# Image and Video Processing

## Programming Assignment 10

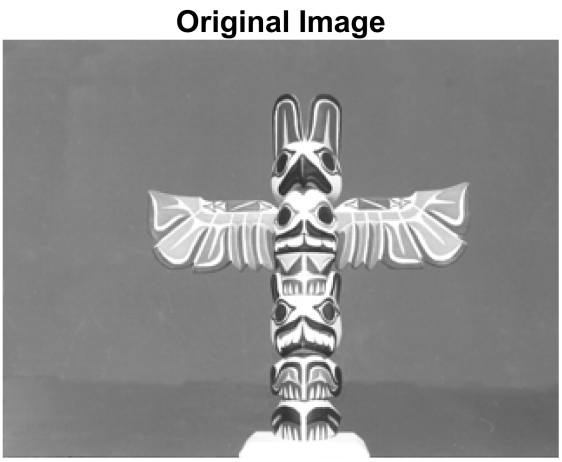
Image Restoration


Submitted by
**Madhu Krishnan A P**
(Student ID: 24100488)
M.Tech VLSI and Embedded Systems
Cochin University of Science and Technology
Cochin - 22

# 1 Read an input image, Add AWGN noise of zero mean and variance 1000. Implement an arithmetic mean and a geometric mean filter of 7 X 7 size and compare the result.

**Read and display input image**

```
img = imread('totem.tif');
img = im2double(img); % Convert to double for processing

resized_img = imresize(img, 0.5);

figure;
imshow(resized_img);
title('Resized Original Image');
```




**Add AWGN noise with zero mean and variance 1000**

```
variance = 1000 / 255^2; % Normalize for 8-bit image scale
noisy_img = imnoise(img, 'gaussian', 0, variance);
```

**Apply Arithmetic Mean Filter (7x7)**

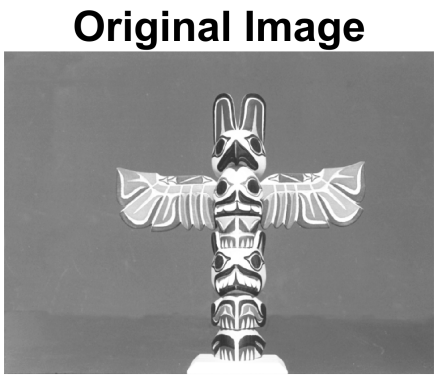
```
h = fspecial('average', [7 7]);
arithmetic_filtered = imfilter(noisy_img, h, 'replicate');
```

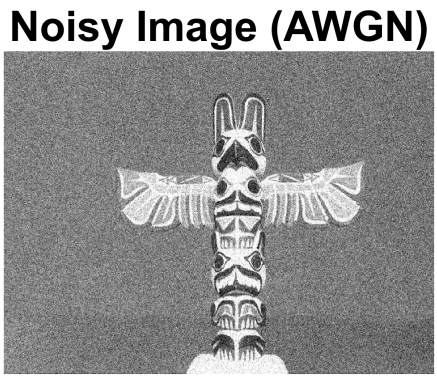
**Apply Geometric Mean Filter (7x7)**

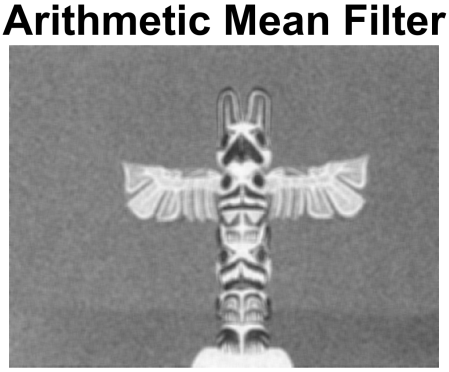
```
[M, N] = size(noisy_img);
geo_filtered = noisy_img;

for i = 4:M-3
    for j = 4:N-3
        patch = noisy_img(i-3:i+3, j-3:j+3);
        geo_filtered(i, j) = exp(mean(log(patch(:) + eps))); % Prevent log(0)
    end
end
```

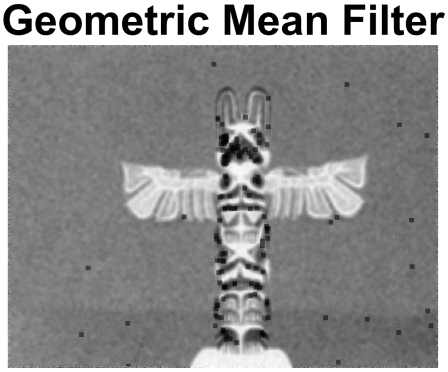
**Display results**

```
figure;
subplot(2,2,1); imshow(img); title('Original Image');
subplot(2,2,2); imshow(noisy_img); title('Noisy Image (AWGN)');
subplot(2,2,3); imshow(arithmetic_filtered); title('Arithmetic Mean Filter');
subplot(2,2,4); imshow(geo_filtered); title('Geometric Mean Filter');
```




## 2 Read an input image, Add salt and pepper noise with probabilities P(a) = P(b) = 0.1. Implement a median filter of size 3 X 3. Pass the output image 3 times to the same filter and compare the results.

**Read and display input image**

```
img = imread('f18.tif');
img = im2double(img); % Convert to double for processing

resized_img = imresize(img, 0.5);
```

```
figure;
imshow(resized_img);
title('Resized Original Image');
```


**Add salt and pepper noise with probabilities $\mathbf{P}_a = \mathbf{P}_b = \mathbf{0.1}$**

```
noisy_img = imnoise(img, 'salt & pepper', 0.1);
```

**Apply 3x3 Median Filter - First Pass**

```
filtered_1 = medfilt2(noisy_img, [3 3]);
```

**Apply 3x3 Median Filter - Second Pass**

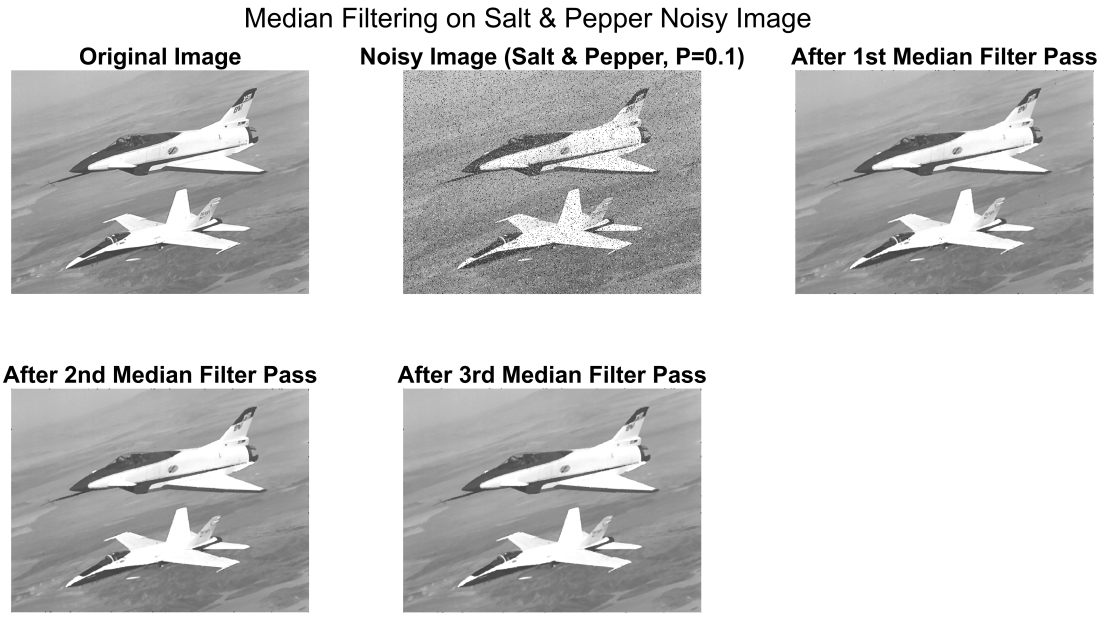
```
filtered_2 = medfilt2(filtered_1, [3 3]);
```

**Apply 3x3 Median Filter - Third Pass**

```
filtered_3 = medfilt2(filtered_2, [3 3]);
```

**Display results**

```
figure;
set(gcf, 'Position', [100, 100, 1200, 500]); % Set figure size
subplot(2,3,1), imshow(img), title('Original Image');
subplot(2,3,2), imshow(noisy_img), title('Noisy Image (Salt & Pepper, P=0.1)');
subplot(2,3,3), imshow(filtered_1), title('After 1st Median Filter Pass');
subplot(2,3,4), imshow(filtered_2), title('After 2nd Median Filter Pass');
subplot(2,3,5), imshow(filtered_3), title('After 3rd Median Filter Pass');
```




## 3 Read an input image and model its degradation as atmospheric turbulence with k values of 0.0025, 0.001, and 0.00025, and uniform linear motion with a = b = 0.1 and T = 1. Obtain the degraded images in both cases.

**Read and display input image**

```
img = imread('f18.tif');
img = im2double(img); % Convert to double for processing
[M, N] = size(img); % Get image dimensions

figure;
imshow(img);
title('Original Image');
```

### Create frequency domain coordinates

```
[u, v] = meshgrid(-N/2:N/2-1, -M/2:M/2-1);
D = u.^2 + v.^2; % Distance squared for turbulence model
```

### Perform FFT of the image

```
F = fftshift(fft2(img));
```

### Apply Atmospheric Turbulence Degradation

```
k_values = [0.0025, 0.001, 0.00025];
degraded_turbulence = cell(1,3);

for i = 1:3
    k = k_values(i);
    H_turbulence = exp(-k * D); % Turbulence degradation function
    G_turbulence = H_turbulence .* F; % Apply degradation in frequency domain
    degraded_turbulence{i} = abs(ifft2(ifftshift(G_turbulence))); % Inverse FFT
end
```

### Apply Uniform Linear Motion Blur Degradation

```
a = 0.1; b = 0.1; T = 1;
H_motion = (sinc(a * u + b * v) .* exp(-1j * pi * (a * u + b * v) * T));

G_motion = H_motion .* F;
degraded_motion = abs(ifft2(ifftshift(G_motion))); % Inverse FFT
```

### Display Results

```
figure;
set(gcf, 'Position', [100, 100, 1400, 600]);

subplot(2,3,1), imshow(img), title('Original Image');
subplot(2,3,2), imshow(degraded_turbulence{1}), title('Turbulence (k = 0.0025)');
subplot(2,3,3), imshow(degraded_turbulence{2}), title('Turbulence (k = 0.001)');
subplot(2,3,4), imshow(degraded_turbulence{3}), title('Turbulence (k = 0.00025)');
subplot(2,3,5), imshow(degraded_motion), title('Motion Blur (a = b = 0.1, T = 1)');
```