# Table of Contents



```
%NAME: Vatsalya Chaubey
%INST: IIT, Bhubaneswar
%DATE: 29.10.2020
%CATEGORY: Btech
%BRANCH: Electronics and Communication
%Roll Number: 17EC01044
```

# Image and Video Processing Assignment - 5

```
clc;
clear all;
close all;
```

# Question 1: Motion Blur through Degradation function

```
% I will first create the functions which are needed for
 implementation in
% other parts of the code. Finally, the entire code will be written
% together.
```

# Motion Blur

```
% The function returns a motion blurred image

function [out] = motion_blur(img, T, a, b)
```

```
% img: input image
% T: the duration of exposure
% a: rate of motion in x-direction
% b: rate of motion in y-direction
%
% out: the motion blurred image

[m, n] = size(img);
u = linspace(0,m-1,m)' + 1e-16;
v = linspace(0,n-1,n) + 1e-16;

u = repelem(u, 1, n);
v = repelem(v, m, 1);

cord = a.*u + b.*v';

% The degradation model derived
H_uv = (T.*sin(pi*cord).*exp(-1i*pi*cord))./(pi*cord);

F_uv = fft2(img);

out_fft = F_uv .* H_uv;

out = real(ifft2(out_fft));

out = mat2gray(out);

end
```

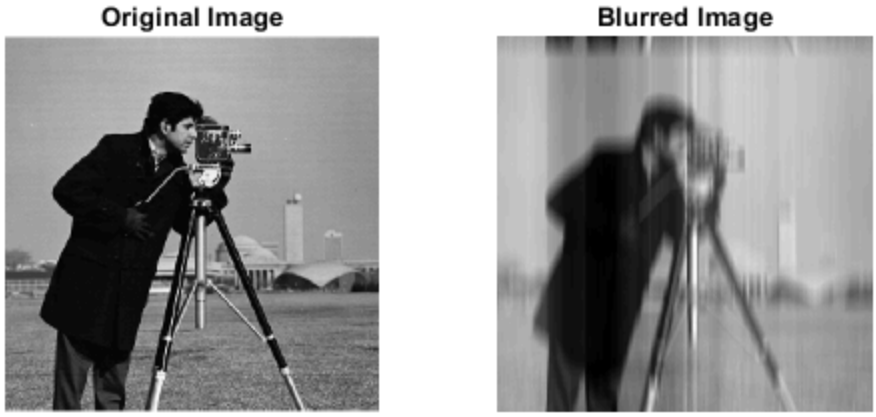
# Example: Motion Blur

```
% Read the input image as a double
orig_img = imread('cameraman.tif');
img = double(orig_img);

motion = motion_blur(img, 1, 0.05, 0.01);

figure('Name', 'Motion Blur');
subplot(121)
imshow(orig_img);
title('Original Image');

subplot(122)
imshow(motion);
title('Blurred Image');
```




# Inverse Filtering

We can remove the motion blurring of an image if we can come up with a mathematical model of the blurring and then we can design a inverse filter for it.

The mathematical model of the motion blur

$$g(x,y) = \int_0^T [x - x_0(t), y - y_0(t)]dt$$

It is assumed that the image has undergone a movement $x_0(t)$ and $y_0(t)$ in the $x$ and $y$ direction respectively. $T$ is the exposure time of the camera sensor.

In the frequency domain, it can be expressed as:

$$G(u,v) = F(u,v) * H(u,v)$$

$$H(u,v) = \frac{T}{\pi(ua+vb)} * sin(\pi(ua+vb)) * e^{-i\pi(ua+vb)}$$

Where: $x_0(t) = at/T$ and $y_0(t) = bt/T$

$F(u,v)$ is the FFT of the original image $H(u,v)$ is the degradation function $G(u,v)$ is the FFT of the degraded image

We can theortically get back the original image by using inverse filter

$$F(u,v) = G(u,v)/H(u,v)$$

# Code for inverse filter

```
% The function returns a inverse filter image

function [out] = inverse_filter(img, T, a, b)
% img: input image
% T: the duration of exposure
% a: rate of motion in x-direction
% b: rate of motion in y-direction
%
% out: the restored image

[m, n] = size(img);
u = linspace(0,m-1,m)' + 1e-16;
v = linspace(0,n-1,n) + 1e-16;

u = repelem(u, 1, n);
v = repelem(v, m, 1);

cord = a.*u + b.*v';

H_uv = (T.*sin(pi*cord).*exp(-1i*pi*cord))./(pi*cord);
G_uv = fft2(img);

out_fft = G_uv ./ H_uv;

out = real(ifft2(out_fft));

out = mat2gray(out);

end
```

# Example: Inverse Filtering

```
% Read the input image as a double
orig_img = imread('cameraman.tif');
img = double(orig_img);

motion = motion_blur(img, 1, 0.05, 0.01);
filtered = inverse_filter(motion, 1, 0.05, 0.01);

figure('Name', 'Inverse filtering');
subplot(131)
imshow(orig_img);
title('Original Image');
```

```
subplot(132)
imshow(motion);
title('Blurred Image');

subplot(133)
imshow(filtered);
title('Filtered Image');
```

# Radial Inverse Filtering

We can see that the inverse filtering did not give us back the original image. This is due to the high frequency components which get introduced during the filteration process. To extract the original image we can pass the output of the inverse filter through a low pass butterworth filter to remove the high frequency components.

The frequency response of a low pass Butterworth filter with order 10:

$$H(u,v)^2 = \frac{1}{1 + (\frac{D(u,v)}{D_0})^{20}}$$

where $D(u,v)$ is the distance of a point from origin in frequency domain

and $D_0$ is the threshold frequency

# Code for Radial Inverse Filtering

```
% The function returns a radial inverse filtered image

function [out] = radial_inv_filter(img, T, a, b, D_th)
% img: input image
% T: the duration of exposure
% a: rate of motion in x-direction
% b: rate of motion in y-direction
% D_th: The threshold distance for the Butterworth filter
%
% out: the restored image

[m, n] = size(img);
u = linspace(0,m-1,m)' + 1e-16;
v = linspace(0,n-1,n) + 1e-16;

u = repelem(u, 1, n);
v = repelem(v, m, 1);

cord = a.*u + b.*v';

% The degradation function
H_uv = (T.*sin(pi*cord).*exp(-1i*pi*cord))./(pi*cord);
```

```
% The butterworth filter
D = sqrt(u.^2 + v.^2);
D_uv = 1./(1 + (D./D_th).^20);

G_uv = fft2(img);

out_fft = G_uv ./ H_uv;
out_fft = out_fft .* D_uv;

out = real(ifft2(out_fft));

out = mat2gray(out);

end
```

# Example: Radial Inverse Filtering

```
% Read the input image as a double
orig_img = imread('cameraman.tif');
img = double(orig_img);

motion = motion_blur(img, 1, 0.05, 0.01);
filtered = radial_inv_filter(motion, 1, 0.05, 0.01, 9);

figure('Name', 'Radial Inverse filtering');
subplot(131)
imshow(orig_img);
title('Original Image');

subplot(132)
imshow(motion);
title('Blurred Image');

subplot(133)
imshow(filtered);
title('Radial Inverse Filtered Image');
```

# Weiner Filtering

We can also use Weiner filtering to remove the motion blur where we try to reduce the distance between the two images. The frequency response of Weiner filter is:

$$F(u,v) = \frac{|H(u,v)|^2}{|H(u,v)|^2 + K} * \frac{G(u,v)}{H(u,v)}$$

# Code for weiner filter

```
% The function returns a weiner filtered image
```

```
function [out] = weiner_filter(img, T, a, b, K)
% img: input image
% T: the duration of exposure
% a: rate of motion in x-direction
% b: rate of motion in y-direction
% K: constant for weiner filtering
%
% out: the restored image

[m, n] = size(img);
u = linspace(0,m-1,m)' + 1e-16;
v = linspace(0,n-1,n) + 1e-16;

u = repelem(u, 1, n);
v = repelem(v, m, 1);

cord = a.*u + b.*v';

H_uv = (T.*sin(pi*cord).*exp(-1i*pi*cord))./(pi*cord);
G_uv = fft2(img);

out_fft = G_uv .* ((abs(H_uv).^2)./(H_uv .* (abs(H_uv).^2 + K)));

out = real(ifft2(out_fft));

out = mat2gray(out);

end
```

# Example: Weiner Filtering

```
% Read the input image as a double
orig_img = imread('cameraman.tif');
img = double(orig_img);

motion = motion_blur(img, 1, 0.05, 0.01);
filtered = weiner_filter(motion, 1, 0.05, 0.01, 0.04);

figure('Name', 'Weiner filtering');
subplot(131)
imshow(orig_img);
title('Original Image');

subplot(132)
imshow(motion);
title('Blurred Image');

subplot(133)
imshow(filtered);
title('Weiner Filtered Image');
```

# Example: Weiner Filtering on Facial Images

```
% Read the input image as a double
orig_img = imread('face-image.jpg');
img = double(rgb2gray(orig_img));

filtered = weiner_filter(img, 0.004, 0.001, 0.01, 0.01);

figure('Name', 'Weiner filtering');
subplot(121)
imshow(orig_img);
title('Original Image');

subplot(122)
imshow(filtered);
title('Weiner Filtered Image');
```

# Conclusion

Through this experiment we investigated motion blurring and how can we come up with basic degradation models which describe the blurring. The first part of the experiment we introduced motion blurring in an image through our derived degradation function. Then we used inverse filtering to remove the effects of the blurring. But it is clear that during inverse filtering all values get saturated and hence the required image is not visible. To reduce the saturation we use radial inverse filtering in which the output spectrum is again passed through a Butterworth filter to remove very high values.

One of the better methods which gives the best results of all the filtering processes is the weiner filter. I also demonstrate its applicability on a motion blurred facial image.

*Published with MATLAB® R2020b*