SYDE 575

Group #22

Lab 4 Report

Prof David A. Clausi

Abdulhameed Naji - *20642623*

Shilpan Shah - *20603389*

Contents

List of Figures

1 Introduction

This lab investigates various image processing techniques related to image restoration and removing noise within images. This is done using various frequency domain filters and adaptive spatial domain filters. Image restoration techniques related to inverse filtering and Weiner filtering were investigated and compared with various types of noise. Similarly, statistical information such as noise and image variance where incorporated to compute the results of the adaptive Lee Filter.

2 Q2.1 Image Restoration

Figure 1 shows the original cameraman image. First a disk blur function was created using the blurring function provided in the lab. The function was transformed to frequency domain and applied to the Cameraman image to create a smooth image. The result of the smoothing is shown in Figure 2.

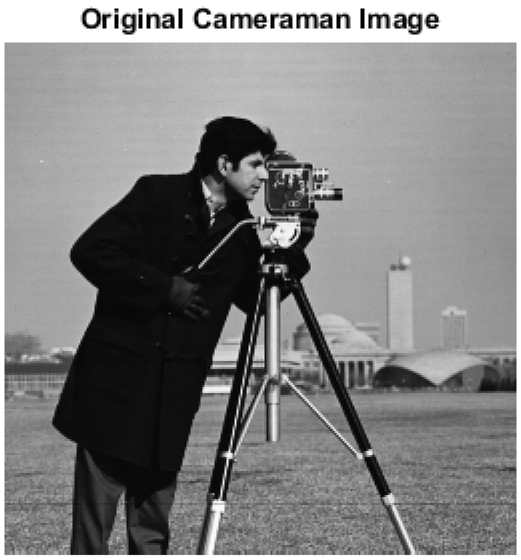

Figure 1: Original Image

In an attempt to restore the image, the inverse filter was used for de-blurring. The inverse filter works well because it performs a naive inverse of the distortion model without accounting for noise. The restored image is virtually identical to the original. The edges have been enhanced with more clear distinction while maintaining image integrity. Figure 3 shows the result of the filter.

The calculated PSNR of the restored image is 255, which is a very large PSNR value compared to typical filtering procedures encountered so far; as a result, the filter is very effective at inverting the blur.

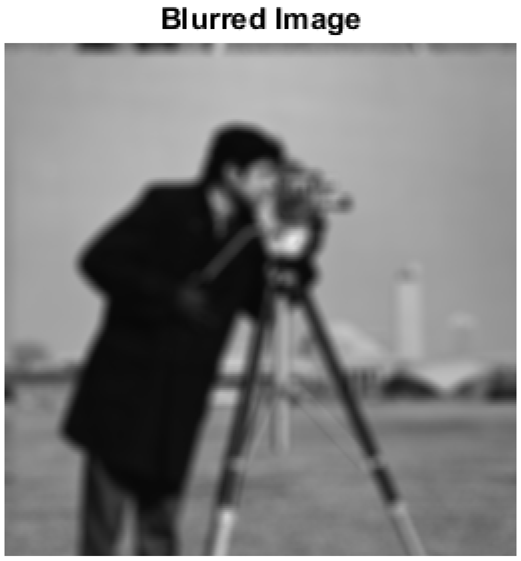

Figure 2: Blurred Cameraman

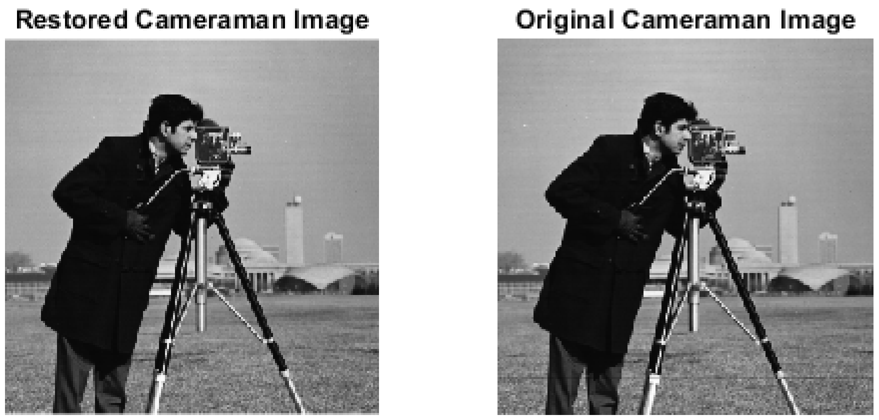

Figure 3: Restored Image and Original Image

2.1 Q2.2 Addition of Gaussian Noise

Next, the Gaussian noise was added to the blurred Cameraman image. The mean of the noise is zero with variance of 0.002. The resulting image is shown in Figure 4 with a PSNR of 16.51. This is a lower PSNR compared to the original, since more noise was added to the image, making it further away from the original. The inverse filter is naively applied to the noisy image. The resulting PSNR is a morbid -41.27, which means that there are a lot of errors and noise within the restored image. Figure 5 shows the restored image done with inverse filtering on the noisy blurred degraded image.

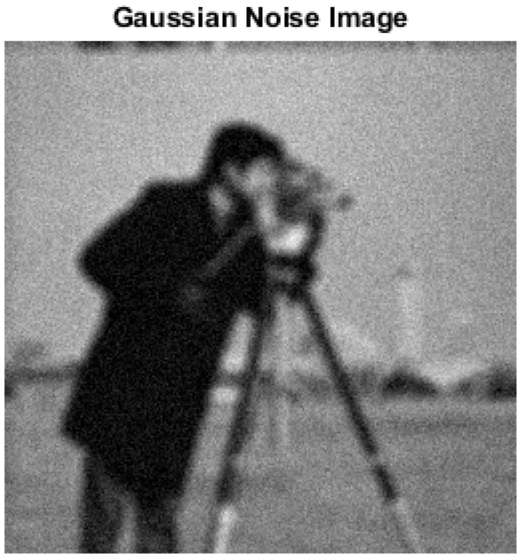

Figure 4: Blurred image contaminated with Gaussian noise.

Noise is associated with high frequency components; the inverse filter is a high pass filter that will amplify the high frequency components, which include noise. As a result, the output is an unintended incoherent image formed by saturated noise samples.

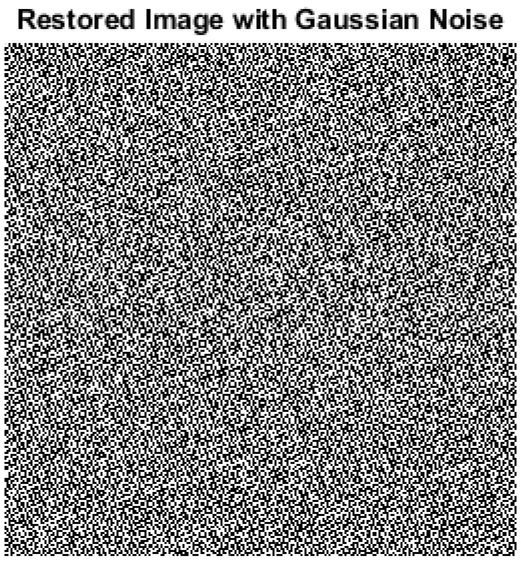

Figure 5: Incoherent restoration due to presence of noise.

2.2 Q2.3 Conclusion of Inverse Filtering

Clearly, inverse filtering does not perform well for image containing noise. The frequency domain representation is given by:

$$\hat{F}(u) = F(u) + \frac{N(u)}{H(u)}$$

The filter attempts to ignore the presence of noise, which ironically exacerbates its effect. For small distortion H, the noise N dominates, and the entire signal gets saturated by noise.

2.3 Q2.4 Wiener Filtering

Next,the Wiener Filter was used to restore the Gaussian contaminated blurred Cameraman image. The Wiener process isolates and removes noise first and then performs the naive inverse filtering. The Wiener filter requires calling the built in MATLAB *deconvwnr* function.The signal to noise ratio of the additive noise was calculated by dividing the variance of the noise with the variance of the noisy image. The NSR can be found as:

$$NSR = \frac{\sigma_{noise}}{\sigma_{signal}}$$

Since the variance of the noise is 0.002 and the variance of the noisy image is 0.0508, the NSR was calculated as 0.0394. Apply the Wiener filter on the image yields the results shown in Figure 6

2.4 Q2.5 Wiener Filter Conclusion

With a PSNR of 15.866, the Wiener filter is significantly superior to the inverse filtering method. By initially rectifying the noise, the filtered image is more coherent while also smoothing the present Gaussian noise. Although superior to the inverse filter, the method did not fully arrive at a sufficiently desirable output. The image experiences less turbulent noise (higher variance Gaussian). The original noise contaminated image had a PSNR of 16.49. As a result, we conclude that the Wiener filter's success is reliant on a suitable NSR calculation. In fact, by performing trial and error, a contrived NSR of 0.09 produces the most optimal image with PSNR 16.17.

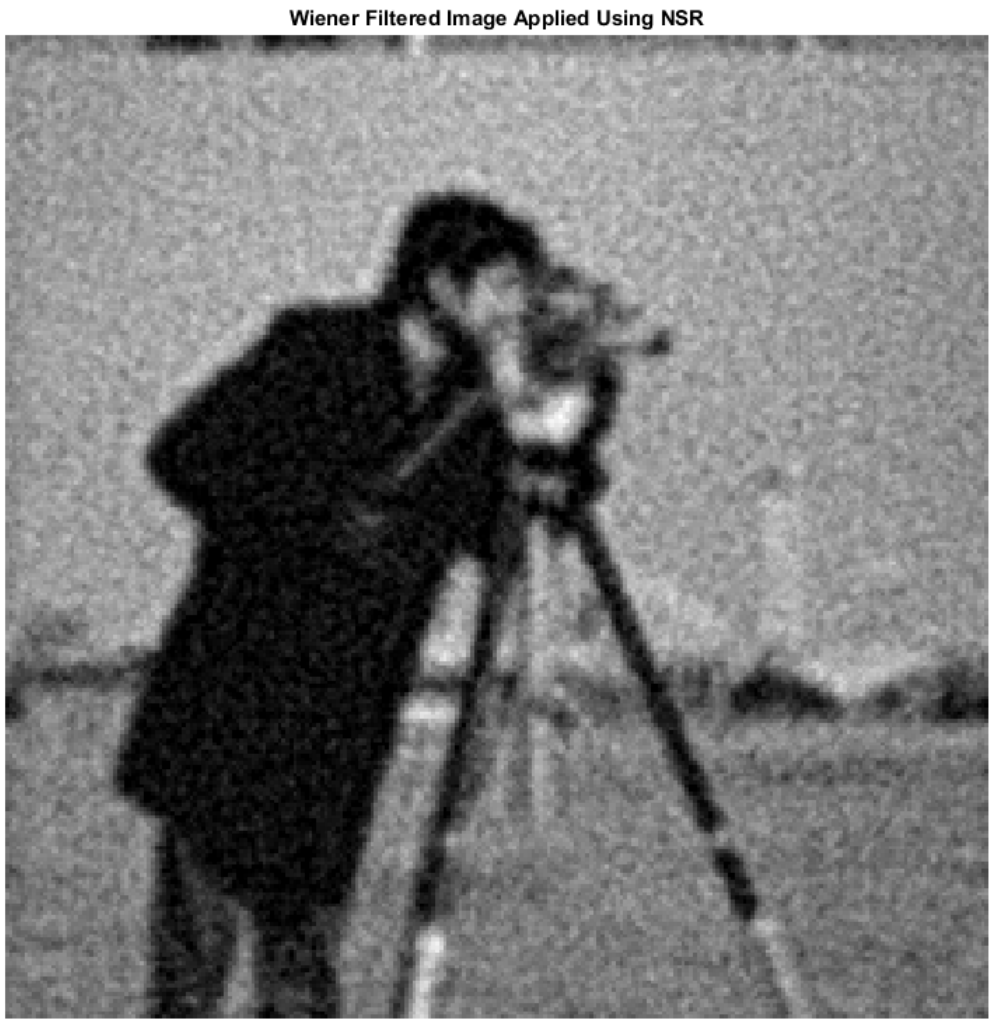

Figure 6: Wiener filtered image

3 Q3.1 Adaptive Filtering

The Lee Filter is implemented to perform adaptive filtering that would preserve edge details while blurring noisy regions of the image. The filter works by weighing the blur operation based on the variance of a local patch in the image. For a given pixel, if the local patch has a variance that is significantly higher than the noise variance, the pixel remains intact. Conversely, when noise variance dominates, the blurred value is selected.

To perform this operation, the mean (or blurred) value of each pixel is calculated based on the local window size. Simultaneously, the variance of the local window is also pre-calculated for the upcoming stage. A weight value, K, is the adaptive variable that dictates whether the pixel should use the pre-calculated mean versus its original intensity.

$$K = \frac{\sigma_g^2 - \sigma_n^2}{\sigma_g^2}$$

As the image patch variance dominates, the value of K approaches 1 – effectively giving more weight to the original intensity. As the influence of noise variance dominates, the value of K becomes smaller – effectively giving more weight to the mean intensity.

$$\hat{f} = Kg + (1 - K)\bar{g}$$

Following the implementation of the Lee Filter in the Appendix, the degraded image undergoes filtering with a window size of 5 and a sample noise variance approximately equal to 0.011. The result is shown in Figure 7. The resulting PSNR is **18.988**.

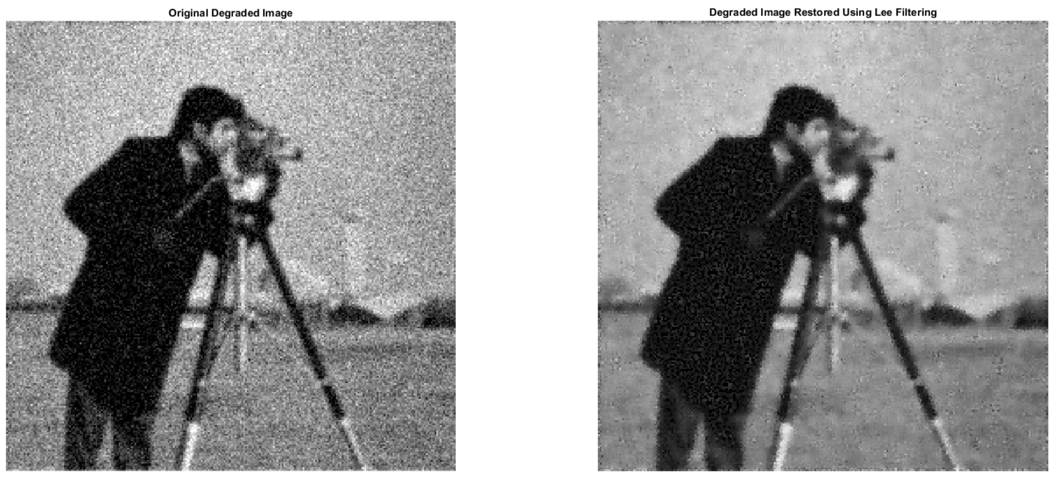

Figure 7: Lee Filtered image with size 5 & sampled noise variance 0.011

3.1 Q3.2 Comparison With Gaussian

To form a better conclusion about the workings of the Lee Filter, it is compared to a Gaussian Filter with standard deviation of 30. The result of Gaussian smoothing is shown in 8.

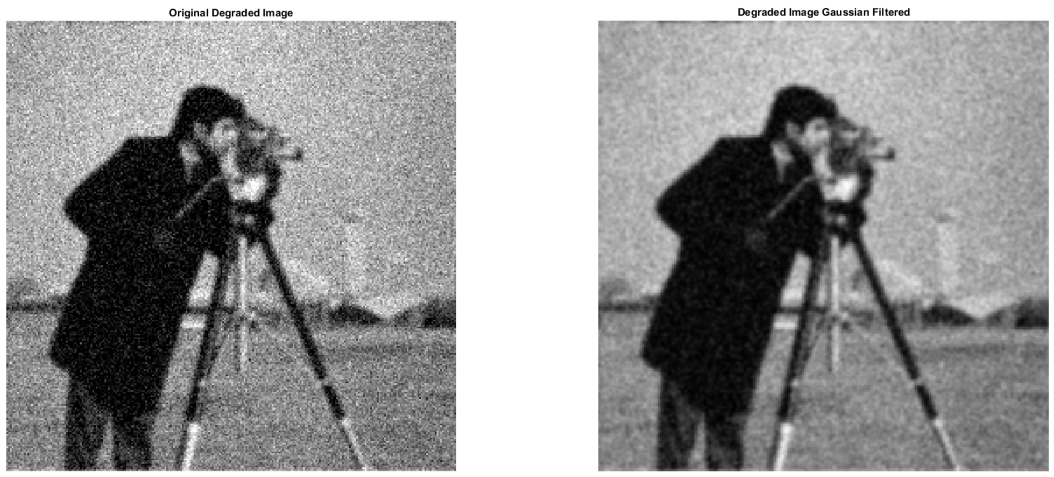

Figure 8: Gaussian Filtered image with standard deviation 30

The immediate difference of the filters is notable in their ability to remove background speckle. The Gaussian filtered image exhibits remnant noise whereas the Lee filter produces a clearer environment background. In terms of edge preservation, the Gaussian indiscriminately blurs the edge of the cameraman and the equipment.

The Lee filtered image shows minimal notable changes around the edges from the original image. This aligns with our interpretation as edges are expected to have minimal weight to be influenced by the mean intensity. Conversely, when looking past the edge (i.e the cameraman's clothing), we see the filter influence returning to rectify noise present in a low variance patch of the image.

Another interesting confirmation regarding the working of the filter are the borders of the images as shown in Figure 9. In the Lee Filter case, the borders are very high frequency edges (most likely due to 0 padding or periodicity); consequently, they appear noisy and intact from any blurring procedure.

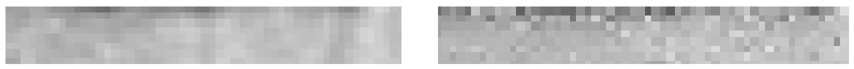

Figure 9: Border of Gaussian Filtered image (left) and Lee Filtered image (right)

3.2 Q3.3 Effect of Noise Variance

Based on the expression for the weighing factor, K, the dominance of noise is expected to increase the blurring effect. Conversely, when the image variance dominates, the image is left intact. This is proven by Figure 10

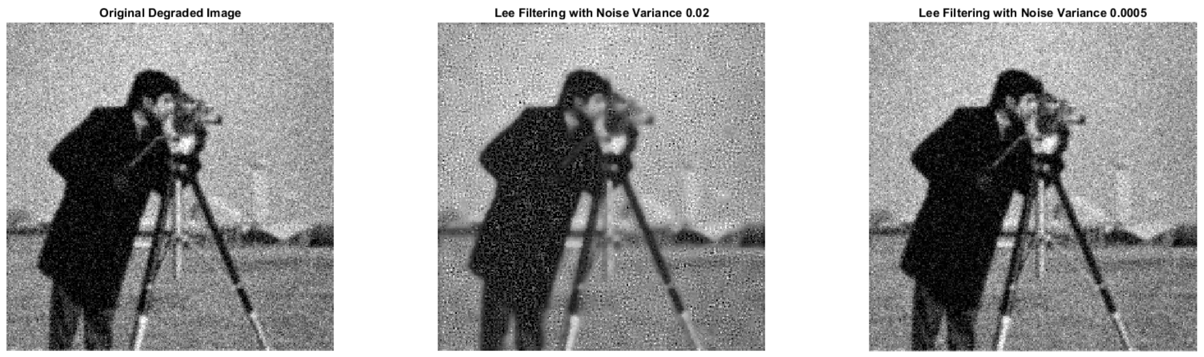
Original Degraded Image
Lee Filtering with Noise Variance 0.02
Lee Filtering with Noise Variance 0.0005

Figure 10: Lee Filter under varying noise variance

When the noise variance is as low as 0.0005, the variance of the local patch dominates. This leads to a K value of approximately 1. As a result, the image experience little to no filtering as the filter simply applies the original intensities to the pixels.

When the noise variance is as high as 0.02, the edges of the cameraman get heavily blurred since the weight is biased by the dominance of noise variance; hence, the mean value is applied to the pixels.

3.3 Q3.4 Effect of Local Size

The Lee filter works by determining statistical properties about features of an image – namely the variance. The size of the filter dictates how much samples are taken to calculate the variance. More samples yield more accurate statistics about an image feature; however, the trade-off is the risk of introducing outlier pixels that share no spatial context with the patch of interest. By shrinking the size of the filter, the outlier risk is negated, but there is not enough samples to accurately describe the statistics of an image feature.

Figure 11 shows the effects of varying the size of the Lee Filter. The base case was filtered with a window size of 5, whereas the figure depicts the results of choosing a smaller size of 2 and a larger size of 10.

With a small window size, the small number of sample capture a relatively erratic change in intensities. As a result, the variance of the image patch is perceived as large. This dominates over the noise variance in the filter response – consequently, the filter assumes an edge or texture detail has been detected, which in turn excludes the feature from smoothing. The result is a very noisy image.

For a large window size, areas without sufficient contrast are blurred. With a larger window size, we introduce outliers that bias the variance of the patch to a lower value; this occurs because such outliers typically belong to uniform areas such as the background. As a result, the image is generally blurred. However, due to the biasing nature of size, the opposite

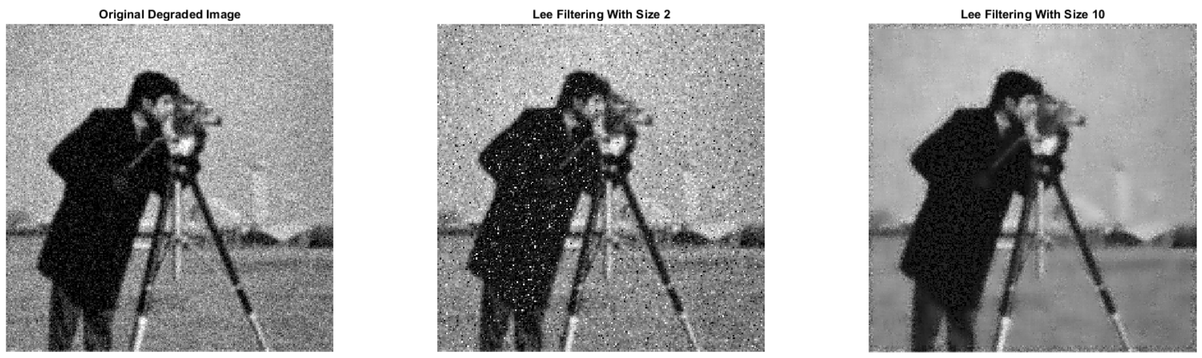

Figure 11: Lee Filter under varying sizes

phenomena occurs. The edges of the image appear "thicker". When calculating samples for an area that is just outside an edge feature, the large window size introduces an outlier edge which biases the variance to a high value. The effect is a thicker un-smoothed boundary due to pixels excluded from the blur.

4 Conclusion

Image distortion generically describes an undesirable transformation that needs to be inverted. Often, the distortion is unknown - making the inverse difficult to attain. In the case of the naive inverse filtering performed on the cameraman image, the exact distortion was already known, and was used to calculate the inverse to arrive at a pristine result. However, noise adds a layer of complexity for the inverse process. In order to avoid saturation of noisy components, the Weiner deconvolution was pursued to undo the effect of the noise before undo-ing the distortion effect. The result shows the imperfections resulting from a restoration process – moreover, it highlights the difference between restoration and enhancement. In the case of the Weiner filter, the image had an objectively poor PSNR, yet it is perceived better than the noisy image. We say that the image is restored, but not necessarily enhanced.

The Lee Filter experiment is a precursor to the notion of adaptive filtering. Statistical data is extracted from the filter's window in order to control the impact of the filter. By relying on the variance of the samples, the Lee Filter arrived at a linearly interpolated result for each pixel – the impact of the filter is suppressed in patches where variance dominates the noise variance. This in turn excludes the edges from the smoothing process. The Lee Filter is effective at preserving edges; however, its performance is dictated by size and noise statistics. With larger window sizes, its effect is stronger on both edges and blurred backgrounds. Conversely, an incorrect noise characterization will bias the result to either over-smooth details or ignore the smoothing operation on some areas.

5 Appendix

5.1 lab4.m

```
%% Section 1
degraded = imread('degraded.tif');
cameraman = imread('cameraman.tif');

figure;
imshow(degraded);
figure;
imshow(cameraman);

%% Section 2 - Image Restoration in the Frequency Domain

h = fspecial('disk',4);
f = im2double(imread('cameraman.tif'));
h_freq = fft2(h, size(f,1), size(f,2)); % pad h
f_blur = real(ifft2(h_freq.*fft2(f)));

figure;
imshow(f_blur);
title('Blurred Image');

% PSNR of blurred image
f_blur_psnr = psnr(f,f_blur);
f_blur_psnr

% Inverse Filter
inverseFilterImage = real(ifft2((fft2(f_blur)./h_freq)));
figure;
subplot(1,2,1);
imshow(inverseFilterImage);
title('Restored Image');

subplot(1,2,2);
imshow(cameraman);
title('Original Cameraman Image');

% PSNR of Inverse Filtered image
f_inverse_psnr = psnr(f,inverseFilterImage);
f_inverse_psnr

% Image with Gaussian Noise
% Add Gaussian Noise
```

```
noiseyImage = imnoise(f_blur,'gaussian',0,0.002);
inverseFilterNoisyImage = real(ifft2((fft2(noiseyImage)./h_freq)));

figure;
imshow(inverseFilterNoisyImage);
title('Restored Image with Gaussian Noise');

figure;
imshow(noiseyImage);
title('Gaussian Noise Image');

% PSNR of Noisy image and Restored image
f_noisy_image_psnr = 10*log10(1/mean2((f-noiseyImage).^2));
f_noisy_image_restored_psnr = psnr(f,inverseFilterNoisyImage);
f_noisy_image_psnr
f_noisy_image_restored_psnr
%%
% Wiener Function
% Reference to Winer Function from
    https://www.mathworks.com/help/images/ref/deconvwnr.html
% nsr is the noise-to-signal power ratio of the additive noise...below is
% the example code..
% estimated_nsr = noise_var / var(I(:));
% wnr3 = deconvwnr(blurred_noisy, PSF, estimated_nsr);
% figure, imshow(wnr3)
% title('Restoration of Blurred, Noisy Image Using Estimated NSR');

var_noise = 0.002;
var_f = var(noiseyImage(:));
nsr = var_noise/var_f;
%nsr=0.09;

wienerFilter = deconvwnr(noiseyImage, h, nsr);

figure;
imshow(wienerFilter);
title('Wiener Filtered Image Aplied Using NSR');

% PSNR of Wiener Filtered Image
Wiener_psnr = psnr(f,wienerFilter);
Wiener_psnr

%% Adaptive Filtering
degraded = imread('degraded.tif');
degraded = im2double(degraded);
```

```
imshow(degraded)
rect = getrect();
cropped = imcrop(degraded, rect);
noise_var = var(cropped(:)); % ~0.011

degraded_leeFiltered = LeeFilter(degraded, noise_var, 5);
leeFilter_psnr = psnr(degraded_leeFiltered, degraded);

figure
    subplot(1,2,1)
        imshow(degraded)
        title('Original Degraded Image')

    subplot(1,2,2)
        imshow(degraded_leeFiltered)
        title('Degraded Image Restored Using Lee Filtering')
%% Comparison with Gaussian
filt_gaussian_30 = fspecial('gaussian', 256, 50);
filt_gaussian_30 = filt_gaussian_30./(max(max(filt_gaussian_30)));

fft_degraded = fftshift(fft2(degraded));
fft_gauss_filtered = ifftshift(fft_degraded .* filt_gaussian_30);
degraded_gaussFilt = ifft2(fft_gauss_filtered);

figure
    subplot(1,2,1)
        imshow(degraded)
        title('Original Degraded Image')

    subplot(1,2,2)
        imshow(degraded_gaussFilt)
        title('Degraded Image Gaussian Filtered')

%% Varying Noise Variance
degraded_high_variance = LeeFilter(degraded, .02, 5);
degraded_low_variance = LeeFilter(degraded, 0.0005, 5);
figure
    subplot(1,3,1)
        imshow(degraded)
        title('Original Degraded Image')

    subplot(1,3,2)
        imshow(degraded_high_variance)
        title('Lee Filtering with Noise Variance 0.02')
    subplot(1,3,3)
        imshow(degraded_low_variance)
        title('Lee Filtering with Noise Variance 0.0005')

%% Varying local calculations
degraded_size2 = LeeFilter(degraded, noise_var, 2);
degraded_size10 = LeeFilter(degraded, noise_var, 10);

figure
    subplot(1,3,1)
        imshow(degraded)
        title('Original Degraded Image')

    subplot(1,3,2)
        imshow(degraded_size2)
        title('Lee Filtering With Size 2')

    subplot(1,3,3)
        imshow(degraded_size10)
        title('Lee Filtering With Size 10')
```

5.2 LeeFilter.m

```
function f = LeeFilter(g, noise_var, sz)
    f = zeros(size(g));

    [w, h] = size(f);
    local_mean = colfilt(g, [sz,sz], 'sliding', @mean);
    local_var = colfilt(g, [sz,sz], 'sliding', @var);

    for i = 1 : w
        for j = 1 : h
            K = (local_var(i,j) - noise_var) / (local_var(i,j));
            g_curr = g(i,j);
            g_bar = local_mean(i,j);

            f(i,j) = (K*g_curr + (1-K)*g_bar);
        end
    end
end
```