IMAGE

A gray scaled image refers to a two dimensional light intensity function *f(x,y)* where x and y are spatial coordinates and the value of f at any point (x,y) is proportional to the brightness or gray-level of the image at that point. A digital image is an image *f(x,y)* that has been discretized both in spatial coordinates and brightness. The pixel values are always positive and finite in magnitude.

IMAGE TYPES

The Image Processing Toolbox of matlab supports four basic types of images:

• Index images
• Intensity images
• Binary images
• RGB images

This section discusses how MATLAB and the Image Processing Toolbox represent each of these image types.

Indexed Images

An indexed image consists of a data matrix, X, and a colormap matrix, map. The data matrix can be of class uint8, uint16, or double. The colormap matrix is an m-by-3 array of class double containing floating-point values in the range [0,1]. Each row of map specifies the red, green, and blue components of a single color. An indexed image uses direct mapping of pixel values to colormap values. The color of each image pixel is determined by using the corresponding value of X as an index into map. The value 1 points to the first row in map, the value 2 points to the second row, and so on. A colormap is often stored with an indexed image and is automatically loaded with the image when use with the imread function.

Intensity Images

An intensity image is a data matrix, I, whose values represent intensities within some range. In matlab an intensity image is a single matrix, with each element of the matrix corresponding to one image pixel. The matrix can be of class double, uint8, or uint16. While intensity images are rarely saved with a colormap, MATLAB uses a colormap to display them. The elements in the intensity matrix represent various intensities, or gray levels, where the intensity 0 usually represents black and the intensity 1 represents 255, or 65535 usually represents full intensity, or white.

Binary Images

In a binary image, each pixel assumes one of only two discrete values. Essentially, these two values correspond to on and off. A binary image is stored as a two-dimensional matrix of 0's (off pixels) and 1's (on pixels). In matlab a binary image can be considered a special kind of intensity image, containing only black and white. Other interpretations are possible, however. Binary image can be interpreted as an indexed image with only two colors. A binary image can be stored in an array of class double or uint8. An array of class uint8 is generally preferable to an array of class double, because a uint8 array uses far less memory. In the Image Processing toolbox of matlab, any function that returns a binary image returns it as a uint8 logical array. The matlab toolbox uses a logical flag to indicate the data range of a uint8 logical array. If the logical flag is on, the data range is [0,1]; if the logical flag is off, the toolbox assumes the data range is [0,255].

RGB Images

An RGB image, sometimes referred to as a truecolor image, is stored in matlab as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel. RGB images do not use a palette. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million colors. The precision with which a real-life image can be replicated has led to the commonly used term truecolor image. An RGB matlab array can be of class double, uint8, or uint16. In an RGB array of class double, each color component is a value between 0 and 1. A pixel whose color components are (0,0,0) displays as black, and a pixel whose color Components are (1,1,1) displays as white. The three color components for each pixel are stored along the third dimension of the data array. For example, the red, green, and blue color components of the pixel (10,5) are stored in RGB(10,5,1), RGB(10,5,2), and RGB(10,5,3), respectively.

COORDINATE SYSTEM

Locations in an image can be expressed in various coordinate systems, depending on context. The main coordinate system normally used is PIXEL COORDINATE SYSTEM. Because, the most convenient method for expressing locations in an image is to use pixel coordinates. In this coordinate system, the image is treated as a grid of discrete elements, ordered from top to bottom and left to right.

For pixel coordinates, the first component r (the row) increases downward, while the second component c (the column) increases to the right. Pixel coordinates are integer values and range between 1 and the length of the row or column. There is a one-to-one correspondence between pixel coordinates and the coordinates Matlab uses for matrix subscripting. This correspondence makes the relationship between an image's data matrix and the way the image displays easy to understand. For example, the data for the pixel in the fifth row, second column is stored in the matrix element (5,2).

BASIC MATLAB FUNCTIONS USED IN THIS PROJECT

Before discussing the project and proposed work in detail, some of the basic matlab function used in this project, are described here.

1. **imread**

Imread function reads read image from some graphics file.

A = imread(filename , fmt)

reads the image in filename into A, whose class is uint8.

If the file contains a grayscale intensity image, A is a two-dimensional array. If the file contains a truecolor (RGB) image, A is a three-dimensional (M-by-N-by-3) array. Filename is a string that specifies the name of the graphics file, and FMT is a string that specifies the format of the file. The file must be in the current directory or in a directory on the matlab path. If imread cannot find a file named filename, it looks for a file named filename.fmt.

The possible values for FMT include:

'bmp'	Windows Bitmap (BMP)
'hdf'	Hierarchical Data Format (HDF)
'jpg' or 'jpeg'	Joint Photographic Experts Group (JPEG)
'pcx'	Windows Paintbrush (PCX)
'tif' or 'tiff'	Tagged Image File Format (TIFF)
'xwd'	X Window Dump (XWD)

[X,map] = imread(filename,fmt)

reads the indexed image in FILENAME into X and its associated colormap into map. X is of class uint8, and map is of class double. The colormap values are rescaled when they are read to have the range [0,1].

2. **<u>double</u>**

This function convert image data into double precision. Double(X) returns the double precision value for X. If X is already a double precision array, double has no effect. Double is called for the expressions in “for”, “if”, and “while” loops if the expression isn't already double precision. Double should be overloaded for all objects where it makes sense to convert it into a double precision value.

3. **<u>clip</u>**

Clip function clips a signal so laves fall between "lo" and "hi". For example,

y = clip(x, low, high)

Where,

x	=	input image
low	=	set everything below here to low
high	=	set everything above here to high
y	=	output image

4. **<u>imresize</u>**

Imresize, resizes an image of any type using the specified interpolation method. Supported interpolation methods includes:

'nearest'	nearest neighbor interpolation
'bilinear'	bilinear interpolation

'bicubic'	bicubic interpolation

b = imresize(a, m, method)

returns an image that is m times the size of a. If m is between 0 and 1.0, b is smaller than a. If m is greater than 1.0, b is larger than a. If method is omitted, imresize uses nearest neighbor interpolation as default. The input image can be of class uint8 or double. The output image is of the same class as the input image.

5. **<u>conv2</u>**

Conv2 performs two dimensional convolution on its input data. For example.

C = CONV2(A, B)

performs the 2-D convolution of matrices A and B. If

[ma, na] = size(A)

and

[mb, nb] = size(B)

then

size(C) = [ma+mb-1,na+nb-1].

Similarly, if we use,

C = CONV2(... ,'shape')

returns a subsection of the 2-D convolution with size specified by shape. Possible values of shapes are:

'full'	returns the full 2-D convolution,
‘same’	returns the central part of the convolution
'valid'	returns only those parts of the convolution

It should be noted that conv2 is fastest when size(A) > size(B).

6. **<u>show_img</u>**

Show_img, as it name implies display an image with possible scaling. For example, show_img(img, figno, scaled, map)

Where,

Img	=	input image
figno	=	figure number to use for the plot if 0, re-use the same figure if omitted a new figure will be opened
scaled	=	1 (TRUE) to do auto-scale (DEFAULT) not equal to 1 (FALSE) to inhibit scaling
map	=	user-specified color map

scaled and map, are optional arguments.

7. **truesize**

Truesize adjust display size of image. For example,

TRUESIZE (FIG,[MROWS NCOLS])

adjusts the display size of an image. FIG is a figure containing a single image or a single image with a colorbar. [MROWS MCOLS] is a 1-by-2 vector that specifies the requested screen area (in pixels) that the image should occupy.

TRUESIZE (FIG) uses the image height and width for [MROWS MCOLS]. This results in the display having one screen pixel for each image pixel. If you omit the figure argument, TRUESIZE works on the current figure.

8. fft2

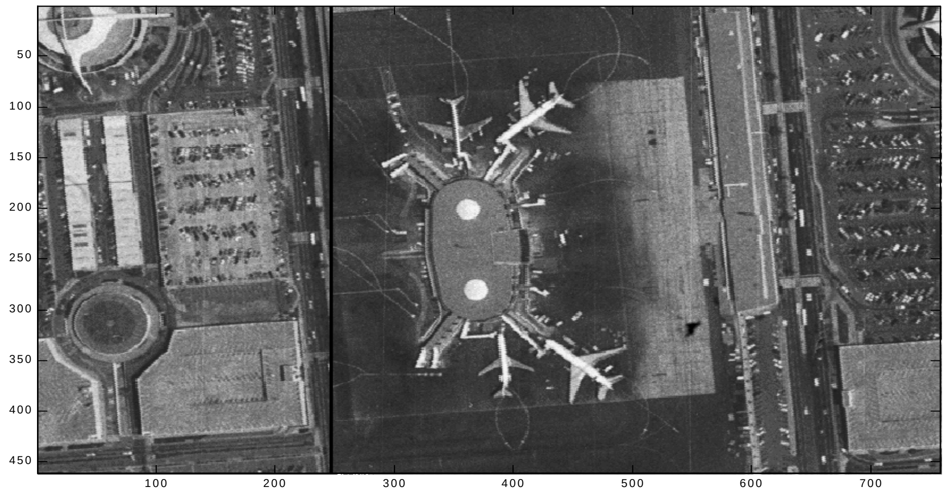
fft2 is basically two dimensional discrete Fourier Transform. FFT2(X) returns the two-dimensional Fourier transform of matrix X. If X is a vector, the result will have the same orientation. If we also specify the size along with argument X than it return the specified size, if size of X is smaller than that one, than it zero pad the remaining entries.

9. <u>fftshift</u>

'fftshift' shifts the dc component to center of spectrum. For vectors, fftshift(x) swaps the left and right halves of x. For matrices, fftshift(x) swaps the first and third quadrants and the second and fourth quadrants. For N-dimensional arrays, fftshift(x) swaps "half-spaces" of x along each dimension. Fftshift is useful for visualizing the Fourier transform with the DC component in the middle of the spectrum.

PROJECT

Now, in this project, a digital image is given, named 'aifport1.tiff', given as,



Resolution of the given image is 461x758, and from this image we have to pick two small images say I_1 and I_2, each of 128x128, and than stretch I_1 and I_2 such that their resolution will becomes 512x512. So we stretch I_1 and I_2 four times of their original size, by inserting zeros. Now than by using following interpolation techniques, try to fill out values instead of that zeros. So basically project relates with image enhancement by using different techniques of interpolations.

ATTACKING ON THE PROBLEM

Now first of all, clear the memory by removing all previously stored variables on command window of matlab, as well as close all previously opened windows, by using statements:

clc;
clear all;
close all;

Than read the image, 'airport1.tif' from a graphics file and stored the image in a variable, lets say MyImage, using the statement,

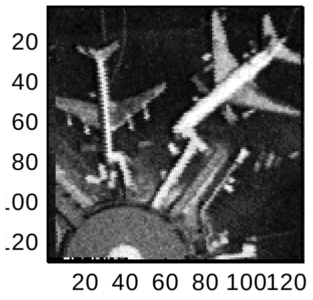
MyImage = imread('airport1.tif');

So as already explained, since airport1 is a gray scaled image so MyImage is a two dimensional matrix, whose values indicates the intensity levels, in the image airport1. After reading the image, change its precision to double precision form by using double function already described above.

original = double(MyImage);

In this project we have to enhance two portions of the image which are 128x128, so we pick those portions by using simple manipulations:

Picked = Original (70:197,80:207);



Now create a zero matrix of required size,

ImageZeros = zeros(512);

Then insert the pixel values in that zero image, hence in this way we have now a 512x512 image having all the pixel values, and there are three zeros between two consecutive pixel values.

The main task of the project is to interpolate the missing values, and using different types of interpolations including:

- Sinc Interpolation
- Triangular Interpolation
- Parabolic Interpolation
- ZeroOrderHold Interpolation

INTERPOLATION

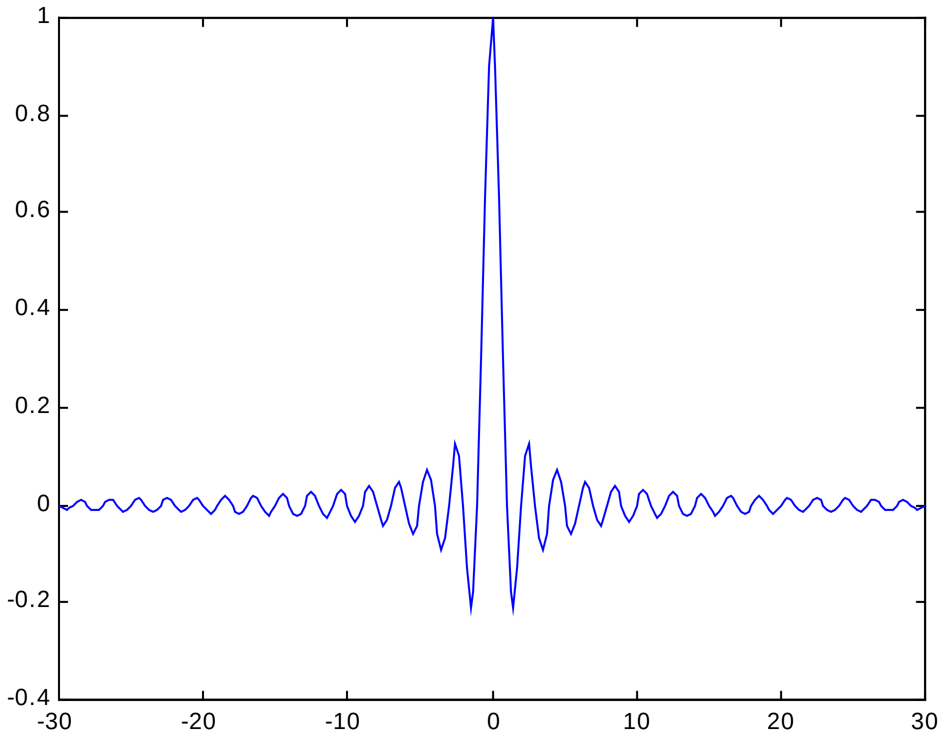
Interpolation is the process by which we estimate an image value at a location in between image pixels. For example, if you resize an image so it contains more pixels than it did originally, the software obtains values for the additional pixels through interpolation. The imresize and imrotate geometric functions use two-dimensional interpolation as part of the operations they perform. The interpolation methods all work in a fundamentally similar way. In each case, to determine the value for an interpolated pixel, you find the point in the input image that the output pixel corresponds to. So we can assign a value to the output pixel by computing a weighted average of some set of pixels in the vicinity of the point. The weightings are based on the distance each pixel is from the point.

The number of pixels considered affects the complexity of the computation. Therefore the bilinear method takes longer than nearest neighbor interpolation, and the bicubic method takes longer than bilinear. However, the greater the number of pixels considered, the more accurate the effect is, so there is a trade-off between processing time and quality. In all interpolation methods we apply here, sinc interpolation provide us the best results.

SINC INTERPOLATION

Sinc Interpolation, interpolate the values between two successive pixel values, and replace all the zeros by some values, such that those values fit on the sinc curve.

```
a = [-30:0.25:30];
y = sinc(a);
```



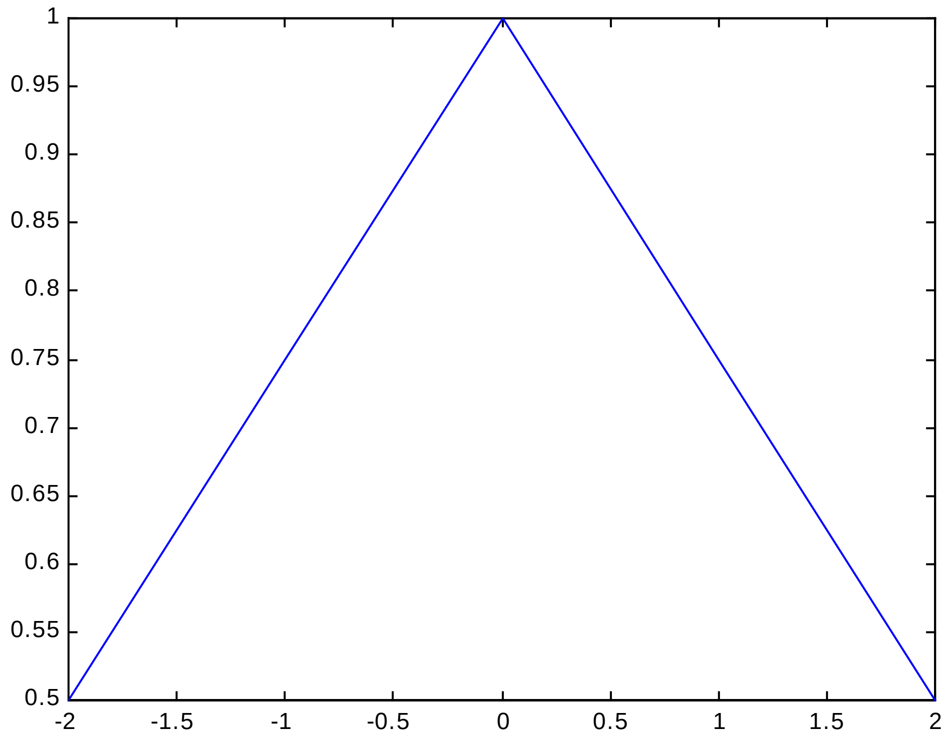
```
h_only = conv2(Img,y,'same');
y_conj = y';
SImage = conv2(h_only,y_conj,'same');
```

So first of all, create a sinc curve and than convolve that sinc curve with our expanded image, such that it interpolate and filling the zeros. Sinc Interpolated image is attached with this report.

TRIANGLULAR INTERPOLATION

Trinagular Interpolation, interpolate the values between two successive pixel values, and replace all the zeros by some values, such that those values fit on the triangular curve.

```
x = [-2:2];
y1 = 1  - (abs(x)./4);
```



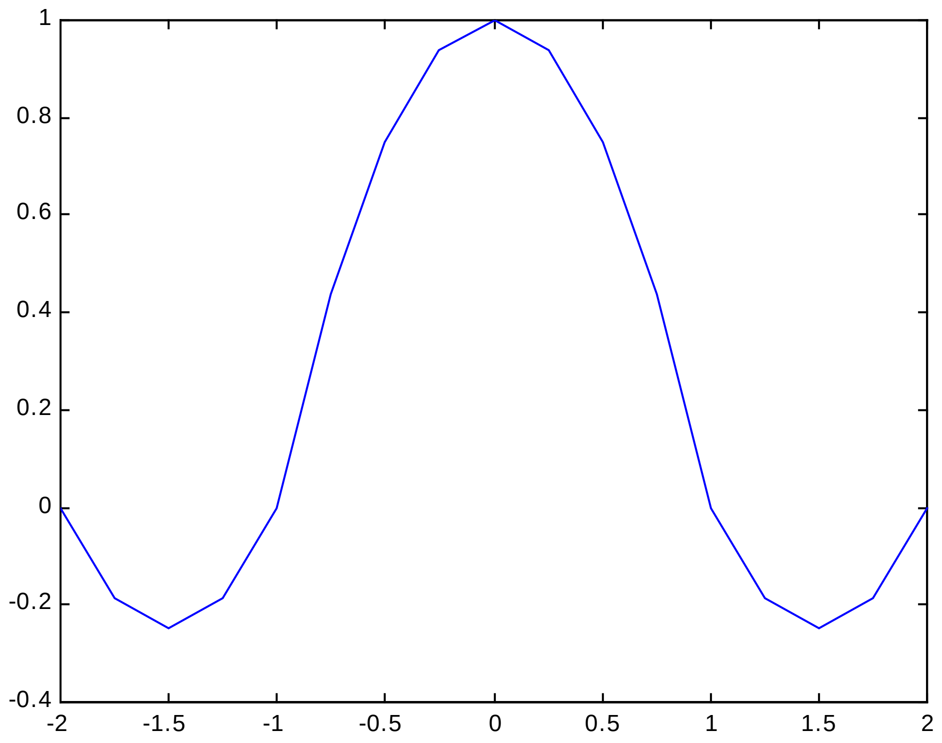
```
h_only1 = conv2(Img,y1,'same');
y_conj = y1';
TriImage = conv2(h_only1,y_conj,'same');
```

I use this filter to interpolate the expanded image. Interpolated Image is attached with this report.

PARABOLIC INTERPOLATION

Parabolic Interpolation, interpolate the values between two successive pixel values, and replace all the zeros by some values, such that those values fit on the a parabola. Upper bound of parabola is sinc pulse, while lower bound is of triangular waveform as specified before.

```
x1 = -2:0.25:-1;
y1 = -0.25+(x1+1.5).^2;
x2 = -0.75:0.25:1;
y2 = x2.^2;
y2 = 1 - y2;
x3 = 1.25:0.25:2;
y3 = -0.25+(x3-1.5).^2;
x = [x1 x2 x3];
y = [y1 y2 y3];
```



```
h_only = conv2(Img,y,'same');
```

```
y_conj = y';
ParaImage = conv2(h_only,y_conj,'same');
ParaImage = clip(ParaImage,0,255);
```

ZERO ORDER HOLD INTERPOLATION

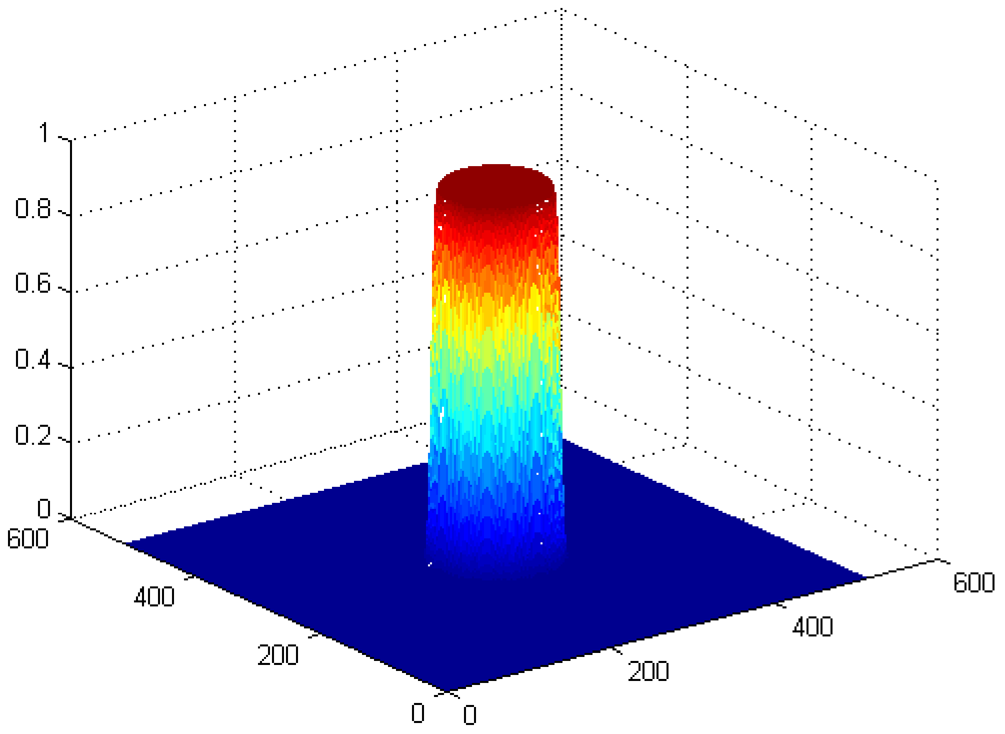
In zero order hold, we use a zero order hold filter which interpolates the values between two successive pixel values, and replace all the zeros by some values, such that the intensity level remain same across the filter length. Results of zero order hold interpolation is shown in the attached image.

FREQUENCY DOMAIN INTERPOLATION

We can also interpolate our image by using frequency domain analysis. So, in doing so, first of all convert the image in frequency domain by taking discrete Fourier transform, than process the image, by just multiplying the image with a low pass two dimensional filter, than taking inverse discrete Fourier transform, and recover the original image. So,

```
MyFilter = zeros(512);
for x = 1:512 for y = 1:512
MyFilter (x,y) = 1 / (1 + (sqrt((x-256)^2 + (y-
256)^2)/62)^40);
   end
end
MyFilter = fftshift(MyFilter);
Filter_img = fft2(Img,512,512);
MyFilteredImage = Filter_img .* MyFilter;
R_MFimage = abs(real(ifft2(MyFilteredImage)));
```

Now the filter we use looks like,



So we have an advantage in frequency domain, than we can reduce our work to a large extent, because in time domain we have to consider filters from negative infinity to positive infinity while in frequency domain we are concerned all about from $-\pi$ to $+ \pi$.

SOURCE CODE

```
clc
clear all;
close all;

MyImage = imread('airport1.tif');
Original = double(MyImage);
Picked = Original (70:197,80:207);
ImageZeros = zeros(512);
Img(1:4:512,1:4:512) = Picked;

%___________PREDEFINED FUNCTION________________________

LinearImage = imresize(Picked,4,'bicubic');

%____________SINC INTERPOLATION________________________



a = [-30:0.25:30];
y = sinc(a);
h_only = conv2(Img,y,'same');
y_conj = y';
SImage = conv2(h_only,y_conj,'same');



%____________TRIANGULAR INTERPOLATION__________________

x = [-2:2];
y1 = 1  - (abs(x)./4);
h_only1 = conv2(Img,y1,'same');
```

```
y_conj = y1';
TriImage = conv2(h_only1,y_conj,'same');
CTriImage = clip(TriImage,0,255);


%________ZERO ORDER HOLD INTERPOLATION_________________

samp_hold = ones(1,4);
h_only2 = conv2(Img,samp_hold,'same');
y_conj = ones(4,1);
ZeroOrderImage = conv2(h_only2,y_conj,'same');
CZeroOrderImage = clip(ZeroOrderImage,1,255);


%___________PARABOLIC INTERPOLATION____________________



x1 = -2:0.25:-1;
y1 = -0.25+(x1+1.5).^2;
x2 = -0.75:0.25:1;
y2 = x2.^2;
y2 = 1 - y2;
x3 = 1.25:0.25:2;
y3 = -0.25+(x3-1.5).^2;
x = [x1 x2 x3];
y = [y1 y2 y3];

h_only = conv2(Img,y,'same');
y_conj = y';
ParaImage = conv2(h_only,y_conj,'same');
ParaImage = clip(ParaImage,0,255);
```

```
%__________FREQUENCY DOMAIN INTERPOLATION_____________

MyFilter = zeros(512);
for x = 1:512 for y = 1:512
      MyFilter (x,y) = 1 / (1 + (sqrt((x-256)^2 + (y-
256)^2)/62)^40);
   end
end
MyFilter = fftshift(MyFilter);
Filter_img = fft2(Img,512,512);
MyFilteredImage = Filter_img .* MyFilter;
R_MFimage = abs(real(ifft2(MyFilteredImage)));


%______________PLOTTING SINC FUNCTION__________________



%figure(1);
%subplot(2,1,1);
%show_img(Img,1);
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(SImage,1);
xlabel('SINC');


%_________PLOTTING TRIANGULAR FUNCTION_________________

%figure(2);
%subplot(2,1,1);
%show_img(Img,2);
```

```
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(TriImage,2);
xlabel('TRIANGULAR');


%__________PLOTTING ZERO ORDER FUNCTION_________________

%figure(3);
%subplot(2,1,1);
%show_img(Img,3);
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(ZeroOrderImage,3);
xlabel('ZERO ORDER HOLD');



%_________PLOTTING PARABOLIC IMAGE______________

%figure(4);
%subplot(2,1,1);
%show_img(Img,4);
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(ParaImage,4);
xlabel('PARABOLIC INTERPOLATION');
```

```
%____________PLOTTING BI CUBIC IMAGE___________________

%figure(5);
%subplot(2,1,1);
%show_img(Img,5);
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(LinearImage,5);
xlabel('BI CUBIC INTERPOLATION');

%_____PLOTTING FREQUENCY DOMAIN INTERPOLATED IMAGE______

%figure(6);
%subplot(2,1,1);
%show_img(Img,6);
%xlabel('PICKED IMG.');

%subplot(2,1,2);
show_img(R_MFimage,6);
xlabel('FREQUENCY DOMAIN INTERPOLATION');
```