), gray(128));
title('Image with 128 gray level');
%using 64 gray levels
subplot(323), imshow(grayslice(a,64), gray(64));
```

```
title('Image with 64 gray level');
%using  32 gray levels
subplot(324),imshow(grayslice(a,32),gray(32));
title('Image with 32 gray level');
%using  16 gray levels
subplot(325),imshow(grayslice(a,16),gray(16));
title('Image with 16 gray level');
%using  8 gray levels
subplot(326),imshow(grayslice(a,8),gray(8));
title('Image with 8 gray level');
```

Output of the above MATLAB code is shown in the figure 2.1.

Beyond the Lab Experiment

1. Display the Camera Man Image using gray level of '2'. Interpret the image that has been obtained by the process.
2. Write a MATLAB code for demonstrating Checkerboard effect.
3. Read the help file for the command im2bw and hence use it. Compare the result of obtained with that of the previous one. [1]

[1] Not required for *Record Work*

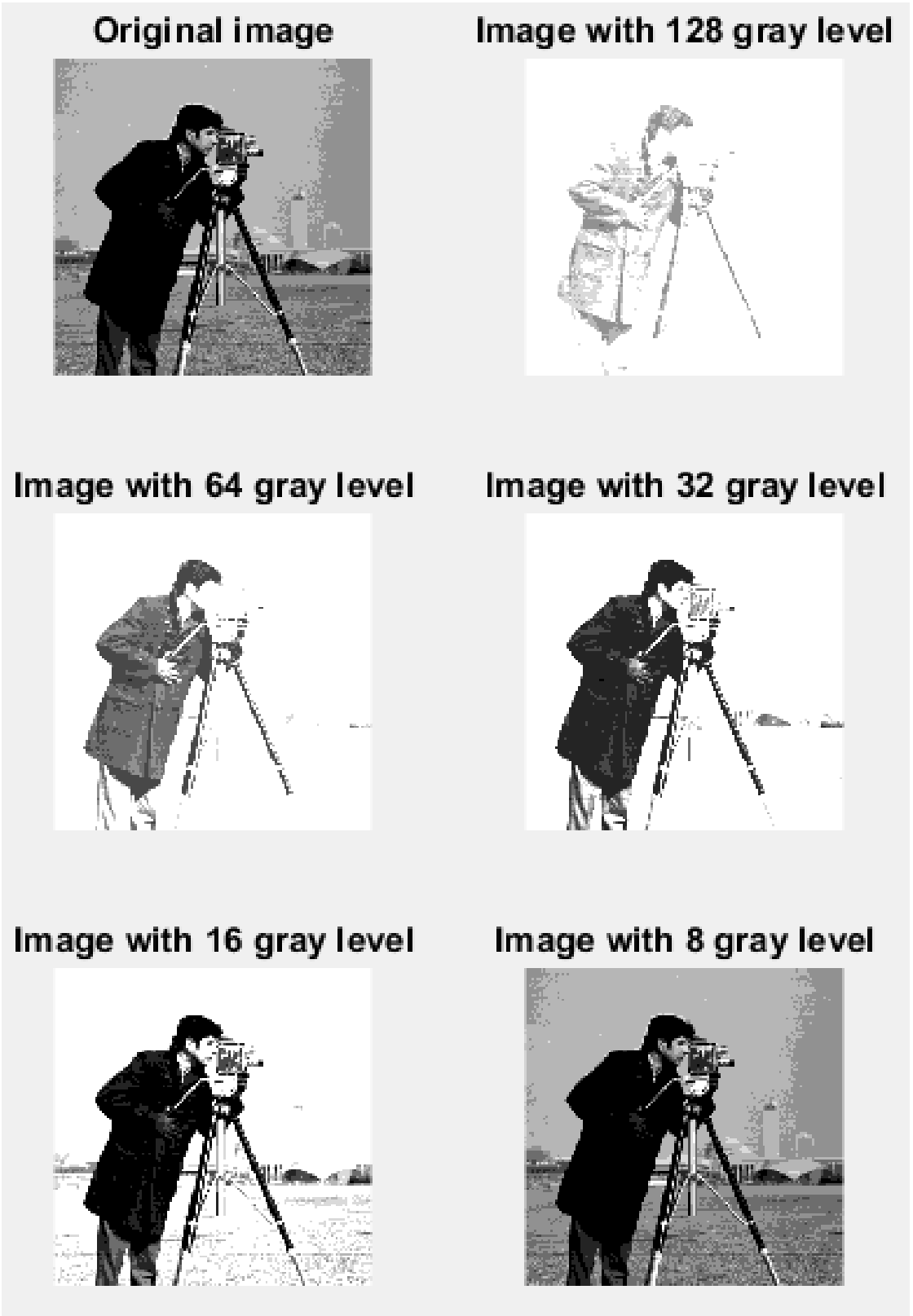

Figure 2.1: Original Image and its corresponding False Contoured Images

Experiment 3

Bit Plane Slicing

Objective of the Experiment

1. To visualize each bit of a pixel collectively as a separate gray scale image.
2. To interpret the obtained gray scale image with other image processing methods such as binarization.

3.1 Slicing the Image Pixels

Given an X-bit per pixel image, slicing the image at different planes (bit-planes) plays an important role in image processing. An application of this technique is data compression. In general, 8-bit per pixel images are processed. Each bit of a pixel is grouped together to be visualized as a gray scale image. For example, the zeroth bit is the least significant bit (LSB). All the LSB bits of the image are placed at their corresponding points to form a Bit Plane Image.

MATLAB Code

```
clc;
clear all;
close all;
a = imread('cameraman.tif');
[m n]= size(a); % Extract the size of the image
%To extract 7th bit as a gray scale image
for i=1:m,
        for j=1:n,
                b7(i,j)= bitand(a(i,j),128);
        end
end
%To extract 6th bit as a gray scale image
for i=1:m,
        for j=1:n,
```

```
                b6(i,j)=bitand(a(i,j),64);
        end
end
%To extract 5th bit as a gray scale image
for i=1:m,
        for j=1:n,
                b5(i,j)=bitand(a(i,j),32);
        end
end
%To extract 4th bit as a gray scale image
for i=1:m,
        for j=1:n,
                b4(i,j)=bitand(a(i,j),16);
        end
end
%To extract 3rd bit as a gray scale image
for i=1:m,
        for j=1:n,
                b3(i,j)=bitand(a(i,j),8);
        end
end
%To extract 2nd bit as a gray scale image
for i=1:m,
        for j=1:n,
                b2(i,j)=bitand(a(i,j),4);
        end
end
%To extract 1st bit as a gray scale image
for i=1:m,
        for j=1:n,
                b1(i,j)=bitand(a(i,j),2);
        end
end
%To extract 0th bit as a gray scale image
for i=1:m,
        for j=1:n,
                b0(i,j)=bitand(a(i,j),1);
        end
end
%Code to show the resultant images
subplot(2,2,1),imshow(a),title('original image'),
subplot(2,2,2),imshow(b5),title('5th bitplane image'),
subplot(2,2,3),imshow(b6),title('6th bitplane image'),
subplot(2,2,4),imshow(b7),title('7th bitplane image');
```

Figure 3.1: Original Image and corresponding 5^{th}, 6^{th} and 7^{th} Bit Planes

Beyond the Lab Experiment

1. The given code shall display only the 5^{th}, 6^{th} and 7^{th} Planes. Extend the code and display all the planes.
2. What is the major difference between the 1^{st} plane of the '*cameraman*' and the output of im2bw command of MATLAB?
3. Perform the same experiment with other standard test images. [1]

[1] Not required for *Record Work*

Experiment 4

Histogram Equalization

Objective of the Experiment

1. To display the Histogram of the gray scale image.
2. Carry out an gray value equalization based on the obtained histogram
3. Compare the Histogram of the original and equalized image.

4.1 Equalizing the Contrast using Histogram

Histogram equalization is a technique for adjusting image intensities to enhance contrast.

Let f be a given image represented by a $m \times n$ matrix of integer pixel intensities ranging from 0 to 255. Let p denote the normalized histogram of f with a bin for each possible intensity. So

$$p_n = \frac{\text{number of pixels with intensity n}}{\text{total number of pixels}} \text{ where } n = 0, 1, \ldots 255$$

The histogram equalized image g will be defined by

$$g_{i,j} = floor((L-1)\sum_{n=0}^{f_{i,j}} p_n)$$

where L represents the maximum possible gray value in the image and i, j indicate the position of the pixel.

This mathematical process of equalization can be merely done using the histeq inbuilt command of MATLAB. The inbuilt command imhist displays the histogram of a image.

MATLAB Code

```
clc; clear all; close all;
a=imread('cameraman.tif');
%Displaying the Original Image
subplot(221); imshow(a);
%Show the Histogram of the Original Image
subplot(222); imhist(a,255);
%Carry Out Histogram Equalization using
%inbuilt histeq command
j=histeq(a);
%Show the Histogram Equalized Image
subplot(223); imshow(j);
%Show the Histogram of the Equalized Image
subplot(224); imhist(j,255);
```

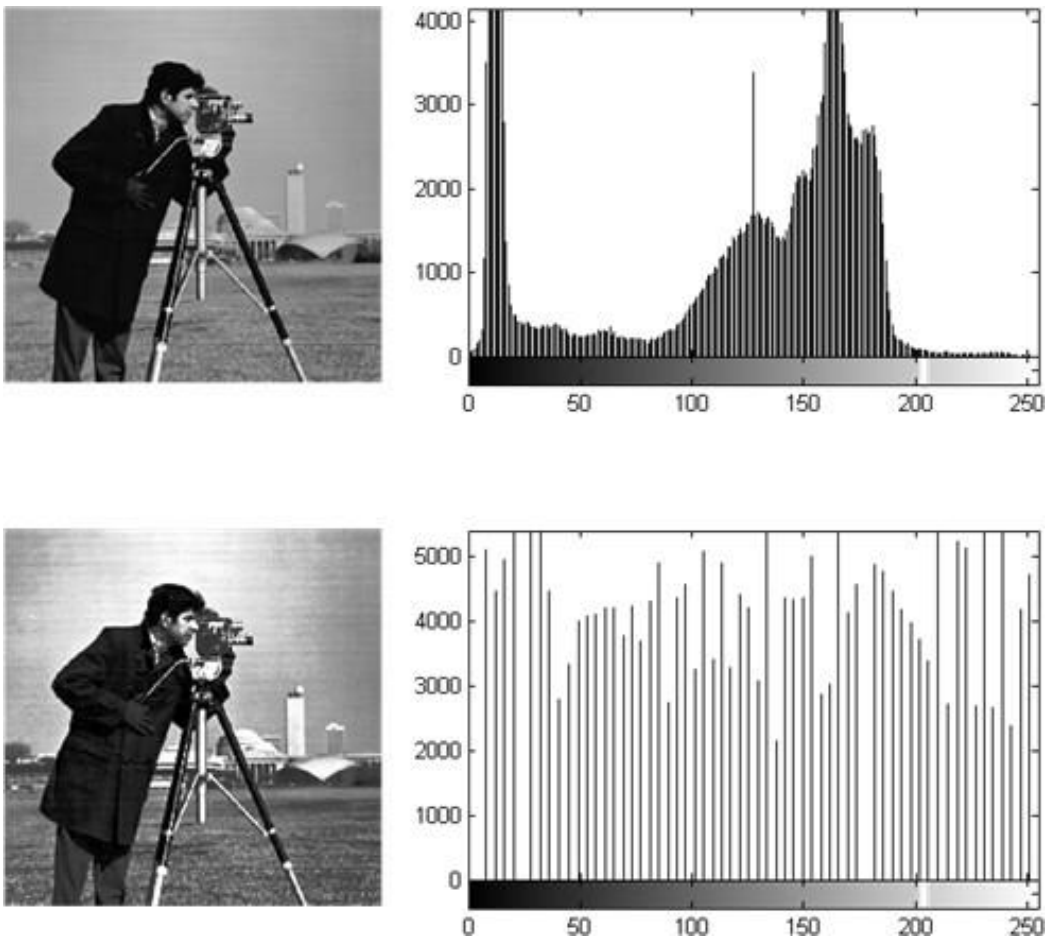

Figure 4.1: From left top (anti-clockwise) Original Image, Histogram of the Original Image, Histogram of Equalized Image, Histogram Equalized Image

Beyond the Lab Experiment

1. What kind of results would you get if bit plane slicing is done on a histogram stretched image?
2. Write a MATLAB code for computing the histogram of the image, without using the inbuilt command of MATLAB.[1]
3. Try giving a color image as input to the program. [2]

[1] Not required for *Record Work*
[2] Not required for *Record Work*

Experiment 5

Frequency Domain Translation

Objective of the Experiment

1. To map a given image from spatial domain to frequency domain ($f : \mathsf{R} \rightarrow \mathsf{C}$ or $f : \mathsf{R} \rightarrow \mathsf{R}$).
2. Make use of Fast Fourier and Discrete Cosine Transformation for mapping.
3. Understand the effect of mapping on the original and disturbed image.
4. Try reconstructing the image from the obtained transform co-efficients.

5.1 Fast Fourier Transform

5.1.1 Fast Fourier Transform Without fftshift

The Fast Fourier Transform (FFT) is simply a fast (computationally efficient) way to calculate the Discrete Fourier Transform (DFT).

The DFT takes N^2 operations for N points. Since, at any stage the computation required to combine smaller DFTs into larger DFTs is proportional to N, and there are $log_2(N)$ stages (for radix 2), the total computation is proportional to $N * log_2(N)$. Therefore, the ratio between a DFT computation and an FFT computation for the same N is proportional to $N/log_2(n)$. In cases where N is small this ratio is not very significant, but when N becomes large, this ratio gets very large. (Every time you double N, the numerator doubles, but the denominator only increases by 1.)

MATLAB Code

```
clc;clear all; close all;
I=imread('liftingbody.png');
I=im2double(I); %Converting to integer values to double
subplot(221),imshow(I);
title('Original Image');%Display the Image
F=fft2(I); % compute the FFT for the image
% Display the Magnitude component of the FFT
subplot(222),imshow(F);
title('Magnitude Spectrum');
% Display the Log Magnitude of FFT.
G=mat2gray(log10(1+abs(F)));
subplot(223),imshow(G);
title('Log-scaled Magnitude Spectrum');
% Using the FFT compute Inverse
irecons=real(ifft2(F));
% Show the reconstructed image
subplot(224),imshow(irecons);
title('Reconstructed Image');
```

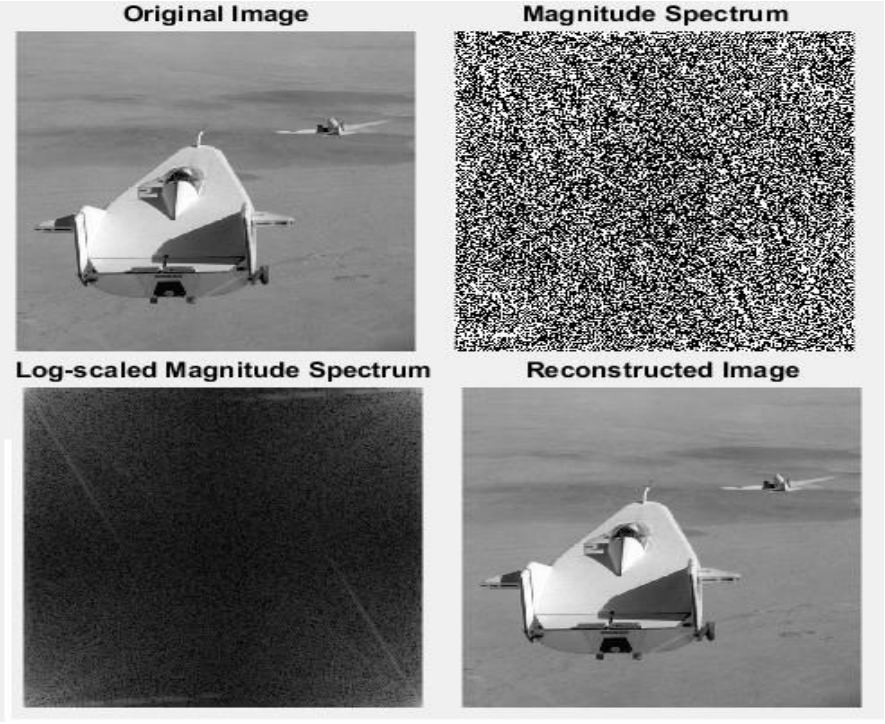

Figure 5.1: Fast Fourier Transform without fftshift

5.1.2 Fast Fourier Transform With fftshift

FFT using Translation Property

The translation property of FFT maps the co-efficients in such a way that

$$f^*(x,y) = f(x,y) \times (-1)^{x+y}$$

The fftshift command of MATLAB does this process.

fftshift command of MATLAB

Shift zero-frequency component to center of spectrum.

Y = fftshift(X) rearranges the outputs of fft, fft2, and fftn by moving the zero-frequency component to the center of the array. It is useful for visualizing a Fourier transform with the zero-frequency component in the middle of the spectrum.

For vectors, fftshift(X) swaps the left and right halves of X. For matrices, fftshift(X) swaps the first quadrant with the third and the second quadrant with the fourth as shown in the figure 5.2.

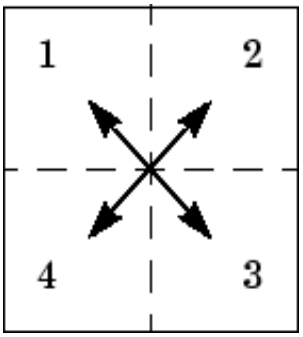

Figure 5.2: fftshift command swaps the first quadrant with the third and the second with the fourth

MATLAB Code

```
clc; clear all; close all;
I=imread('coins.png');
I=im2double(I);
% Display the original Image
subplot(221), imshow(I);
title('Original Image');
F=fftshift(fft2(I));
% Display the Magnitude Spectrum of the FFT
subplot(222), imshow(F);
title('Magnitude Spectrum');
G=mat2gray(log10(1+abs(F)));
subplot(223), imshow(G);
% Display the Log scaled Magnitude Spectrum of the FFT
title('Log-scaled Magnitude Spectrum');
irecons=real(ifft2(fftshift(F)));
subplot(224), imshow(irecons);
% Display the Reconstructed Image
title('Reconstructed Image');
```

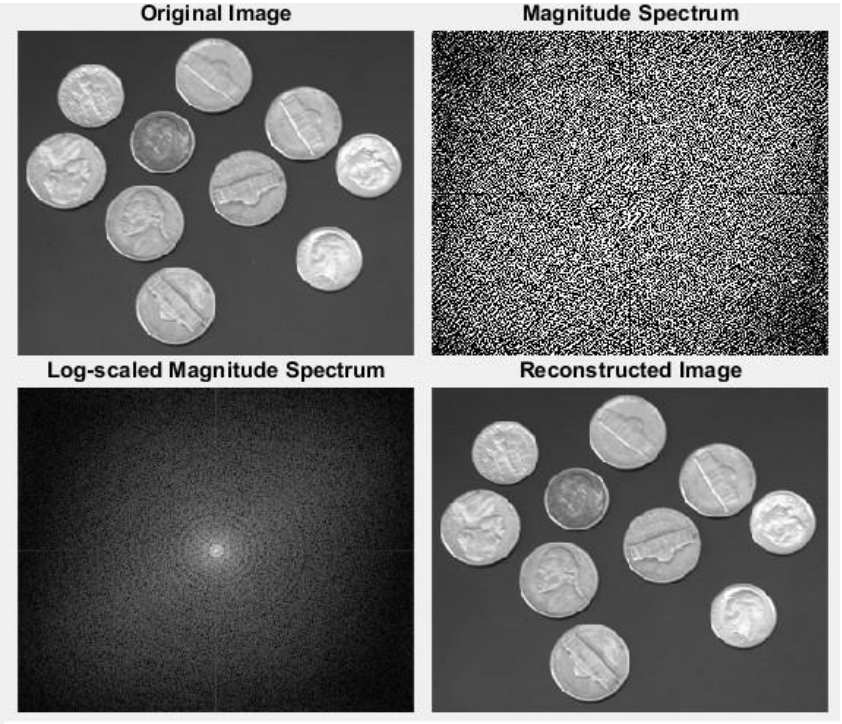

Figure 5.3: Fast Fourier Transform with fftshift

5.2 Discrete Cosine Transform

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. [1]

The discrete cosine transform is a linear, invertible function $f : \mathbb{R}^N \to \mathbb{R}^N$ (where $\mathbb{R}$ denotes the set of real numbers), or equivalently an invertible $N \times N$ square matrix. There are several variants of the DCT with slightly modified definitions. The N real numbers $x_0, \ldots, x_{N-1}$ are transformed into the N real numbers $X_0, \ldots, X_{N-1}$ according to the formula:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right] \qquad k = 0, 1, \ldots N-1$$

MATLAB Code

```
clc;clear all; close all;
I=imread('cameraman.tif');
subplot(131);imshow(I);
title('Original Image');
% Computing the DCT of the image.
F=dct2(I);
subplot(132);imshow(F);
title('Transformed Image');
% Reconstructing the Image from DCT
recon=idct2(F);
subplot(133);imshow(uint8(recon));
title('Reconstructed Image');
```

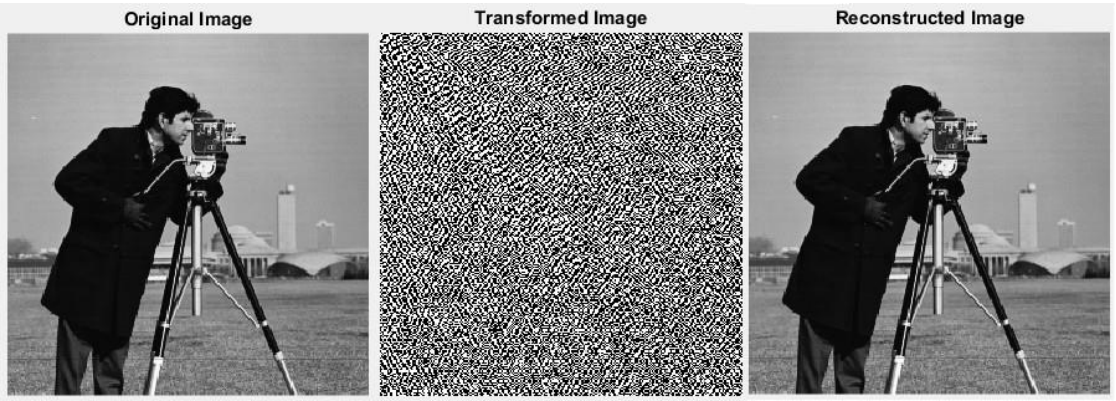

Figure 5.4: Original Image, DCT of the Image and Reconstructed Image from computed DCT

[1] https://en.wikipedia.org/wiki/Discrete cosine transform

Beyond the Lab Experiment

1. Rotate a gray scale image to an angle, say 45° and hence compute the FFT and DCT of the image. Compare it with the FFT and DCT of the original Image and hence interpret the effect of transformation.
2. See what happens when the program is fed with a color image as input[2].
3. Why is the log-magnitude spectrum not computed in case of Discrete Cosine Transformed image? [3].

[2]Not required for *Record Work*
[3]Not required for *Record Work*

Experiment 6

Image Enhancement Frequency Domain

Objective of the Experiment

1. To enhance a given image in frequency domain.
2. Apply the High Pass, Low Pass and Band Pass Filter on standard test images in frequency domain.
3. Understand the various filtering models such as Ideal, Butterworth and Gaussian along with their mathematical expressions.

6.1 Frequency Domain Processing

The reason for doing the filtering in the frequency domain is generally because it is computationally faster to perform two 2D Fourier transforms and a filter multiply than to perform a convolution in the image (spatial) domain. This is particularly so as the filter size increases.

It is assumed that the reader is familiar with the Discrete Fourier Transform, its properties and the transforms of commonly used functions (delta, noise, rectangular pulse, step, etc) [1].

The commonly used filtering models are

- Ideal filter
- Butterworth Filter
- Gaussian Filter

6.1.1 Low Pass Filter

The ideal bandpass filter passes only frequencies within the pass band and gives an output in the spatial domain that is in most cases blurred and/or ringed. It is the easiest bandpass filter to simulate but its vertical edges and sharp corners are not realizable in the physical world.

[1] http://paulbourke.net/miscellaneous/imagefilter/

The corresponding formulas of these filters is shown below, where D_0 is a specified non-negative number. D(u,v) is the distance from point (u,v) to the center of the filter.

Low Pass Filter Models

Ideal:

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

Butterworth :

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}}$$

Gaussian:

$$H(u,v) = e^{-\frac{1}{2}\left[\frac{D(u,v)}{D_0}\right]^2}$$

MATLAB Code - LPF

Gaussian Model

```
clc;clear all; close all;
I=imread('cameraman.tif');
[M N]=size(I);
for u=1:M
        for v=1:N
                D(u,v)=((u-(M/2))^2+(v-(N/2))^2)^(1/2);
        end
end
for u=1:M
        for v=1:N
                H(u,v)=1/(1+(D(u,v)/20)^4);
        end
end
F=fft2(I);
F=fftshift(F);
Y=F.*H;
Y1=ifftshift(Y);
Y2=ifft2(Y1);
subplot(121),imshow(I);
title('Original Image');
subplot(122),imshow(uint8(Y2));
title('Lowpass Filtered Image');
```

Refer Figure 6.1 for the output of the code.

6.1.2 High Pass Filter

High Pass Filter Models

Ideal:

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \geq D_0 \\ 0 & \text{if } D(u,v) < D_0 \end{cases}$$

Butterworth :

$$H(u,v) = \frac{1}{1 + \left[\frac{D_0}{D(u,v)}\right]^{2n}}$$

Gaussian:

$$H(u,v) = 1 - e^{-\frac{1}{2}\cdot\left[\frac{D(u,v)}{D_0}\right]^2}$$

Refer Figure 6.2 for the expected output of the code.

6.1.3 Band Pass Filter

Band Pass Filter Models

Ideal:

$$H(u,v) = \begin{cases} 1 & \text{if } D_l \leq D(u,v) \leq D_h \\ 0 & \text{otherwise} \end{cases}$$

Butterworth :

$$H_{LP}(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_l}\right]^{2n}}$$

$$H_{HP}(u,v) = \frac{1}{1 + \left[\frac{D_h}{D(u,v)}\right]^{2n}}$$

$$H_{BP}(u,v) = H_{LP}(u,v) * H_{HP}(u,v)$$

Gaussian:

$$H_{LP}(u,v) = e^{-\frac{1}{2}\cdot\left[\frac{D(u,v)}{D_l}\right]^2}$$

$$H_{HP}(u,v) = 1 - e^{-\frac{1}{2}\cdot\left[\frac{D(u,v)}{D_0}\right]^2}$$

$$H_{BP}(u,v) = H_{LP}(u,v) * H_{HP}(u,v)$$

Refer Figure 6.3 for the expected output for a High Pass Filtered Image.

Beyond the Lab Experiment

1. Write down the MATLAB code for the various other models of filtering techniques listed above.
2. Can the given MATLAB code be optimized? [Hint: Reduce the for loop in the code]

Figure 6.1: Original Image and its Low Pass Filtered Image

Figure 6.2: Original Image and its High Pass Filtered Image

Figure 6.3: Original Image and its Band Pass Filtered Image

Experiment 7

Restoration Filters

Objective of the Experiment

1. Experiment a Restoration Filter in Frequency Domain
2. Use the method of minimizing mean square error between the estimated random process and the desired process.

7.1 Wiener Filter - LMS Filter

Given a system:

$$y(t) = (h * x)(t) + \eta(t)$$

where $*$ denotes convolution and:

- $x(t)$ is some input signal (unknown) at time t .
- $h(t)$ is the known impulse response of a linear time-invariant system
- $\eta(t)$ is some unknown additive noise, independent of $x(t)$
- $y(t)$ is our observed signal

Our goal is to find some $g(t)$ so that we can estimate $x(t)$ as follows:

$$\hat{x}(t) = (g * y)(t)$$

where $\hat{x}(t)$ is an estimate of $x(t)$ that minimizes the mean square error.

The Wiener deconvolution filter provides such a $g(t)$. The filter is most easily described in the frequency domain:

$$G(f) = \frac{H^*(f)S(f)}{|H(f)|^2 S(f) + N(f)}$$

where:

$G(f)$ and $H(f)$ are the Fourier transforms of g and h, respectively at frequency f . $S(f)$ is the mean power spectral density of the input signal $x(t)$, $N(f)$ is the

mean power spectral density of the noise $n(t)$, the superscript * denotes complex conjugation. The filtering operation may either be carried out in the time-domain, as above, or in the frequency domain:

$$\hat{X}(f) = G(f)Y(f)$$

(where $\hat{X}(f)$ and $Y(f)$ are the Fourier transforms of $\hat{x}(t)$ and $y(t)$, respectively) and then performing an inverse Fourier transform on $\hat{X}(f)$ to obtain $\hat{x}(t)$.

Note that in the case of images, the arguments t and f above become two-dimensional; however the result is the same.

MATLAB Code

```
clc;clear all; close all;
I=checkerboard(8);
imshow(I);
n=imnoise(zeros(size(I)),'gaussian',0,0.01);
[R, psf]=degradation_model(I,n);
sn=abs(fft2(n));
na=sum(sn(:))/numel(n);
sf=abs(fft2(I));
fa=sum(sf(:))/numel(I);
nspr=na/fa;
irestore=deconvwnr(R, psf, nspr);
subplot(133),imshow(i_restore);
title('Restored Image');
```

Function for Degradation Model

```
function [R,psf]=degradation_model(I,n)
subplot(131), imshow(I);
title('Original Image');
psf=fspecial('motion',7,45);
i_degrade=imfilter(I,psf,'circular');
R=i_degrade+n;
subplot(132),imshow(R);
title('Degraded Image');
```

Beyond the Lab Experiment

1. Try out the same code using a different image, rather than the generating a image in MATLAB.

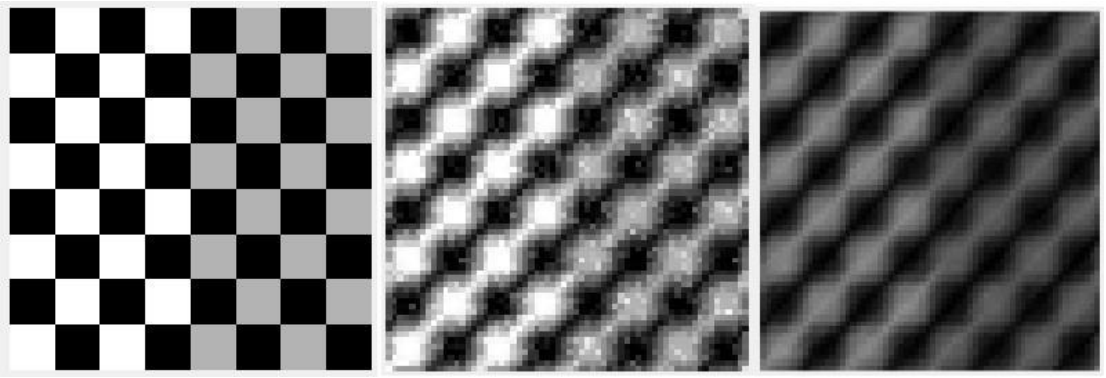

Figure 7.1: From left: Image generated using MATLAB, Gaussian Noise added to the image, restored image

Experiment 8

Spatial Domain Filtering

Objective of the Experiment

1. To sharpen the image using various mathematical equations.
2. Understand the simple noise models (viz. Gaussian Noise and Salt & Pepper Noise) that can be present in a image.
3. To experiment the process of Image enhancement in spatial domain.
4. Make use of both inbuilt and mask based filtering techniques for image enhancement.

8.1 Sharpening Filter

Roberts Cross - Gradient Operator

In order to perform edge detection with the Roberts operator we first convolve the original image, with the following two kernels

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

Let I(x,y) be a point in the original image and $G_x(x, y)$ be a point in an image formed by convolving with the first kernel and $G_y(x, y)$ be a point in an image formed by convolving with the second kernel. The gradient can then be defined as: $\nabla I(x, y) = G(x, y) = \sqrt{G_x^2 + G_y^2}$.

Sobel Operator

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define $\mathbf{A}$ as the source image, and G_x and G_y are two images which at each point contain the horizontal and vertical derivative approximations, the computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

where $*$ denotes the 2-dimensional signal processing convolution operation.

Prewitt Operator

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define $\mathbf{A}$ as the source image, and $\mathbf{G_x}$ and $\mathbf{G_y}$ are two images which at each point contain the horizontal and vertical derivative approximations, the latter are computed as:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * \mathbf{A}$$

where $*$ denotes the 1-dimensional convolution operation.

Laplacian of Gaussian (LoG)

The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

This can be calculated using a convolution filter.

0	1	0
1	4	-1
0	1	0

Table 8.1: Log Mask

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian.

Using one of these kernels, the Laplacian can be calculated using standard convolution methods.

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result.

Canny Edge Detection

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply non-maximum suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

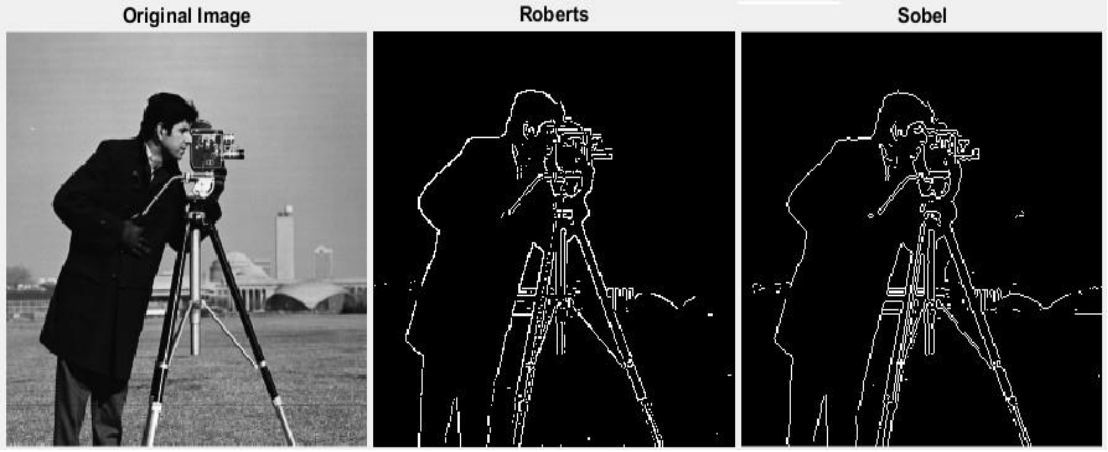

Figure 8.1: Original Image, Roberts and Sobel Edge Detection

Figure 8.2: Perwitt, Log and Canny Edge Detection

MATLAB Code

```
clc;
clear all;
a = imread('cameraman.tif');
%a=rgb2gray(I);
b=edge(a,'roberts');
c=edge(a,'sobel');
d=edge(a,'prewitt');
e=edge(a,'log');
f=edge(a,'canny');
subplot(321),imshow(a),title('Original Image');
subplot(322),imshow(b),title('Roberts');
subplot(323),imshow(c),title('Sobel');
subplot(324),imshow(d),title('Prewitt');
subplot(325),imshow(e),title('Log');
subplot(326),imshow(f),title('Canny');
```

Noise Models

Gaussian Noise Model

Gaussian noise is statistical noise having a probability density function (PDF) equal to that of the normal distribution, which is also known as the Gaussian distribution. In other words, the values that the noise can take on are Gaussian-distributed.

The probability density function p of a Gaussian random variable z is given by:

$$p_{tt}(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

where z represents the grey level, μ the mean value and σ the standard deviation

Salt and Pepper Noise Model

To obtain an image with 'speckle' or 'salt and pepper' noise we need to add white and black pixels randomly in the image matrix. This common form of noise is data drop-out noise (commonly referred to as intensity spikes, speckle or salt and pepper noise). Here, the noise is caused by errors in the data transmission. The corrupted pixels are either set to the maximum value (which looks like snow in the image) or have single bits flipped over. In some cases, single pixels are set alternatively to zero or to the maximum value, giving the image a 'salt and pepper' like appearance. Unaffected pixels always remain unchanged.

8.2 Average Filter

The idea of average filtering is simply to replace each pixel value in an image with the average value of its neighbors, including itself. This has the effect of eliminating pixel values which are not representing of their surroundings.

MATLAB Code

Gaussian Noise Removal using Average Filter

```
clc;clear all; close all;
I=imread('cameraman.tif');
subplot(131), imshow(I);
title('Original Image');
h=ones(5,5)/25;%neighbourhood averaging
i_noise=imnoise(I,'gaussian',0,.01); % Gaussian Noise
j=imfilter(i_noise,h);
subplot(132),imshow(i_noise),title('Noisy Image');
subplot(133),imshow(j); title('Average Filtered Image');
```

The output of the code is shown in Figure 8.4

Salt and Pepper Noise Removal

In the above code, the Salt and Pepper noise can be added by changing noise addition line as follows:

```
i_noise=imnoise(I, 'salt & pepper',.02);
```

The output of the code in shown in Figure 8.5

8.3 Median Filter

The median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.) Figure 8.3 illustrates an example calculation.

The inbuilt command medfilt2 is used for Median Filtering a given image. For example, it can be used as

```
j=medfilt2(i_noise,[5 5]);
```

The outputs for the Median Filter in the case of Gaussian Noise and Salt and Pepper Noise are shown in the Figures 8.6 and 8.7

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Neighbourhood values:

115, 119, 120, 123, 124,
125, 126, 127, 150

Median value: 124

Figure 8.3: Median Filter Computation Sample

Figure 8.4: Removal of Gaussian Noise using Average Filtering

Beyond the Lab Experiment

1. Carry out the Edge Enhancement Technique using the following mask $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

2. Carry out the various Edge Detection techniques (Robert, Sobel and Prewitt) without using MATLAB inbuilt command. [Hint: For Prewitt Operator the masks are $\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$ and $\mathbf{G_y} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} * \mathbf{A}$. Apply these on the images to get the edges.]

Figure 8.5: Removal of Salt and Pepper Noise using Average Filter

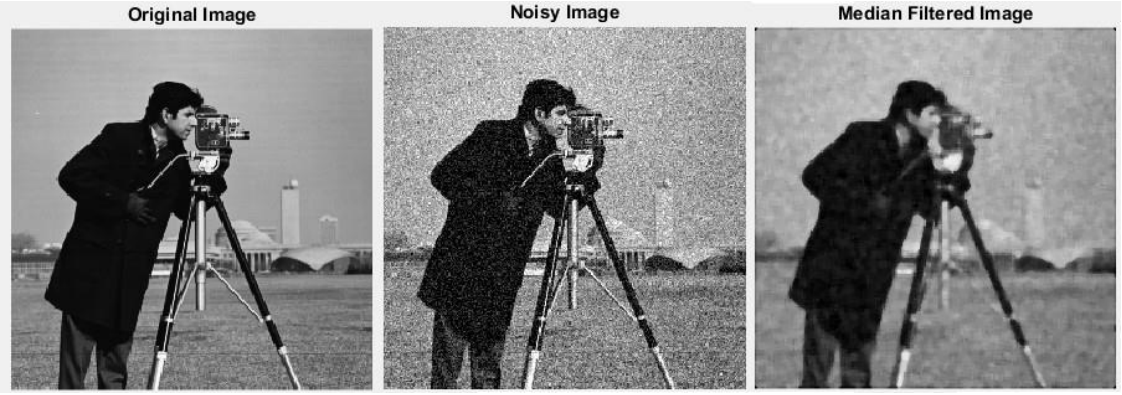

Figure 8.6: Removal of Gaussian Noise using Median Filter

Figure 8.7: Removal of Salt and Pepper Noise using Median Filter

Experiment 9

Image Compression

Objective of the Experiment

1. Experiment the process of JPEG Compression using MATLAB code
2. Improvise the code by including Huffman Coding technique.

9.1 JPEG

JPEG uses a lossy form of compression based on the discrete cosine transform (DCT). This mathematical operation converts each frame/field of the video source from the spatial (2D) domain into the frequency domain.

MATLAB Code

```
clc;
clear all;
I2=imread('liftingbody.png');
I1=I2;
[r c]= size(I2); % Extract the size of I2
I12=double(I2); % Ensure that I2 is computable
I=I12 -(128*ones(r)); % Diff b/w I12 and 128
DCT_matrix8=dct(eye(8)); % Form DCT matrix
iDCT_matrix8=DCT_matrix8';
for i1=[1:8:r]
        for i2=[1:8:c]
                zBLOCK=I(i1:i1+7,i2:i2+7);
                win1=DCT_matrix8 * zBLOCK * iDCT_matrix8;
                dct_domain(i1:i1+7,i2:i2+7)=win1;
        end % For each 8 x 8 block compute the DCT
end
for i1=[1:8:r]
        for i2=[1:8:c]
```

```
                win1=dct_domain(i1:i1+7,i2:i2+7);
                win2=round(win1./128);
                dct_quantized(i1:i1+7,i2:i2+7)=win2;
        end % For each 8 x8 quantize the computed DCT
end
for i1=[1:8:r]
        for i2=[1:8:c]
                win2=dct_quantized(i1:i1+7,i2:i2+7);
                win3=win2 .*128;
                dct_dequantized(i1:i1+7,i2:i2+7)=win3;
        end % For each computed DCT, Dequantize
end
for i1=[1:8:r]
        for i2=[1:8:c]
                win3=dct_dequantized(i1:i1+7,i2:i2+7);
                win4=iDCT_matrix8 * win3 * DCT_matrix8;
                dct_restored(i1:i1+7,i2:i2+7)=win4;
        end % Compute the Inverse DCT
end
subplot(221),imshow(I2),title('original image');
subplot(222),imshow(dct_quantized),title('quantized image');
subplot(223),imshow(dct_restored),title('reconstructed image');
```

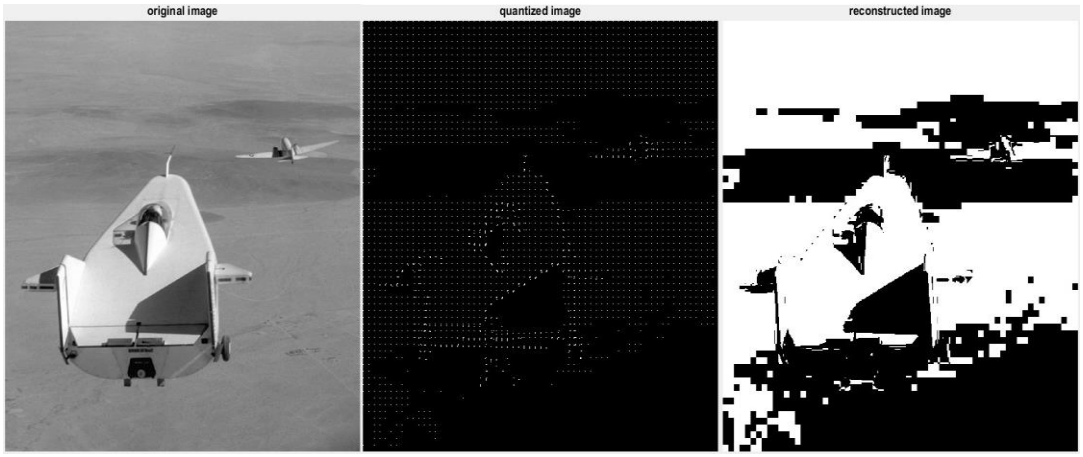

Figure 9.1: From Left: Original Image, DCT Quantized Image, Reconstructed Image *(Note the Dots in the Quantized Image which indicate the DCT values at each 8 x 8 block)*

Beyond the Lab Experiment

1. Improvise the code by including Huffman Coding technique.

Part II

LabVIEW

Using LabVIEW for Image Processing

Prerequisites :
Knowledge in the following is mandatory

- Building own programs in LabVIEW.
- Distinguishing between LabVIEW's different data types.
- Create sub-VIs.
- Read and understand the LabVIEW Help files.

LabVIEW is a programming environment, developed by the company National Instruments.

LabVIEW uses a dataflow programming model that frees programmers from the sequential architecture of text-based programming, where instructions determine the order of program execution. LabVIEW is programmed using a graphical programming language, G, that uses icons instead of lines of text to create applications. The graphical code is highly intuitive for engineers and scientists familiar with block diagrams and flowcharts. The flow of data through the nodes (icons) in the program determines the execution order of the functions, allowing users to easily create programs that execute multiple operations in parallel. The parallel nature of LabVIEW also makes multitasking and multithreading simple to implement.

Some of the most commonly used nodes for Image Processing are listed below for ready reference.

1. **IMAQ Create** Creates a temporary memory location for an image. Use IMAQ Create in conjunction with the IMAQ Dispose VI to create or dispose of NI Vision images in LabVIEW.

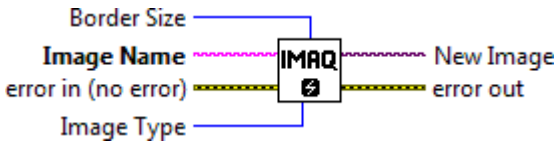

Figure 9.2: IMAQ Create

2. **IMAQ Read File** Reads an image file. The file format can be a standard format (BMP, TIFF, JPEG, JPEG2000, PNG, and AIPD) or a nonstandard format known to the user. In all cases, the read pixels are converted automatically into the image type passed by Image.

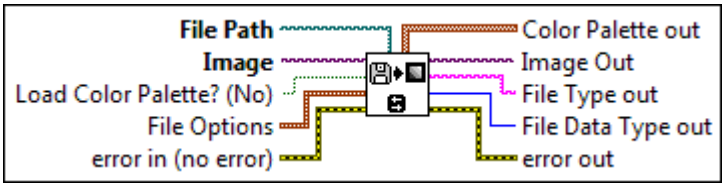

Figure 9.3: IMAQ ReadFile

3. **IMAQ Get Image Info** Gives different characteristics of the image.

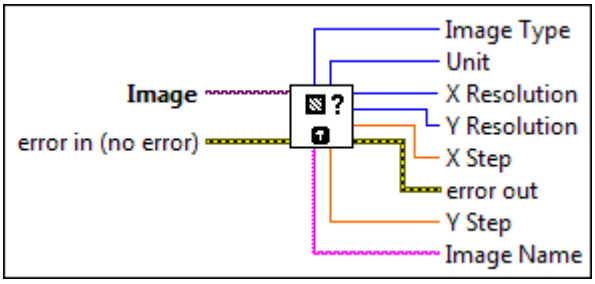

Figure 9.4: IMAQ Get Image Info

4. **IMAQ WindDraw** Displays an image in an image window. The image window appears automatically when the VI is executed.

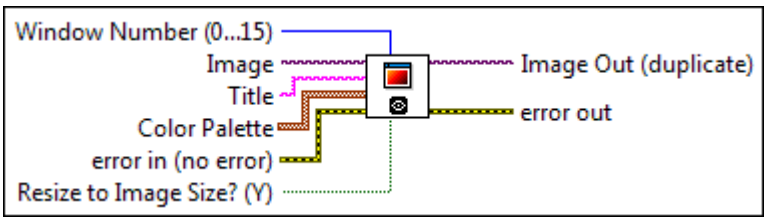

Figure 9.5: IMAQ WindDraw

5. **IMAQ Image to Array** Extracts (copies) the pixels from an image, or part of an image, into a LabVIEW 2D array. This array is encoded in 8 bits, 16 bits, or floating point, as determined by the type of input image. Refer Figure 9.6.

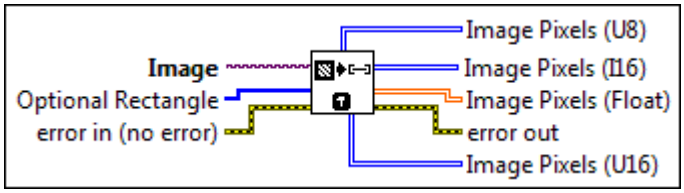

Figure 9.6: IMAQ Image to Array

LabVIEW VI's can be created for the following purposes :

- Transform images using predefined or custom lookup tables, change the contrast information in an image, invert the values in an image, and segment the image
- Filter images to enhance the information in the image. Use these VIs to smooth your image, remove noise, and find edges in the image. You can use a predefined filter kernel or create custom filter kernels
- Perform morphological basic morphological operations, such as dilation and erosion, on grayscale and binary images. Other VIs improve the quality of binary images by filling holes in particles, removing particles that touch the border of an image, removing noisy particles, and removing unwanted particles based on different characteristics of the particle
- Compute the histogram information and grayscale statistics of an image, retrieve pixel information and statistics along any one-dimensional profile in an image, and detect and measure particles in binary images
- Perform basic processing on color images; compute the histogram of a color image; apply lookup tables to color images; change the brightness, contrast, and gamma information associated with a color image; and threshold a color image
- Perform arithmetic and bit-wise operations in NI Vision; add, subtract, multiply, and divide an image with other images or constants or to apply logical operations and make pixel comparisons between an image and other images or a constant
- perform frequency processing and other tasks on images; convert an image from the spatial domain to the frequency domain using a two-dimensional Fast Fourier Transform (FFT) and convert an image from the frequency domain to the spatial domain using the inverse FFT. These VIs also extract the magnitude, phase, real, and imaginary planes of the complex image

Experiment 10

Basic Processing of Digital Images

10.1 Image Properties & Pixel Distance

Objective

For a given color ($\mathbb{R}^{m\times n\times 3}$) or gray scale image ($\mathbb{R}^{m\times n}$), display the following in the Front Panel of LabVIEW.

1. The type of image in which LabVIEW is operating.
2. The number of pixels along the x and y direction respectively.
3. The name of the temporary variable in which LabVIEW has stored the image.
4. Obtain the Horizontal and vertical Distance between two adjacent pixels in a given image. (Distance between one pixel and the other).

10.2 Re-Sampling / Resolution Modification

Objective

For a given gray scale image ($\mathbb{R}^{m\times n}$), display the following using LabVIEW.

1. Display the Resolution of the Image.
2. Re-sample / change the resolution of the image to a higher and lower level and thereby display the same.
3. Store the re-sampled image to a destination file with specific extension.
4. Ensure if the pixel distance is the same or different for the input and re-sampled image.

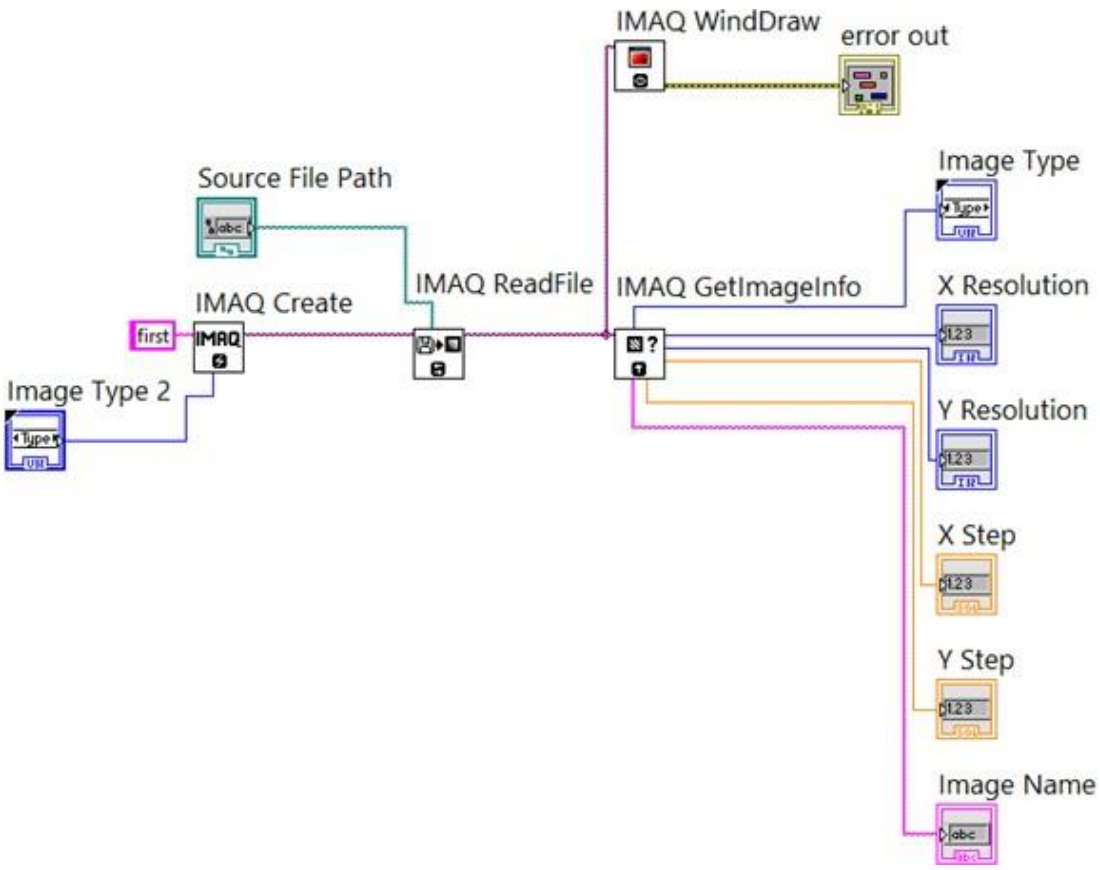

Figure 10.1: LabVIEW Code for Obtaining the Image Properties

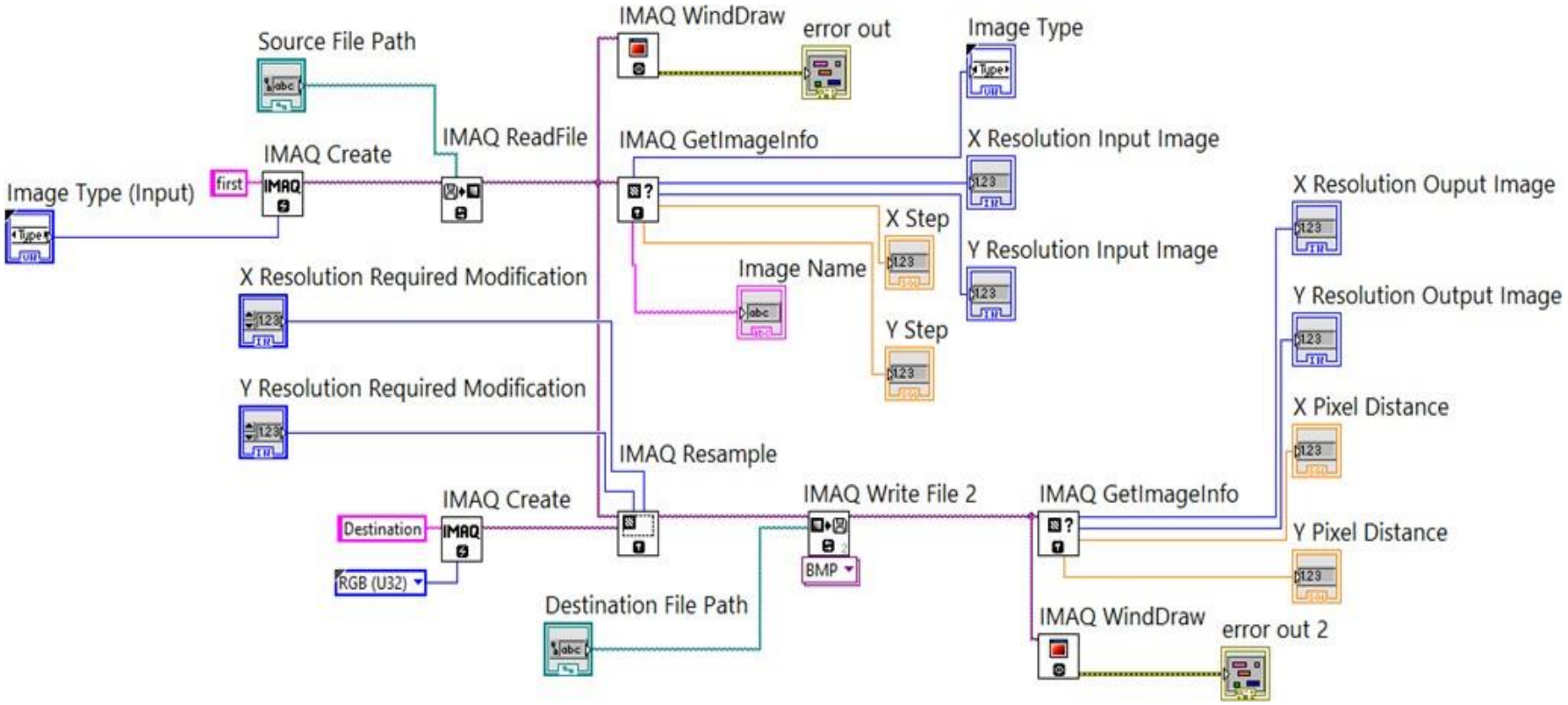

Figure 10.2: LabVIEW Code for Image Re-sampling and storing

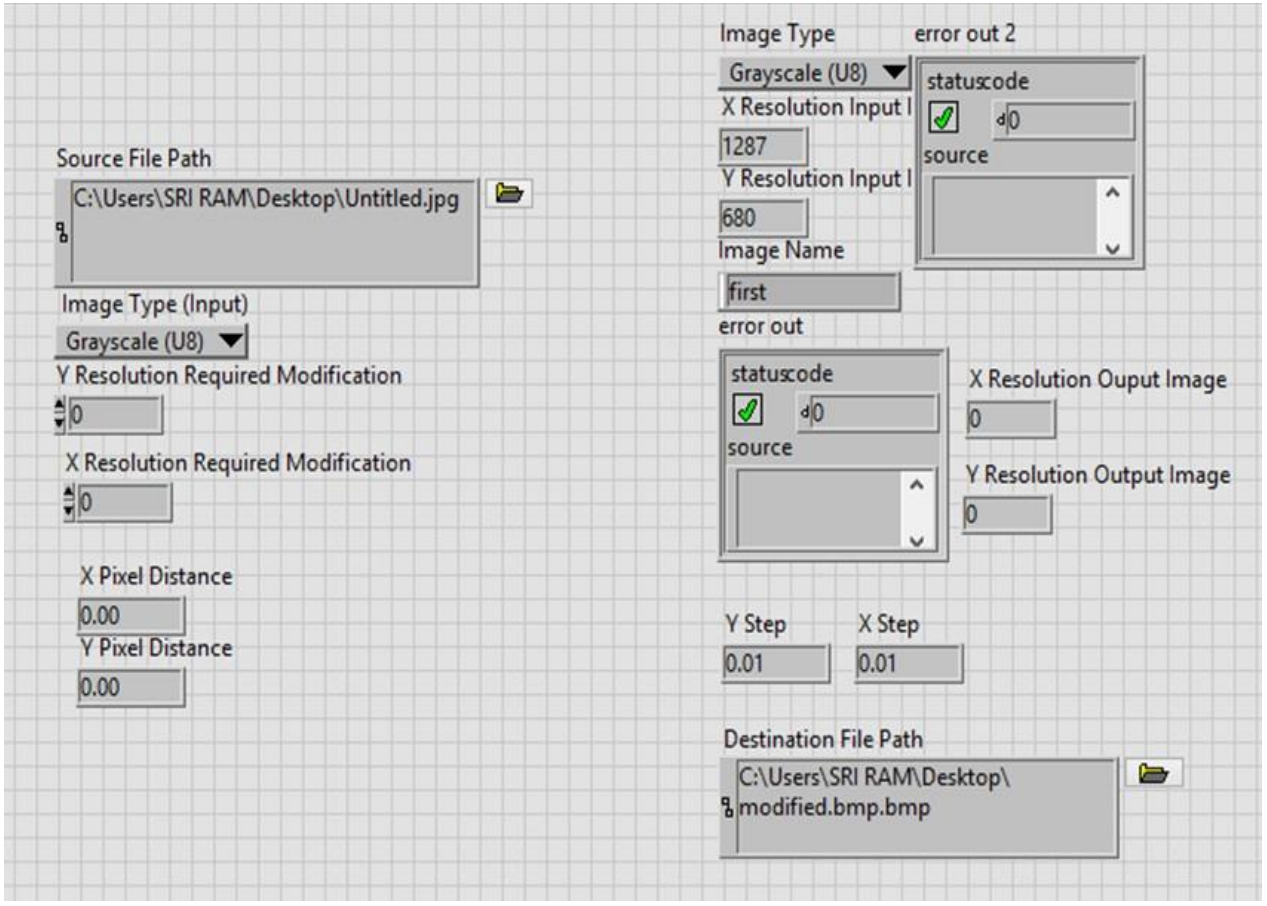

Figure 10.3: LabVIEW Front Panel - Image Re-sampling

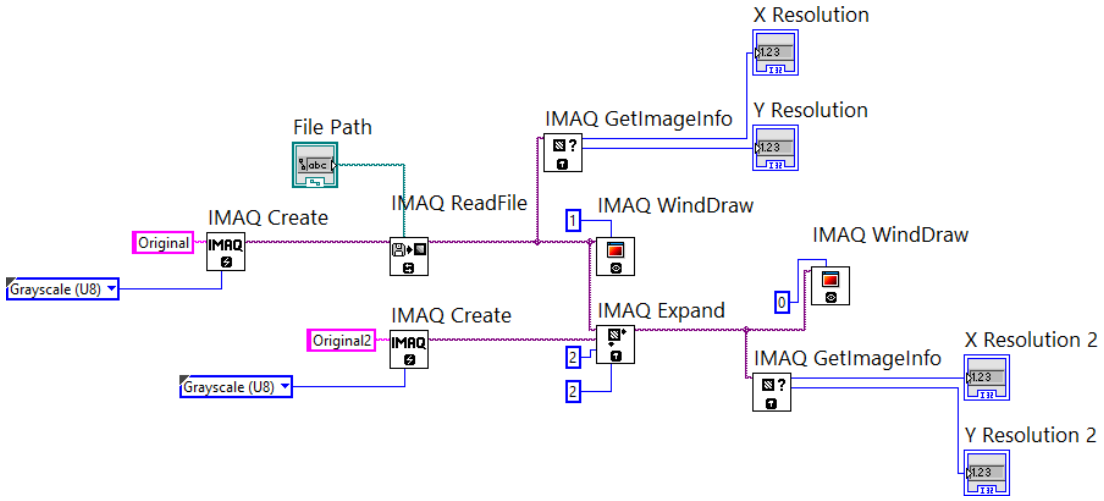

Figure 10.4: LabVIEW Front Panel - Image Zooming(Super-resolution)

Experiment 11

Image Arithmetic

11.1 Image arithmetic Processing

Objective

Given two gray scale images ($\mathbb{R}^{m\times n}$), of the *same size* display the following using LabVIEW.

1. Perform basic operations such as addition, subtraction, multiplication, and division of these two images.
2. Displays the output of the operations and interpret the results.

Beyond the Lab Experiment

1. Modify the program in such a way that, if there are two images of different size, then resample the smaller image to the size of the bigger one and then carry out the operations.
2. Try carrying out the same set of operations on a color image (RGB format in LabVIEW) and interpret the results that are obtained.

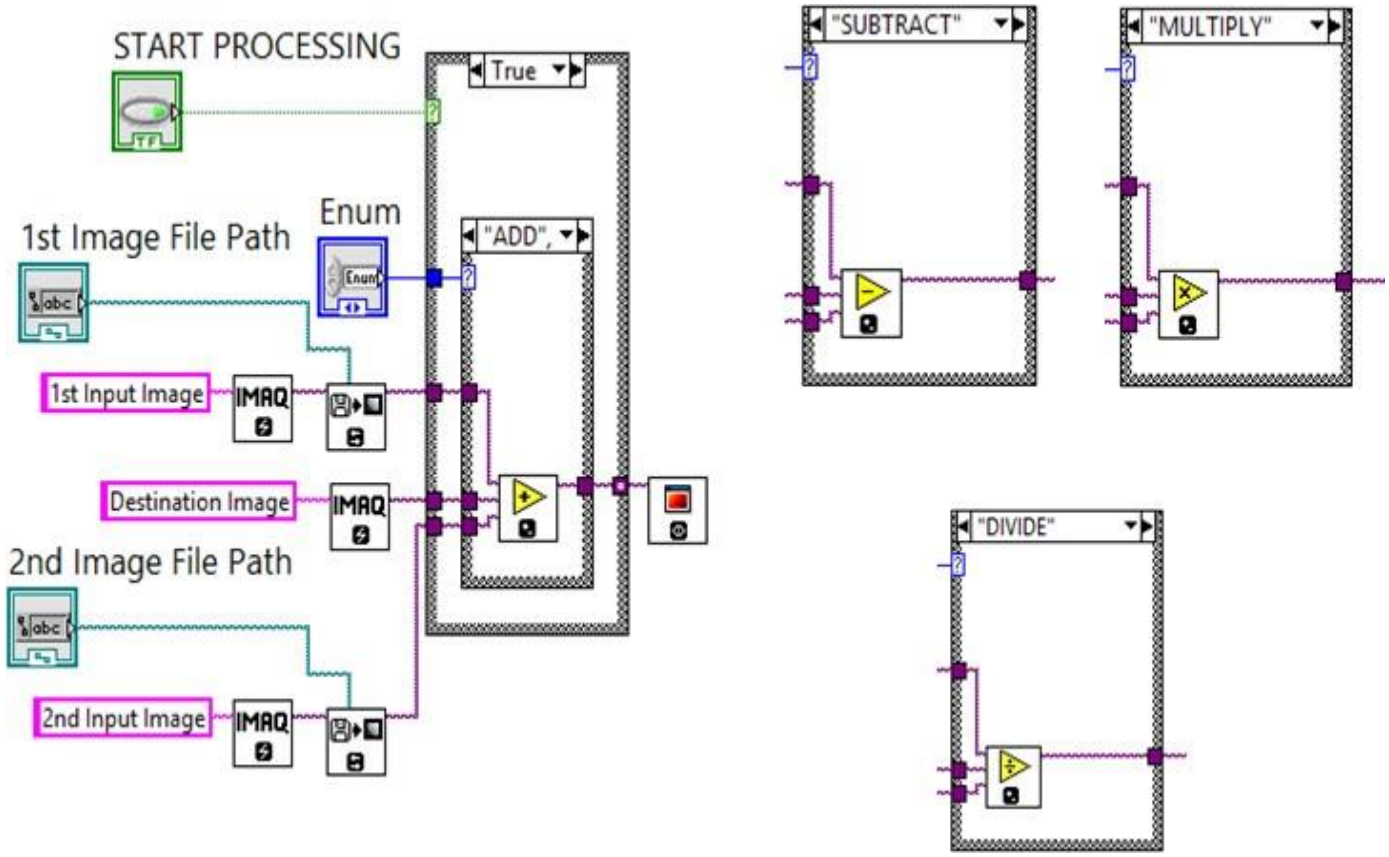

Figure 11.1: LabVIEW code for performing Image Arithmetic. The Three case structures on the right are a part of the inner case structure in the code available in the left

11.2 Scalar Image Processing

Objective

For a given gray scale image ($\mathbb{R}^{m \times n}$), display the following using LabVIEW.

1. Perform basic arithmetic operations such as addition, subtraction, multiplication, and division using a scalar value say 128.
2. Display the output of the operations and interpret the results.

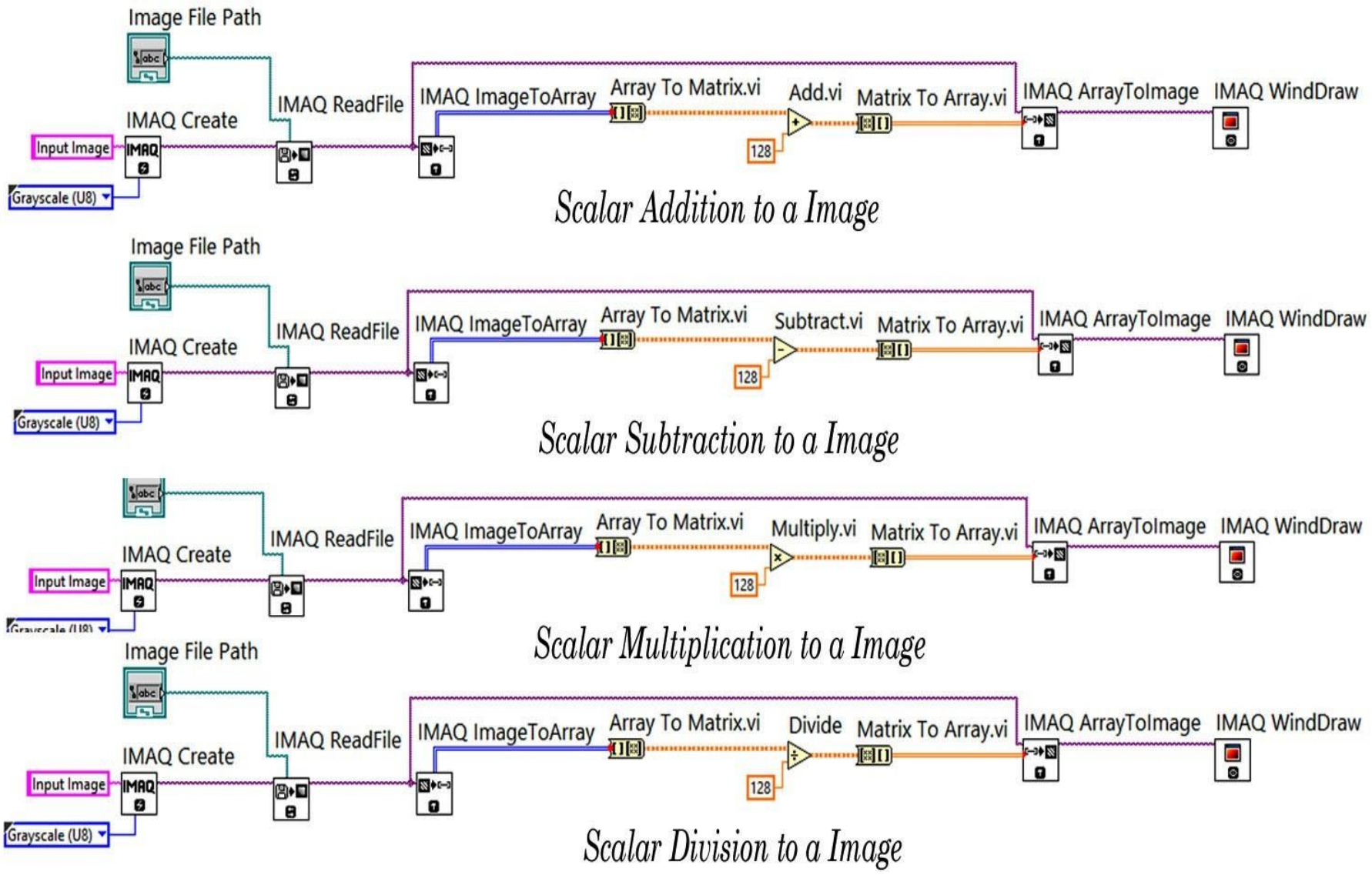

Figure 11.2: LabVIEW codes for scalar processing on a Image

Experiment 12

Basic Color Image Processing

12.1 Plane Extraction

Objective

Given a gray scale image ($R^{m \times n}$),display the following using LabVIEW.

1. Take up an image of any size.
2. Ensure that it is stored in RGB /HSI format in LabVIEW.
3. Extract each of the planes from the given image
   - Red, Green, Blue Planes
   - Hue, Saturation and Intensity planes
4. Interpret the resultant images based on the planes.

Beyond the Lab Experiment

1. Modify the code in such a way that when the user chooses any one of the color planes the output image is shown as a dominance of the corresponding color. (i.e.) If the red plane is chosen then the output image must be red shaded.

Question to Think Upon

You may note that during the process of extracting the color planes (i.e.) Red, Green and Blue Planes the output is only shown as a gray scale image. Why is that so?

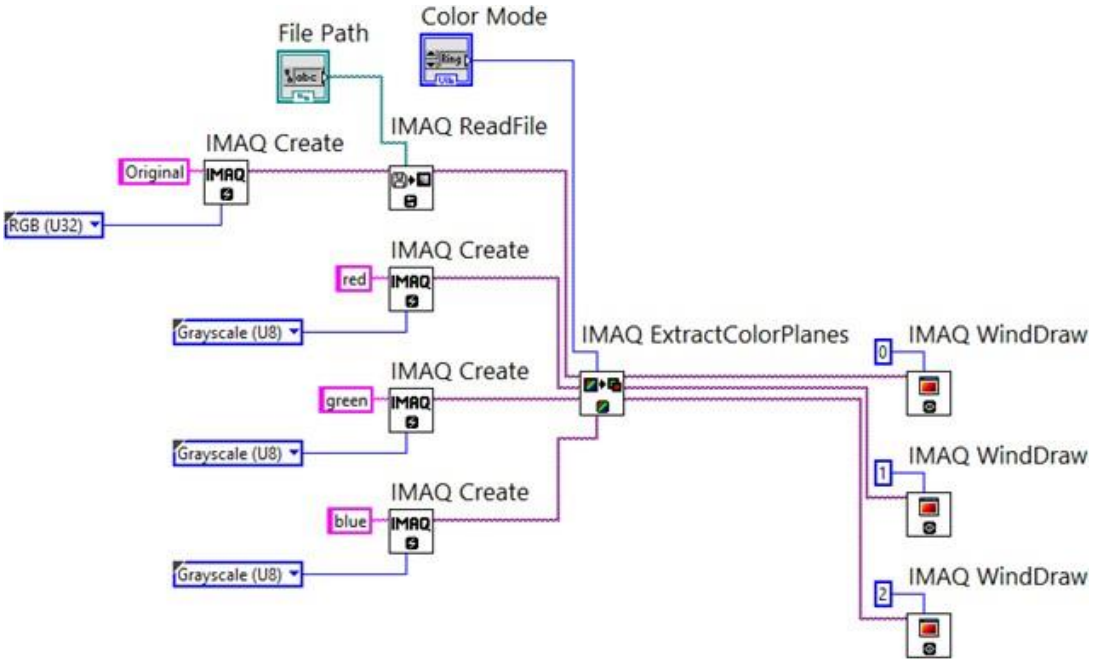

Figure 12.1: LabVIEW Code for Red, Green and Blue Plane Extraction

12.2 Color Plane Enhancement

Objective

Given a gray scale image ($R^{m \times n}$),display the following using LabVIEW.

1. Extract any one of the planes from the given image, say the red plane.
2. Enhance the pixel values of the obtained plane with a constant.
3. Interpret the output image.

Beyond the Lab Experiment - Plane Enhancement

1. Modify the code in such a way that the user can simultaneously change values for all the planes and visualise the change in the output image.

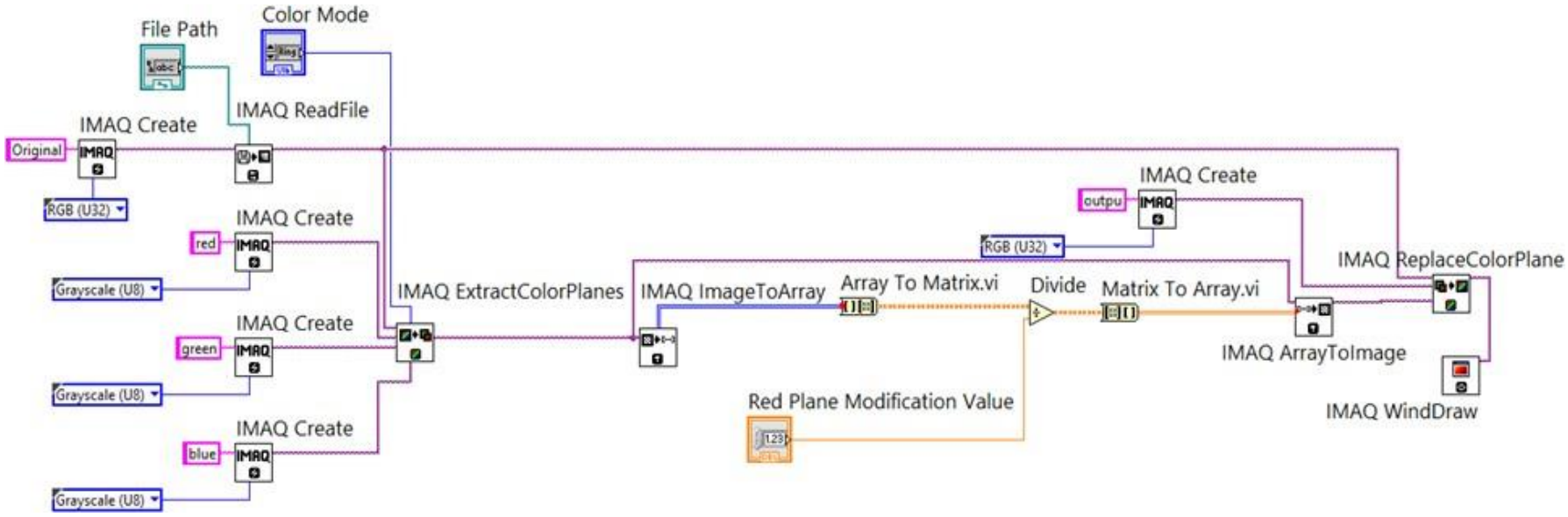

Figure 12.2: LabVIEW Code for Color Plane Enhancement

Experiment 13

Transforms for Image Processing

13.1 Fast Fourier Transform

Objective

Given a gray scale image ($\mathbb{R}^{m \times n}$),display the following using LabVIEW.

1. Store the image in Complex ($\mathbb{C}$) form within LabVIEW.
2. Compute the FFT of the given image.
3. Interpret the output image.
4. Try reconstructing the image from the FFT co-efficients.

Beyond the Lab Experiment

1. Rotate the original image to a certain angle using a picture editor. Determine if the FFT computed for the image is same as that of the previous one or different. Interpret the output image.
2. From the FFT image compute Inverse FFT and check if the original image is reconstructed.

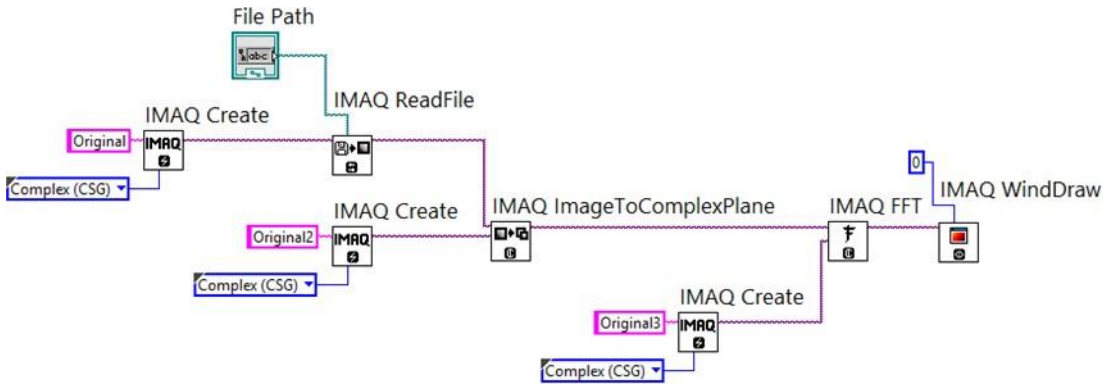

Figure 13.1: LabVIEW Code for Computing the FFT of the Image

13.2 Discrete Wavelet Transform

Objective

Given a gray scale image ($R^{m \times n}$),display the following using LabVIEW.

1. Extract the LL, LH, HL and HH features of Discrete Wavelet Transform.
2. Display the images and hence interpret the images.

Beyond the Lab Experiment

1. Extract one more level of DWT and hence interpret the output image with that of the previously obtained LL, LH, HL and HH images.

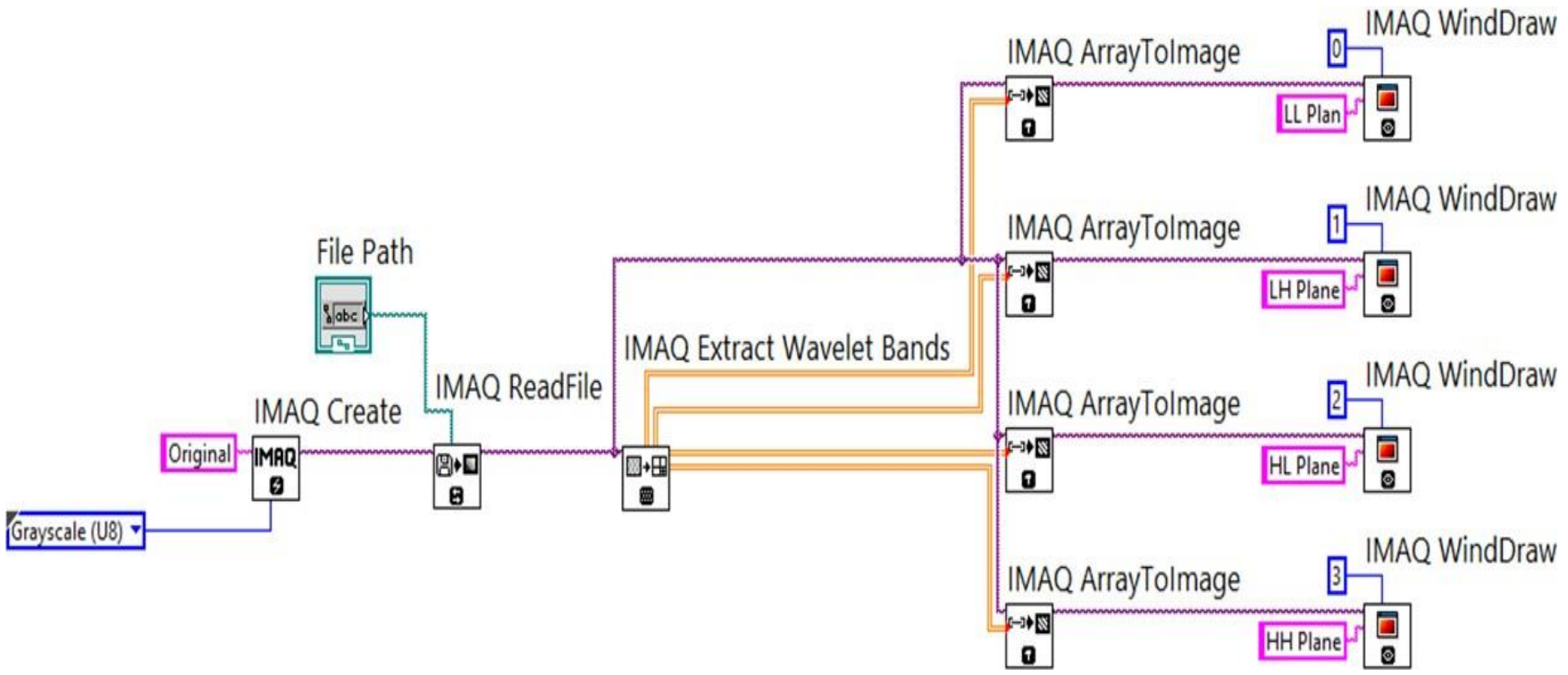

Figure 13.2: LabVIEW Code for Discrete Wavelet Transform

13.3 Discrete Cosine Transform

Objective

Given a gray scale image ($\mathbb{R}^{m \times n}$),display the following using LabVIEW.

1. Compute the DCT of the given image.
2. Interpret the output image.
3. Try reconstructing the image from the DCT co-efficients.

Beyond the Lab Experiment [1]

1. The LabVIEW code given is only for computing the DCT. Try to compute inverse DCT.
2. Using DCT, try coding the JPEG compression method.

[1] Not required for *Record Work*

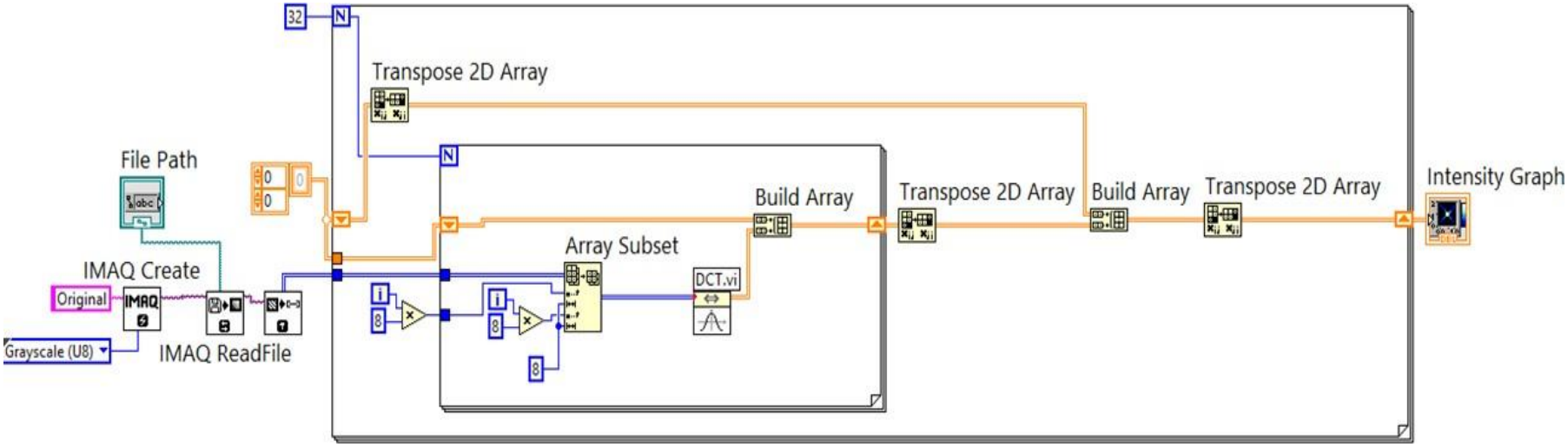

Figure 13.3: LabVIEW Code for computing Discrete Cosine Transform

Experiment 14

Edge Detection

Objective

Given a gray scale image ($\mathbb{R}^{m \times n}$),display the following using LabVIEW.

1. Perform the various edge detection operations using **IMAQ Edge Detection** and **IMAQ Canny Edge Detection** Nodes of LabVIEW.
2. Interpret the results.

14.1 Canny Edge Detection

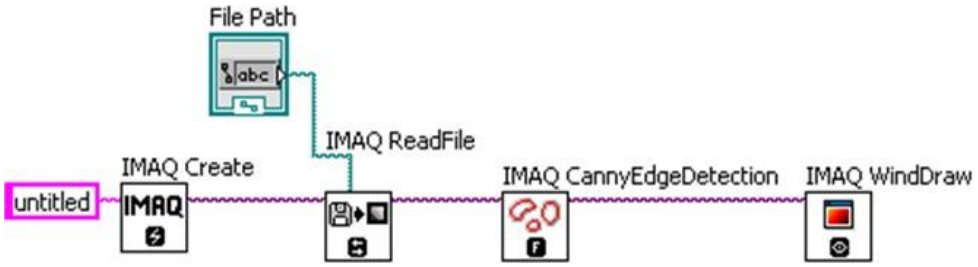

Figure 14.1: LabVIEW Code for Canny Edge Detection

Figure 14.2: Canny Edge Detection Output

14.2 Prewitt Edge Detection

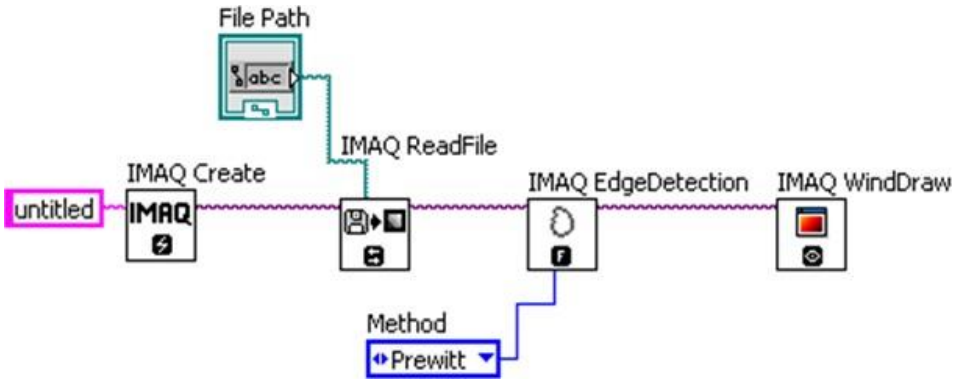

Figure 14.3: LabVIEW code for Prewitt Edge Detection

Figure 14.4: Prewitt Edge Detection Output

Beyond the Lab Experiment

1. Perform Edge Detection without using the **IMAQ Edge Detection** and **IMAQ Canny Edge Detection** Nodes of LabVIEW. [Hint: Use the node **IMAQ Convolve** and feed the node with the respective masks].

Experiment 15

1^{st} Order Statistical Features and Histogram

Objective

Given a gray scale image ($\mathsf{R}^{m \times n}$),display the following using LabVIEW.

1. Evaluate the First Order Statistical Features such as Mean, Standard Deviation and Variance.
2. Display the Histogram of the Image.

Mean of the Image

The average of sum of all the values in the image f: $\mathsf{R}^{m \times n}$.

$$Mean = \frac{1}{m \times n} \times \sum_{\forall x,y} f(x, y)$$

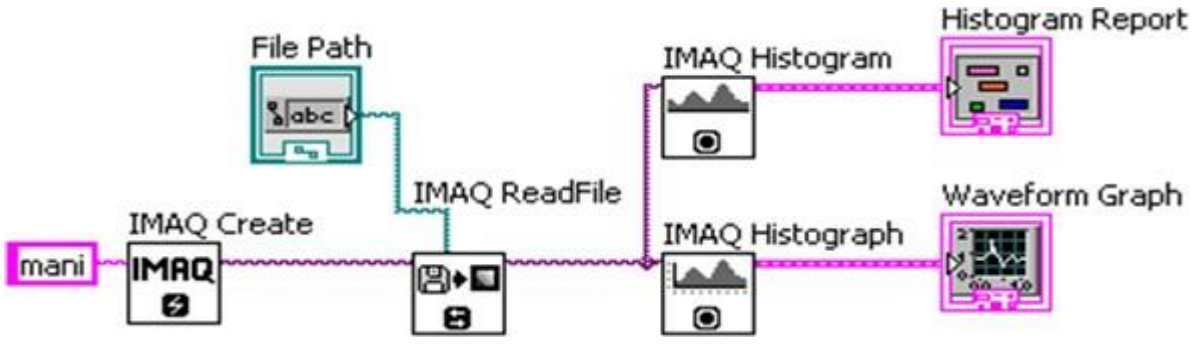

Figure 15.1: LabVIEW Code for First Order Statistical Features and Histogram Display

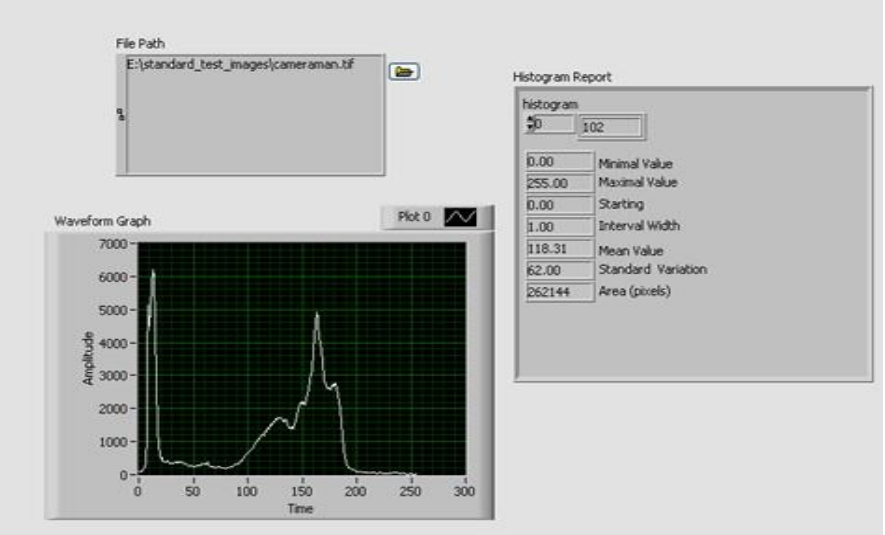

Figure 15.2: LabVIEW Front Panel Display

Figure 15.3: Input '*cameraman*' Image

Beyond the Lab Experiment

1. The histogram report in the front panel computes only the standard deviation. Compute the Variance of the given image.
2. Increase the brightness of the given image and hence repeat the experiment.
3. In the experiment 10, re-sample an image to upper and lower dimensions and hence repeat the experiment.

Viva Questions

1. Define hue and saturation of an image.
2. Define brightness and contrast of an image.
3. What is image acquisition system?
4. What is weber ratio?
5. What is dithering of an image?
6. What are the different methods for estimating the pixel distance?
7. What is the resolution of an image?
8. What is the application of sampling?
9. How many components are present in a colour image?
10. What is Mach band defect?
11. What are the applications of image addition & subtraction?
12. What is image averaging?
13. What is the function of spatial filtering?
14. Give function of medium filter & sharpening filter.
15. What are the applications of High boost filter?
16. What is the necessity for enhancing an image?
17. What is contrast stretching?
18. How do you obtain the negative of an image?
19. What do you mean by spatial domain?
20. What is grey level slicing?
21. Why do we need transforms?
22. What are the various available transforms?
23. What is the application of DFT?
24. What is the application of DWT?
25. What are the properties of 2D DFT?