Aim(Part-1):

Implement Low pass Filter and High Pass filter in frequency domain.

Image Filter in frequency domain:

To apply image filter in frequency domain , first you have to apply fourier transform on given Image. That will give you representation of image in frequency domain. Now if you want low frequencies data from that image you have to multiply it with low pass filter that is guassian filter.Or if you want to high frequencies then you have to multiply it with high pass filter then it will give you all high frequencies components. Then again to get back that filter image apply inverse fourier transform that gives you filtered Image.

ALGORITHM:

1) For given Image, apply fft2(Image) Fourier transform over Image.
2) If filter choice is low_pass_filter than multiply it to low-pass filter else multiply it to high pass filter.
3) To get frequency spectrum apply fftshift() function to that filtered image.
4) Apply inverse fourier transform ifft2(Image_filtered) to get back filterd image.
5) I compare the results using two different low pass thresholds. One for 7 and another one for threshold value 9.

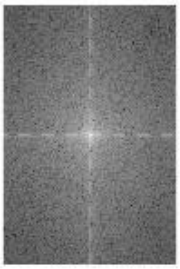
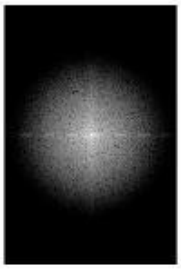
Code:

```
img=imread(image);

%Convert to grayscale

img=rgb2gray(img);

subplot(2,2,1), imshow(img)


%Determine good padding for Fourier transform

padd = paddedsize(size(img));


%Create a Gaussian Lowpass filter 5% the width of the Fourier transform


D0 = 0.07*padd(1);


if filter_type == 0
```

```
 filter_img = hpfilter(filter, padd(1), padd(2), D0);
elseif filter_type == 1
 filter_img = lpfilter(filter, padd(1), padd(2), D0);
 end
% Calculate the discrete Fourier transform of the image
FT=fft2(double(img),size(filter_img,1),size(filter_img,2));


% Apply the filter to the Fourier spectrum of the image
FS_I = filter_img.*FT;
% spacial domain.
S_D=real(ifft2(FS_I));
S_D=S_D(1:size(img,1), 1:size(img,2));
subplot(2,2,2), imshow(S_D, [])
% Move the origin of the transform to the center of the frequency rectangle.
centered=fftshift(FT);
Centerd_Shift=fftshift(FS_I);
% use abs to compute the magnitude and use log to brighten display
O1=log(1+abs(centered));
O2=log(1+abs(Centerd_Shift));
subplot(2,2,3), imshow(O1,[])
subplot(2,2,4), imshow(O2,[])
output = filter_img;
```

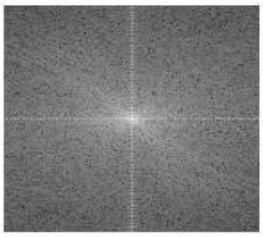
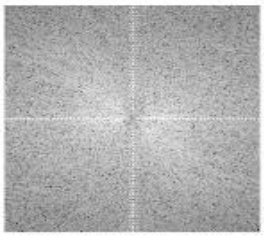
Output:



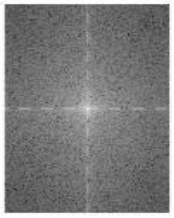
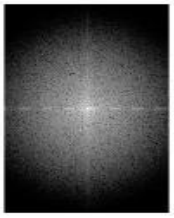
Low_Frequency_image_spectrum



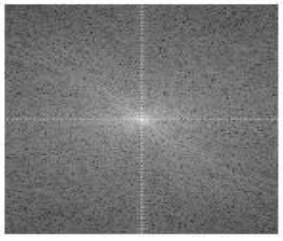
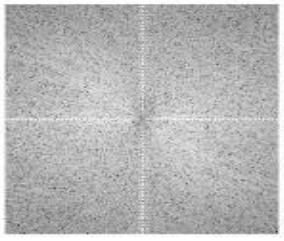
High_Frequecy_image_spectrum



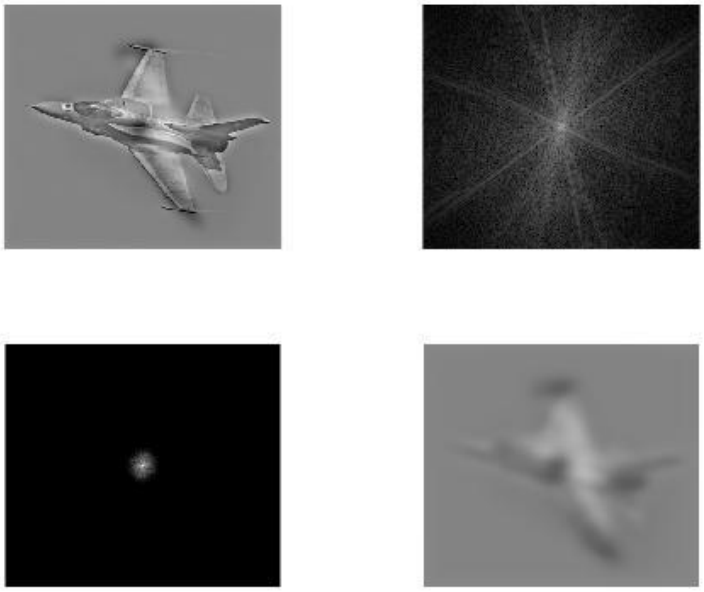



Low_Frequency_image_spectrum_cutoff_freq=11

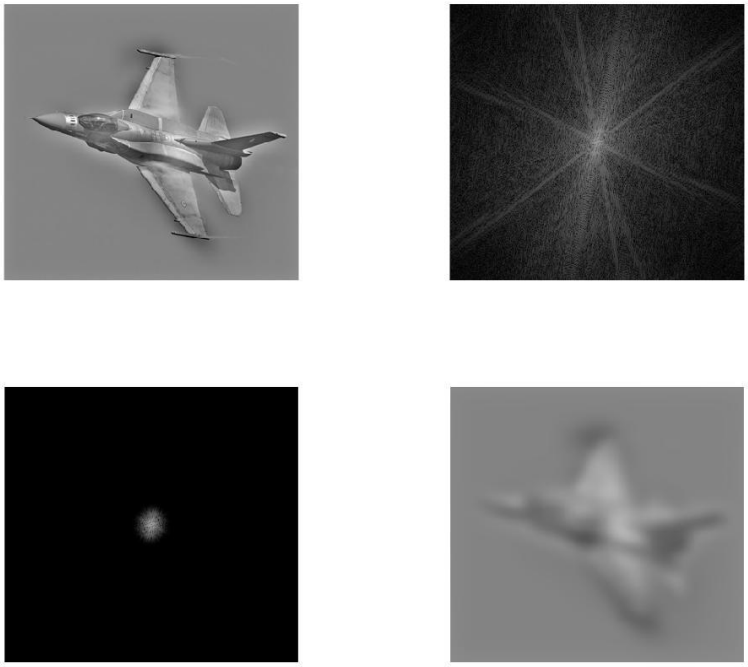


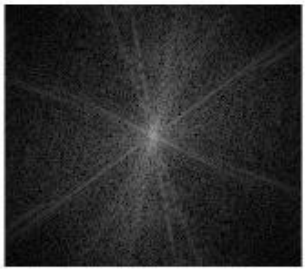
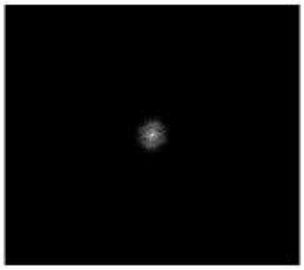
High_Frequency_image_spectrum_cutoff_freq=11



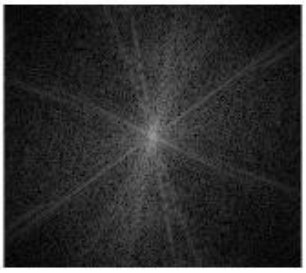
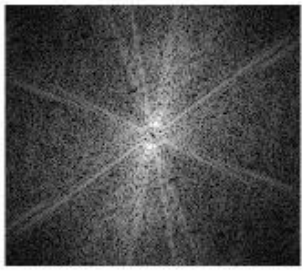
Low pass filter Hybrid Image(With cut-off frequency=7)



high pass filter Hybrid Image (With cut-off frequency=7)


Low pass filter Hybrid Image(With cut-off frequency=11)


High pass filter Hybrid Image (With cut-off frequency=11)

Aim (Part-2):

Implement Sliding window method for template matching.

Template Matching:

Template matching is an algorithm for finding the location of given template image in the original Image. There are several methods that can be implemented for template matching to find exact location of template image in original image. These techniques are:

1) Correlation
2) Zero mean correlation
3) Sum Square Difference
4) Normalized Cross Correlation

Here I have implemented some techniques to find template matching using sliding windows method. Those are zero mean correlation, sum square difference and normalised cross correltion.

Then after finding template in Original Image I calculated Euclidean distance between original center of template and found template. It gives me localisation error in this method.

ALGORITHM:

1) For given query Image first I crop template image from that query Image
2) Then I resize it with facor 2 or 0.5
3) I stored the center of that template image as a point 1
4) Then I am applying either Sum Square difference method or Normalized cross correlation method to localize the template image in original Image.
5) Once I got it I am finding its center and save it as a point2
6) Now I am finding Euclidian distance between these 2 points and that give me some localisation error.

Code:

% Sum Square Difference

```
function output = ssd_f(image, filter)
original = image;

[m n] = size(filter);
if (mod(m,2)==0)
   filter = padarray(filter,[1 0],'post');
end

if (mod(n,2)==0)
   filter = padarray(filter,[0 1],'post');
end
```

```
original=padarray(original,[((size(filter,1))-1)/2,((size(filter,2))-1)/2]);

for k = 1:size(image,3)
for i = 1+(((size(filter,1))-1)/2):size(image,1)+(((size(filter,1))-1)/2)
for j = 1+(((size(filter,2))-1)/2):size(image,2)+(((size(filter,2))-1)/2)
      temp = (filter - original(i-(((size(filter,1))-1)/2):i+(((size(filter,1))-1)/2),j-(((size(filter,2))-
1)/2):j+(((size(filter,2))-1)/2),k)).^2 ;
      out(i-(((size(filter,1))-1)/2),j-(((size(filter,2))-1)/2),k) = sum(temp(:));
end
end
end
output = out;
```

%Zero mean correlation

```
function output = zmc_f(image, filter)

original = image;

[m n] = size(filter);
if (mod(m,2)==0)
   filter = padarray(filter,[1 0],'post');
end

if (mod(n,2)==0)
   filter = padarray(filter,[0 1],'post');
end

mn = mean(filter(:));

filter = filter-mn;

original=padarray(original,[((size(filter,1))-1)/2,((size(filter,2))-1)/2]);

for k = 1:size(image,3)
for i = 1+(((size(filter,1))-1)/2):size(image,1)+(((size(filter,1))-1)/2)
for j = 1+(((size(filter,2))-1)/2):size(image,2)+(((size(filter,2))-1)/2)
   temp = original(i-(((size(filter,1))-1)/2):i+(((size(filter,1))-1)/2),j-(((size(filter,2))-
1)/2):j+(((size(filter,2))-1)/2),k) .* filter;
   out(i-(((size(filter,1))-1)/2),j-(((size(filter,2))-1)/2),k) = sum(temp(:));
end
end
end
output = out;
```

Output Part2A (Template Matching Algorithms):

zero mean correlation

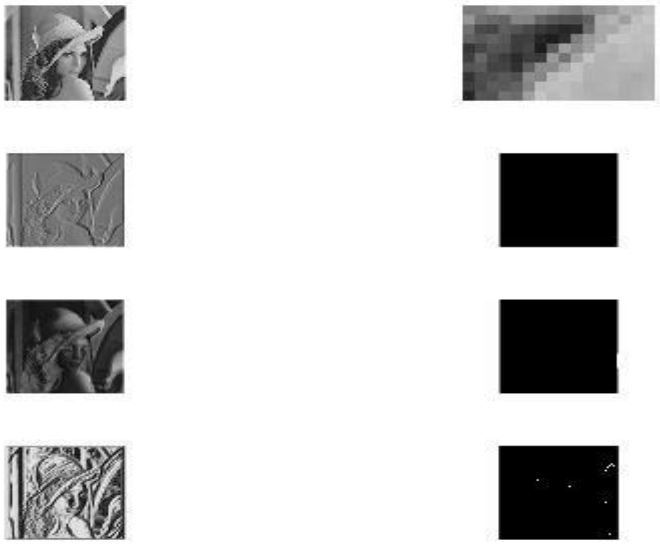
sum square difference

normalized cross correlation

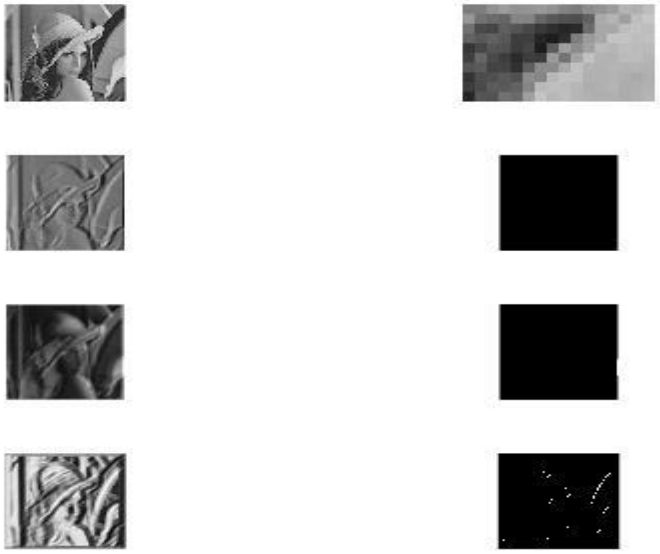
Localisation Error (Euclidean Distance) in zero mean correlation, SSD and NCC:

Error from ZMC - 181.372545
Error from SSD - 120.104121
Error from NCC - 8.944272

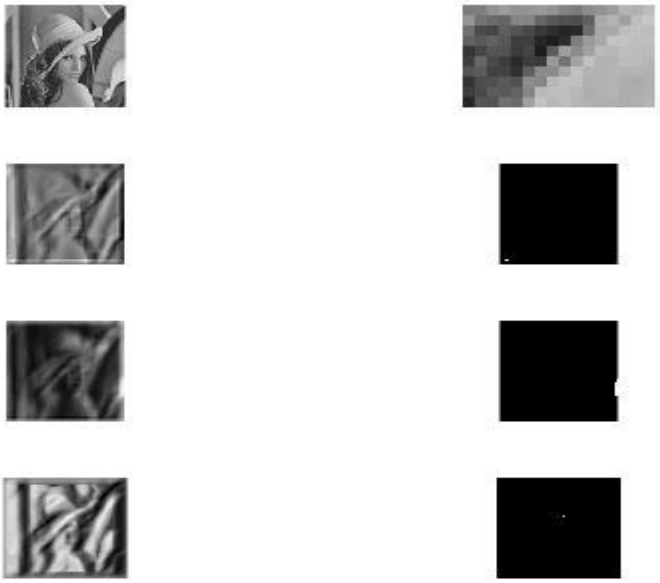
Output Part2B:



Scale Factor:2



Scale Factor:1



Scale Factor: 0.5

Localisation Error (Euclidean distance):

Sacling Factor - 2.000000

Error from ZMC - 383.942704
Error from SSD - 383.942704
Error from NCC - 4.472136

Sacling Factor - 1.000000

Error from ZMC - 188.430358
Error from SSD - 188.430358
Error from NCC - 88.600226

Sacling Factor - 0.500000

Error from ZMC - 128.319913
Error from SSD - 128.319913
Error from NCC - 8.062258

Conclusion:

From this assignment I found that normalize cross correlation give less localisation error than sum square difference and Zero mean correlation. For different scaling I got different localisation error. As I take scale bigger the more normalisation error I got.