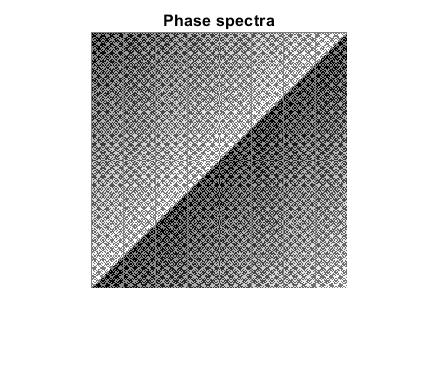
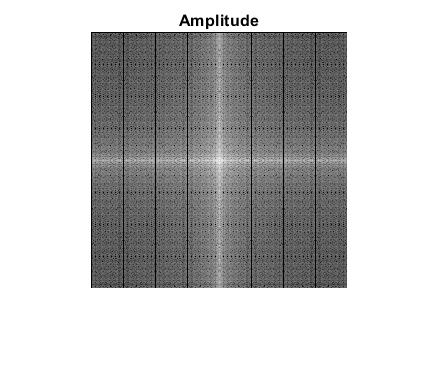
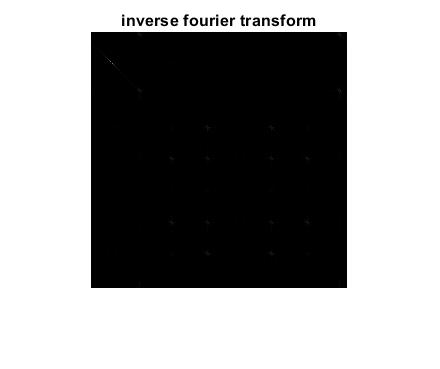
**Computer Assignment 2**

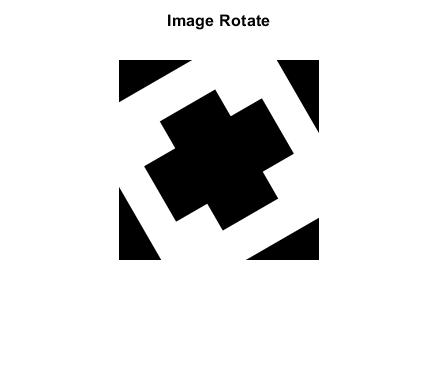
1. Properties of the Fourier Transform

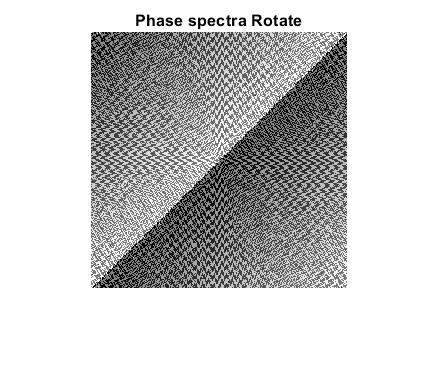
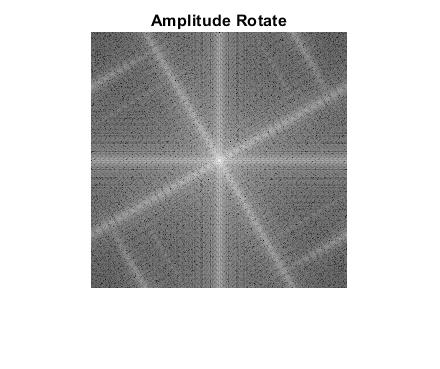
1.1 ทำการ Pad รูปจากเดิมที่ขนาด 200x200 ให้เป็น 256x256 หลังจากนั้นทำ FFT และ Shift ในการหา Amplitude ให้นำค่า shift ของ FFT ที่ได้ไป abs และ log ตามลำดับ ในกรณีของ Phase ให้นำ Shift ของ FFT ไปเข้าฟังก์ชั่น angle ก็จะได้รูปของ Phase ตามต้องการ



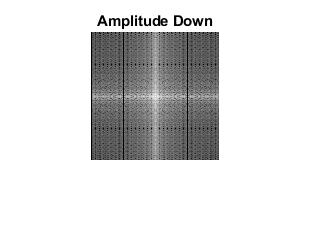
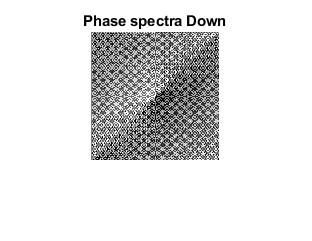
 1.2 นำ Phase ของข้อ 1.1 มาคูณกับ complex ที่มีค่า exp(-2j\*pi\*(((20\*X)/256)+((30\*Y)/256))) แล้วไปนำไปเข้ากระบวนการ IFFT คือ IFFTShift และ IFFT ตามลำดับ แล้วจึงนำไปหา abs ก็จะได้รูปมุมของรูป โดยรูปจะมีการเลื่อนแกนไป

x = 20 , y = 30 ดังรูป

 1.3 นำรูปที่ต้องการมาหมุนด้วยมุม 30 องศา หลังจาดนั้นก็เข้ากระบวนการ FFT เพื่อหา Amplitude และ Phase เหมือนข้อ 1.1 โดยผลลัพธ์ จะเห็นว่า รูปของ Amplitude นั้น เมื่อเทียบกับกรณีที่ยังไม่ได้หมุน รูปผลลัพธ์ที่ได้จะ หมุนไปในทิศทางเดียวกันกับที่เราหมุนรูป โดยได้ผลลัพธ์ดังรูป

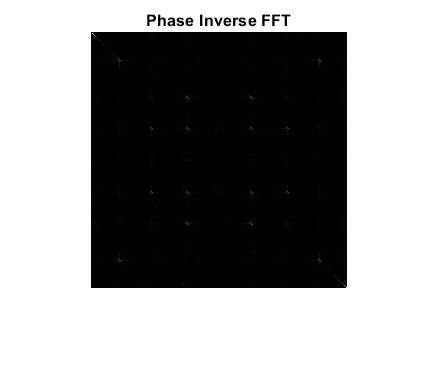
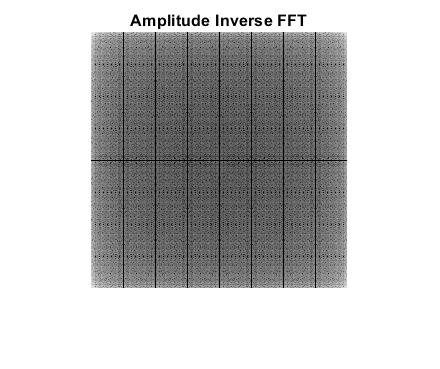


1.4 นำรูปที่ต้องการมาลดขนาดให้เท่าที่เราต้องการ ในกรณีของข้อนี้ ให้ลด ½ แล้วนำรูปที่ได้ไป Pad และทำกระบวนการเพื่อหา Amplitude และ Phase ตามข้อ 1.1 โดยผลลัพธ์ จะเห็นว่า รูป Amplitude และ Phase นั้น ความละเอียดจะลดลง ขนาดของรูปก็เช่นกัน แต่ในส่วนของรายละเอียดอื่น ๆ แทบไม่ได้แตกต่างกันเลย ดังรูป

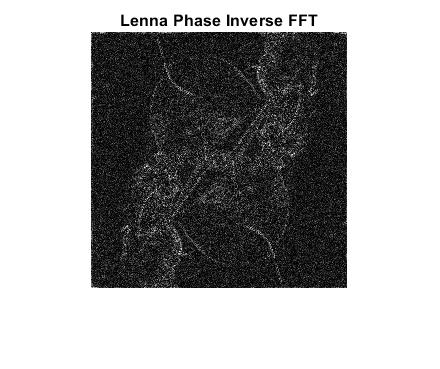
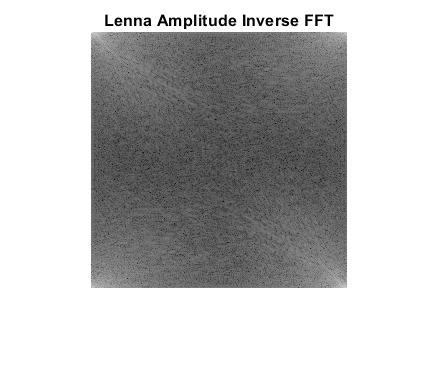


1.5.1 นำค่า Amplitude ในข้อ 1.1 มาทำกระบวนการ IFFT นั่นคือ IFFTShift และ IFFT ตามลำดับ ก็จะได้รูปของ IFFT โดยผลลัพธ์ จะออกมาเป็นข้อมูลความเข้มของแสงในรูป

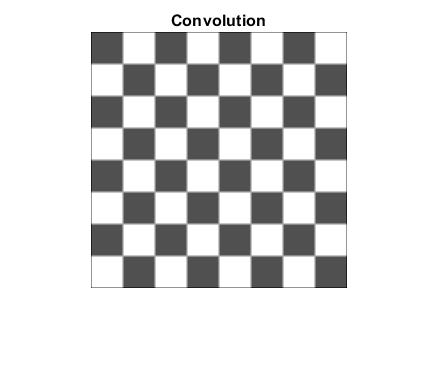
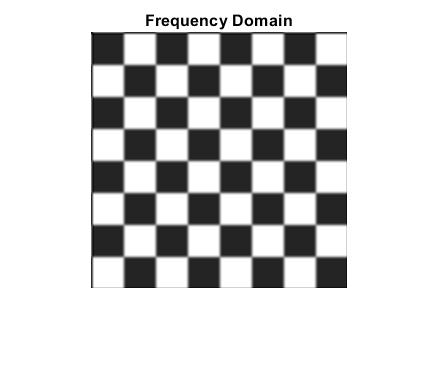
1.5.2 นำค่า Phase ในข้อ 1.1 มาทำเช่นเดียวกับข้อ 1.5.1 เลย แต่ในตอนท้ายก่อนได้ผลลัพธ์ต้องทำการหา abs ก่อน โดยผลลัพธ์ที่ออกมา จะมุมของรูป เหลี่ยมอย่างชัดเจน



1.6 ทำเหมือนข้อ 1.5.1 และ 1.5.2 เลย เพียงเปลี่ยนรูป input เป็น Lenna โดยผลลัพธ์ก็ออกมาในกรณีเดียวกันกับ ข้อ 1.5 เลย นั่นคือ เป็น ค่าความเข้มแสง และ มุมของรูป input ดังรูป



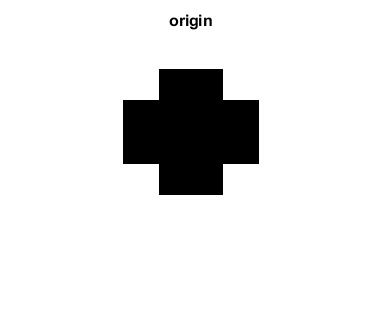
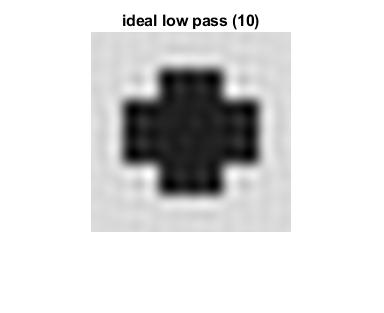
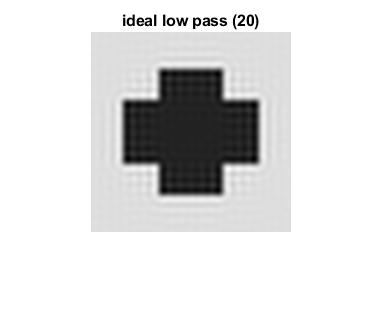
1.7 กำหนด kernel ที่มีขนาด 3x3 ขึ้นมา 1 อัน และนำไป Convolution กับรูป input ก็จะได้รูป ที่ blur ด้วย Convolution และ นำไป filter ด้วย FFT ของ kernel โดย เราจะนำ kernel ที่กำหนด มา pad ให้มีขนาดเท่ากันกับรูป input แล้วทำกระบวนการ FFT กับ kernel ที่ pad แล้ว เมื่อได้ค่ามา ให้นำค่านั้น ไปคูณ เข้ากับ ค่า FFT ของ รูปที่ blur ด้วย convolution และมาทำกระบวนการ IFFT ตามลำดับ โดย ผลลัพธ์ที่ได้ จะเห็นว่ารูปทั้ง 2 มีการเบลอ แต่ ในกรณีของ filter ด้วย frequency domain นั้น รูปที่ได้จะดูชัดเจนกว่ากรณีของ Blur ด้วย Convolution ดังรูป

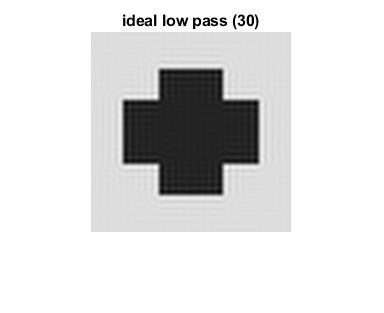
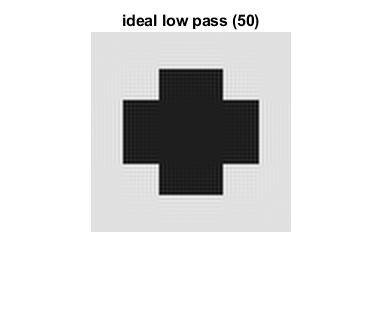


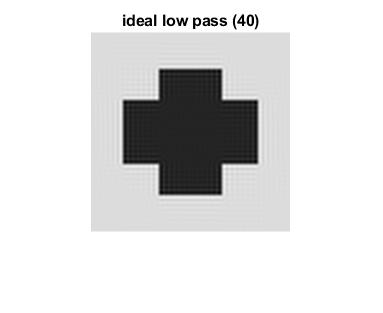
**2. Filter Design**

**2.1 Ideal low-pass filter**

โดยผลลัพธ์ทีได้นั้น เมื่อใช้ Cutoff ต่ำ Output ได้นั้นจะเกิด Ringing effect ทำให้มองเห็นภาพได้ไม่ชัด แต่เมื่อใช้ Cutoff ที่สูงขึ้น ภาพก็จะชัดขึ้นแต่ก็ยังเกิด Ring effect ขึ้นอยู่แต่ไม่มากเท่าในตอนที่ Cutoff น้อย ๆ และพื้นหลังของ output นั้นมีสีเทาไม่เป็นสีขาวตามรูป Input โดยได้ผลลัพธ์ดังรูป

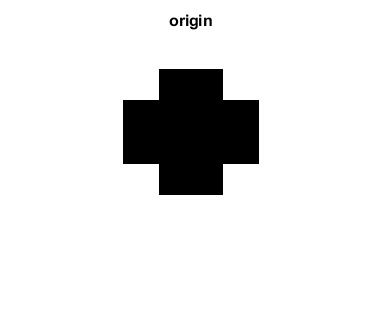
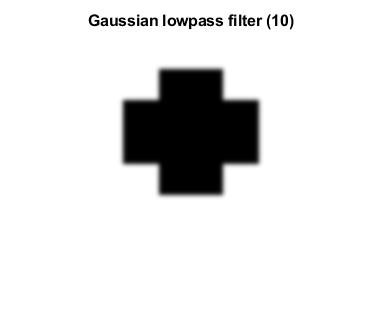
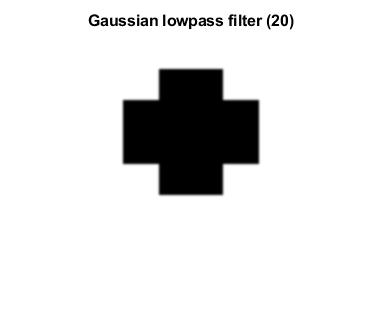


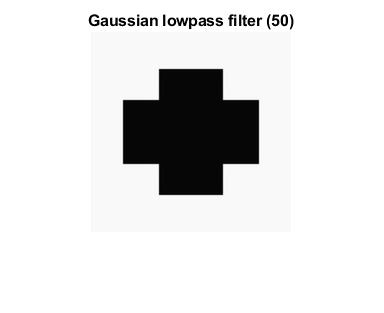
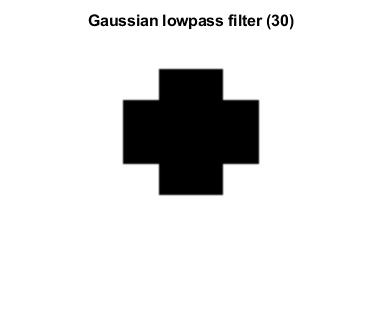


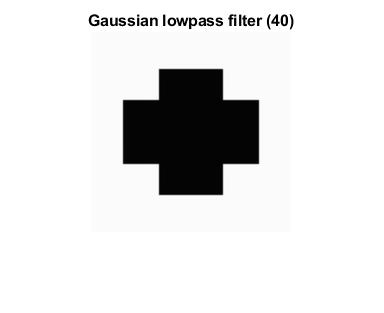


**Gaussian lowpass filter**

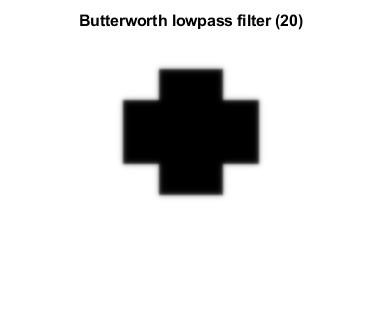
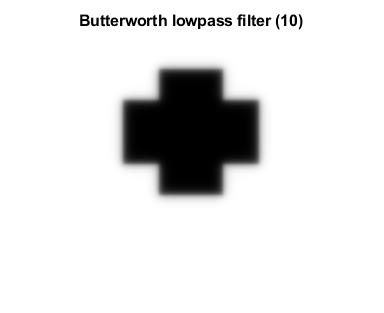
โดยผลลัพธ์ที่ได้นั้น เมื่อใช้ Cutoff ต่ำ ภาพ Output ที่ได้นั้นก็จะออกมา เบลอ และยิ่ง cutoff สูง ก็ยิ่งชัดขึ้นเรื่อย ๆ และ ในการ Filter นี้ไม่เกิด Ringing effect แต่พื้นหลังของ Output จะออกเทา มากกว่า รูป Input ที่ได้ก็ ออกมาเป็นสีขาวตาม Input ด้วย

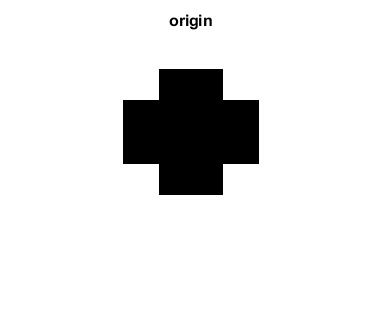
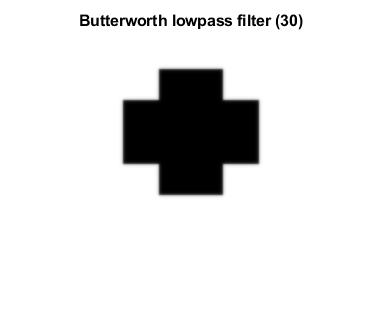
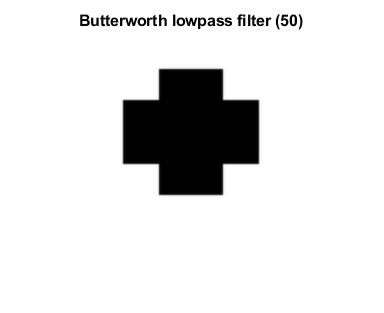


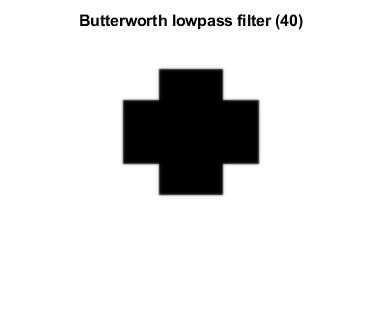




**Butterworth lowpass filter**

 กำหนด n = 1 โดยผลลัพธ์ที่ได้นั้น จะเหมือนกับ 2 กรณีที่ผ่านมา ที่เมื่อ Cutoff ต่ำ ภาพที่ได้ก็จะออกมา เบลอ โดยยิ่ง Cutoff สูงภาพก็ยิ่งชัดขึ้น แต่ในกรณีของ Butterworth นั้น จะไม่เกิด Ringing effect โดยได้ผลลัพธ์ดังรูป

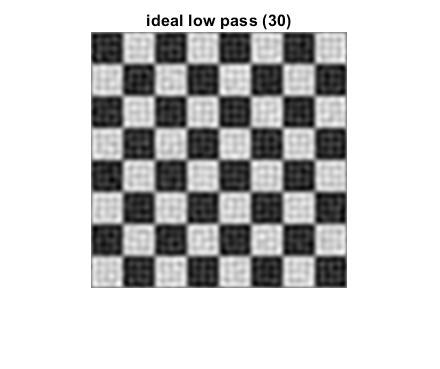
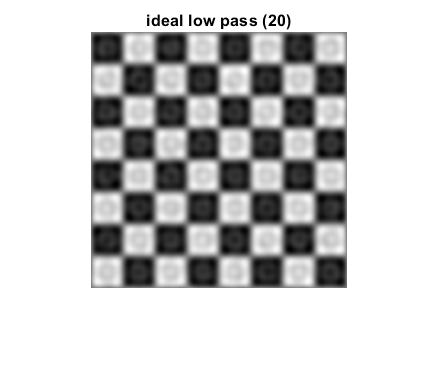
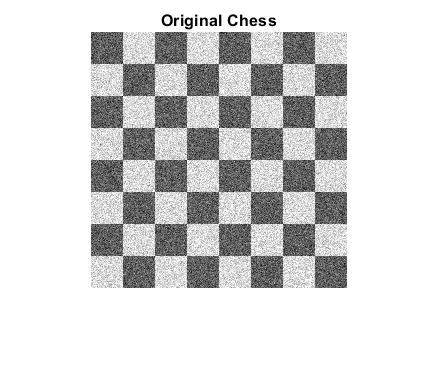
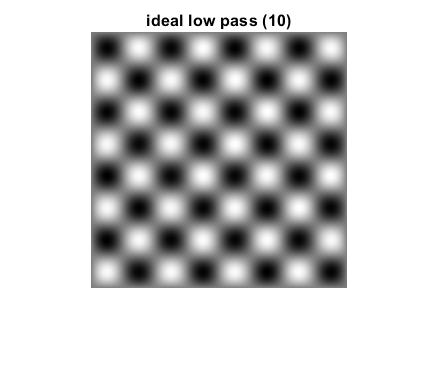


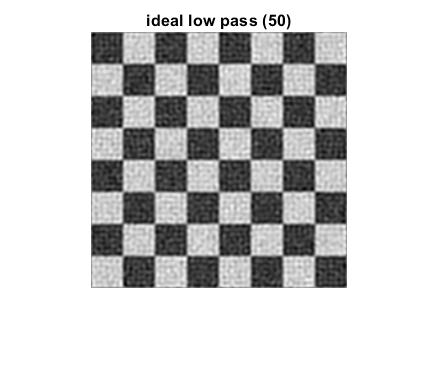
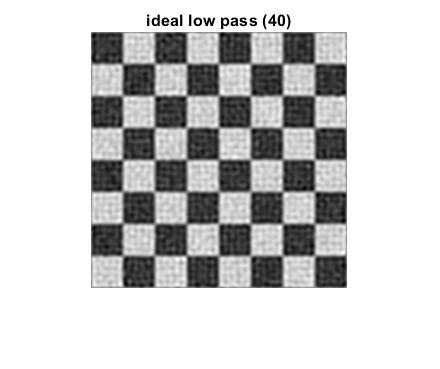


*2.2 ในการจะหา RMS เราจะต้อง นำค่า single ของ รูปต้นฉบับที่ไม่มี noise มาลบด้วย single ของรูปที่มี noise และนำไปหา sum ของค่า error ที่หารด้วย numel และนำค่าที่ได้ไปหา sqrt ก็จะได้ค่า RMS ที่ต้องการ โดยจะได้ผลลัพธ์ดังรูป*

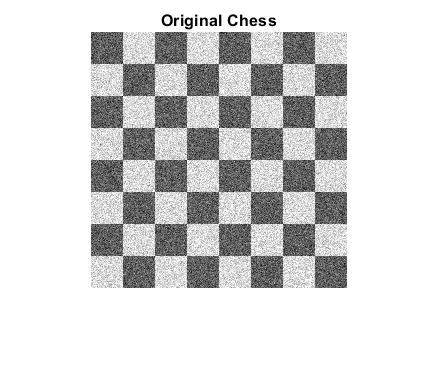
***กรณี Chess.pgm***

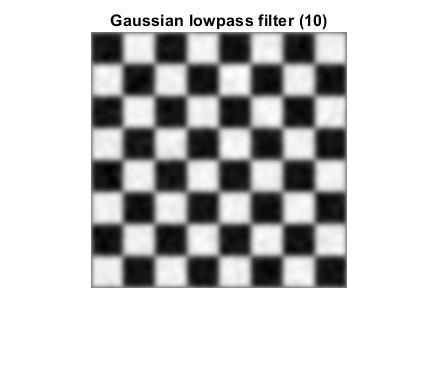
**Ideal low-pass filter**

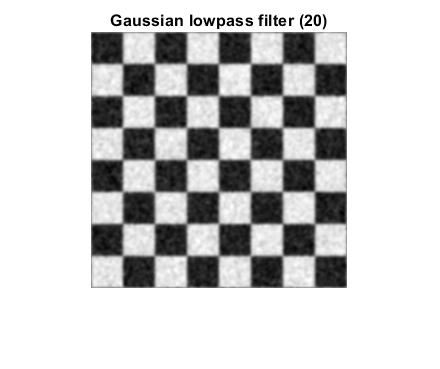
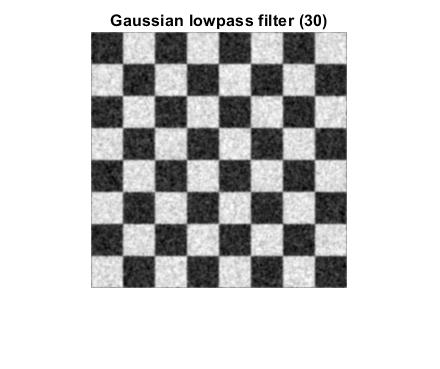
****

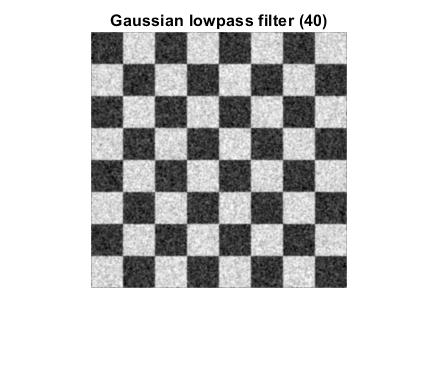
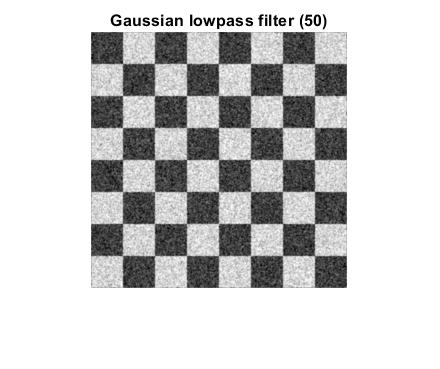


**RMS**   
Cutoff 10 = 35.1179   
Cutoff 20 = 22.1757 Cutoff 30 = 19.1067 Cutoff 40 = 17.5685 Cutoff 50 = 16.7716

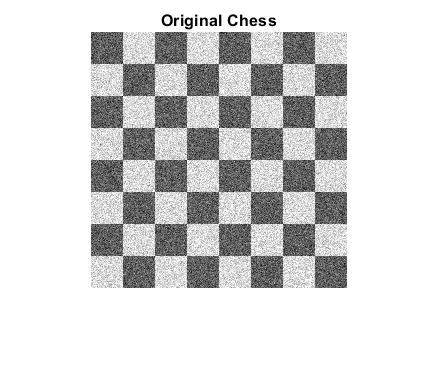
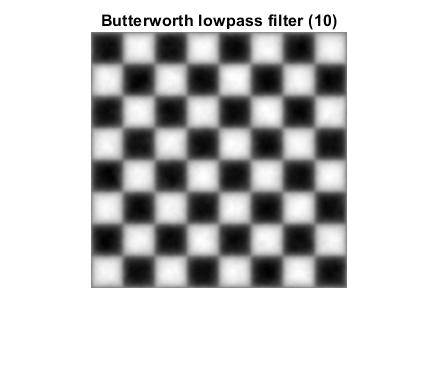
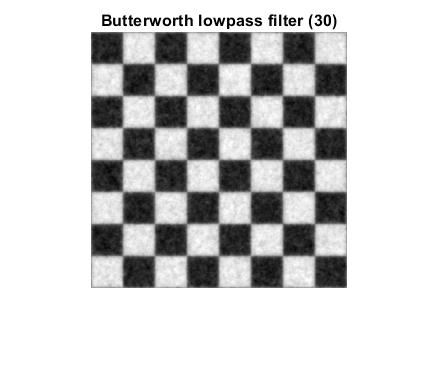
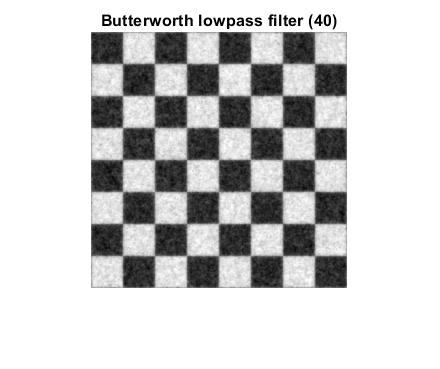
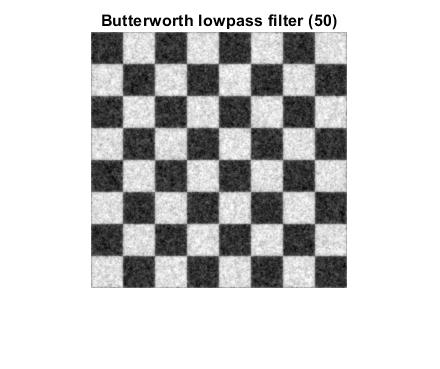
**Gaussian lowpass filter**

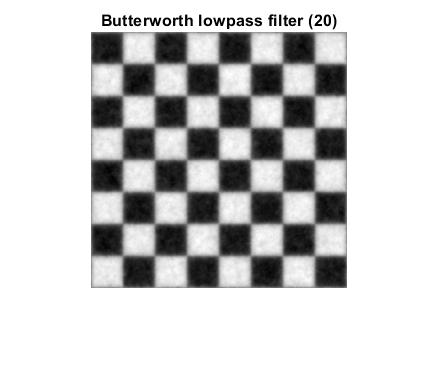






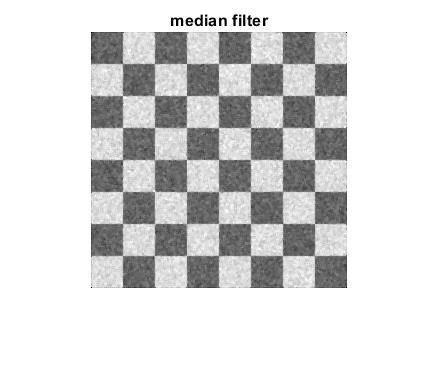
**RMS**   
Cutoff 10 = 24.6600  
Cutoff 20 = 17.4969 Cutoff 30 = 14.7397 Cutoff 40 = 13.8690 Cutoff 50 = 14.2413

**Butterworth lowpass filter**



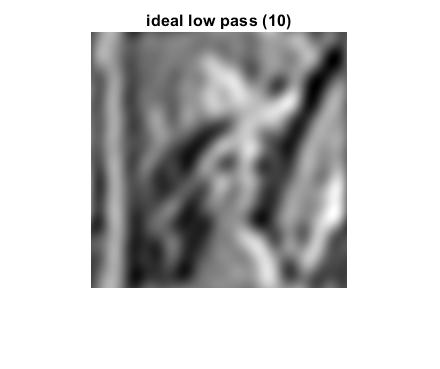
**RMS**   
Cutoff 10 = 30.5152   
Cutoff 20 = 21.3911 Cutoff 30 = 17.5166 Cutoff 40 = 15.4820 Cutoff 50 = 14.4482

**Median filter**



**RMS**   
Median Filter = 12.577671

**กรณี Lenna.pgm**

**Ideal low-pass filter**



**RMS**   
Cutoff 10 = 23.3992  
Cutoff 20 = 16.9965 Cutoff 30 = 13.9135 Cutoff 40 = 12.8150 Cutoff 50 = 12.4744

**Gaussian lowpass filter**





**RMS**   
Cutoff 10 = 17.3594  
Cutoff 20 = 12.2103 Cutoff 30 = 11.0236 Cutoff 40 = 11.6615 Cutoff 50 = 13.1515

**Butterworth lowpass filter**



**RMS**   
Cutoff 10 = 21.1669  
Cutoff 20 = 14.9381 Cutoff 30 = 12.3918 Cutoff 40 = 11.4126 Cutoff 50 = 11.3185

**Median filter**



**RMS**   
Median Filter = 12.95547

***ภาคผนวก***

% 1.1 %

ima = imread('Cross.pgm');

padimage = padarray(ima,[28 28],'both');

imagefft = fft2(double(padimage));

shiftedfft = fftshift(imagefft);

amp = log(abs(shiftedfft));

figure;

imshow(amp,[]);

colormap gray

title('Amplitude');

phase = angle(shiftedfft);

figure;

imshow(phase,[]);

colormap gray

title('Phase spectra');

% 1.2 %

x = -128:127;

y = -128:127;

[X,Y] = meshgrid(x,y);

complex\_number = exp(-2j\*pi\*(((20\*X)/256)+((30\*Y)/256)));

new\_phase = complex\_number.\*phase;

ifftshifted = ifftshift(new\_phase);

img\_ifft = ifft2(ifftshifted);

img\_ifft = abs(img\_ifft);

figure;

imshow(img\_ifft);

colormap gray

title('inverse fourier transform');

% 1.3 %

rotate\_img = imrotate(ima,30,'bilinear','crop');

rotate\_padimage = padarray(rotate\_img,[28 28],255,'both');

rotate\_imagefft = fft2(double(rotate\_padimage));

rotate\_shiftedfft = fftshift(rotate\_imagefft);

rotate\_amp = log(abs(rotate\_shiftedfft));

figure;

imshow(rotate\_padimage);

colormap gray

title('Image Rotate');

figure;

imshow(rotate\_amp,[]);

colormap gray

title('Amplitude Rotate');

rotate\_phase = angle(rotate\_shiftedfft);

figure;

imshow(rotate\_phase,[]);

colormap gray

title('Phase spectra Rotate');

% 1.4 %

ima\_down = ima(1:2:end,1:2:end);

padimage\_down = padarray(ima\_down,[14 14],'both');

imagefft\_down = fft2(double(padimage\_down));

shiftedfft\_down = fftshift(imagefft\_down);

amp\_down = log(abs(shiftedfft\_down));

figure;

imshow(amp\_down,[]);

colormap gray

title('Amplitude Down');

phase\_down = angle(shiftedfft\_down);

figure;

imshow(phase\_down,[]);

colormap gray

title('Phase spectra Down');

% 1.5.1 %

inverse\_amp = ifftshift(amp);

ifft\_amp = ifft2(inverse\_amp);

figure;

imshow(inverse\_amp,[]);

colormap gray

title('Amplitude Inverse FFT');

% 1.5.2 %

inverse\_phase = ifftshift(phase);

inverse\_phase = ifft2(inverse\_phase);

inverse\_phase = abs(inverse\_phase);

figure;

imshow(inverse\_phase,[]);

colormap gray

title('Phase Inverse FFT');

% 1.6 %

ima\_lenna = imread('Lenna.pgm');

imagefft\_lenna = fft2(double(ima\_lenna));

shiftedfft\_lenna = fftshift(imagefft\_lenna);

amp\_lenna = log(abs(shiftedfft\_lenna));

phase\_lenna = angle(shiftedfft\_lenna);

inverse\_amp\_lenna = ifftshift(amp\_lenna);

ifft\_amp\_lenna = ifft2(inverse\_amp\_lenna);

figure;

imshow(inverse\_amp\_lenna,[]);

colormap gray

title('Lenna Amplitude Inverse FFT');

inverse\_phase\_lenna = ifftshift(phase\_lenna);

inverse\_phase\_lenna = ifft2(inverse\_phase\_lenna);

inverse\_phase\_lenna = abs(inverse\_phase\_lenna);

figure;

imshow(inverse\_phase\_lenna,[]);

colormap gray

title('Lenna Phase Inverse FFT');

% 1.7 %

ima\_chess = imread('Chess.pgm');

ima\_chess = padarray(ima\_chess,[1 1],'both');

ima\_chess\_new = ima\_chess(2:end-1,2:end-1);

[X,Y] = size(ima\_chess\_new);

kernel = ones(3,3);

kernel = kernel./9;

for i = 1:X

for j = 1:Y

avg = 0;

for k = 1:3

for l = 1:3

avg = avg + ima\_chess(i+k-1,j+l-1).\*kernel(k,l);

end

end

ima\_chess\_new(i,j) = double(avg);

end

end

ima\_chess\_fft = fft2(double(ima\_chess\_new));

shifted\_ima\_chess\_fft = fftshift(ima\_chess\_fft);

pad\_kernel = padarray(kernel,[253 253],'post');

kernel\_fft = fft2(double(pad\_kernel));

shifted\_kernel\_fft = fftshift(kernel\_fft);

ima\_chess\_fillter = shifted\_kernel\_fft.\*shifted\_ima\_chess\_fft;

ima\_blur\_ifft = ifftshift(ima\_chess\_fillter);

ima\_blur\_ifft = ifft2(ima\_blur\_ifft);

ima\_blur\_ifft = abs(ima\_blur\_ifft);

figure;

imshow(ima\_chess\_new,[]);

colormap gray

title('Convolution');

figure;

imshow(ima\_blur\_ifft,[]);

colormap gray

title('Frequency Domain');

% 2.1 %

img = imread('Cross.pgm');

[X,Y] = size(img);

img\_fft = fft2(double(img));

figure

imshow(img,[]);

title('origin');

u = 1 : X;

v = 1 : Y;

currentx = find(u>X/2);

u(currentx) = u(currentx)-X;

currenty = find(v>Y/2);

v(currenty) = v(currenty)-Y;

[U,V] = meshgrid(u,v);

D = sqrt((U.^2) + (V.^2));

% ideal low pass %

H10 = double(D <= 10);

G10 = H10.\*img\_fft;

ideal10 = ifft2(G10);

figure;

imshow(ideal10,[]);

title('ideal low pass (10)');

H20 = double(D <= 20);

G20 = H20.\*img\_fft;

ideal20 = ifft2(G20);

figure;

imshow(ideal20,[]);

title('ideal low pass (20)');

H30 = double(D <= 30);

G30 = H30.\*img\_fft;

ideal30 = ifft2(G30);

figure;

imshow(ideal30,[]);

title('ideal low pass (30)');

H40 = double(D <= 40);

G40 = H40.\*img\_fft;

ideal40 = ifft2(G40);

figure;

imshow(ideal40,[]);

title('ideal low pass (40)');

H50 = double(D <= 50);

G50 = H50.\*img\_fft;

ideal50 = ifft2(G50);

figure;

imshow(ideal50,[]);

title('ideal low pass (50)');

% gaussian lowpass filter %

H\_glf10 = exp(-(D.^2)./(2.\*10).^2);

G\_glf10 = H\_glf10.\*img\_fft;

GLF10 = ifft2(G\_glf10);

figure;

imshow(GLF10,[]);

title('Gaussian lowpass filter (10)' );

H\_glf20 = exp(-(D.^2)./(2.\*20).^2);

G\_glf20 = H\_glf20.\*img\_fft;

GLF20 = ifft2(G\_glf20);

figure;

imshow(GLF20,[]);

title('Gaussian lowpass filter (20)' );

H\_glf30 = exp(-(D.^2)./(2.\*30).^2);

G\_glf30 = H\_glf30.\*img\_fft;

GLF30 = ifft2(G\_glf30);

figure;

imshow(GLF30,[]);

title('Gaussian lowpass filter (30)' );

H\_glf40 = exp(-(D.^2)./(2.\*40).^2);

G\_glf40 = H\_glf40.\*img\_fft;

GLF40 = ifft2(G\_glf40);

figure;

imshow(GLF40,[]);

title('Gaussian lowpass filter (40)' );

H\_glf50 = exp(-(D.^2)./(2.\*50).^2);

G\_glf50 = H\_glf50.\*img\_fft;

GLF50 = ifft2(G\_glf50);

figure;

imshow(GLF50,[]);

title('Gaussian lowpass filter (50)' );

% butterworth lowpass filter %

n = 1;

H\_blf10 = (1 + ((D./10).^(2\*n))).^(-1);

G\_blf10 = H\_blf10.\*img\_fft;

BLF10 = ifft2(G\_blf10);

figure;

imshow(BLF10,[]);

title('Butterworth lowpass filter (10)');

H\_blf20 = (1 + ((D./20).^(2\*n))).^(-1);

G\_blf20 = H\_blf20.\*img\_fft;

BLF20 = ifft2(G\_blf20);

figure;

imshow(BLF20,[]);

title('Butterworth lowpass filter (20)');

H\_blf30 = (1 + ((D./30).^(2\*n))).^(-1);

G\_blf30 = H\_blf30.\*img\_fft;

BLF30 = ifft2(G\_blf30);

figure;

imshow(BLF30,[]);

title('Butterworth lowpass filter (30)');

H\_blf40 = (1 + ((D./40).^(2\*n))).^(-1);

G\_blf40 = H\_blf40.\*img\_fft;

BLF40 = ifft2(G\_blf40);

figure;

imshow(BLF40,[]);

title('Butterworth lowpass filter (40)');

H\_blf50 = (1 + ((D./50).^(2\*n))).^(-1);

G\_blf50 = H\_blf50.\*img\_fft;

BLF50 = ifft2(G\_blf50);

figure;

imshow(BLF50,[]);

title('Butterworth lowpass filter (50)');

% 2.2 %

%img\_noise = imread('Lenna\_noise.pgm');

img\_ori = imread('Lenna.pgm');

img\_noise = imread('Chess\_noise.pgm');

%img\_ori = imread('Chess.pgm');

[X,Y] = size(img\_noise);

img\_fft = fft2(double(img\_noise));

u = 1 : X;

v = 1 : Y;

currentx = find(u>X/2);

u(currentx) = u(currentx)-X;

currenty = find(v>Y/2);

v(currenty) = v(currenty)-Y;

[U,V] = meshgrid(u,v);

D = sqrt((U.^2) + (V.^2));

% ideal low pass %

H10 = double(D <= 10);

G10 = H10.\*img\_fft;

ideal10 = ifft2(G10);

figure;

imshow(ideal10,[]);

title('ideal low pass (10)');

H20 = double(D <= 20);

G20 = H20.\*img\_fft;

ideal20 = ifft2(G20);

figure;

imshow(ideal20,[]);

title('ideal low pass (20)');

H30 = double(D <= 30);

G30 = H30.\*img\_fft;

ideal30 = ifft2(G30);

figure;

imshow(ideal30,[]);

title('ideal low pass (30)');

H40 = double(D <= 40);

G40 = H40.\*img\_fft;

ideal40 = ifft2(G40);

figure;

imshow(ideal40,[]);

title('ideal low pass (40)');

H50 = double(D <= 50);

G50 = H50.\*img\_fft;

ideal50 = ifft2(G50);

figure;

imshow(ideal50,[]);

title('ideal low pass (50)');

% gaussian lowpass filter %

H\_glf10 = exp(-(D.^2)./(2.\*10).^2);

G\_glf10 = H\_glf10.\*img\_fft;

GLF10 = ifft2(G\_glf10);

figure;

imshow(GLF10,[]);

title('Gaussian lowpass filter (10)' );

H\_glf20 = exp(-(D.^2)./(2.\*20).^2);

G\_glf20 = H\_glf20.\*img\_fft;

GLF20 = ifft2(G\_glf20);

figure;

imshow(GLF20,[]);

title('Gaussian lowpass filter (20)' );

H\_glf30 = exp(-(D.^2)./(2.\*30).^2);

G\_glf30 = H\_glf30.\*img\_fft;

GLF30 = ifft2(G\_glf30);

figure;

imshow(GLF30,[]);

title('Gaussian lowpass filter (30)' );

H\_glf40 = exp(-(D.^2)./(2.\*40).^2);

G\_glf40 = H\_glf40.\*img\_fft;

GLF40 = ifft2(G\_glf40);

figure;

imshow(GLF40,[]);

title('Gaussian lowpass filter (40)' );

H\_glf50 = exp(-(D.^2)./(2.\*50).^2);

G\_glf50 = H\_glf50.\*img\_fft;

GLF50 = ifft2(G\_glf50);

figure;

imshow(GLF50,[]);

title('Gaussian lowpass filter (50)' );

% butterworth lowpass filter %

n = 1;

H\_blf10 = (1 + ((D./10).^(2\*n))).^(-1);

G\_blf10 = H\_blf10.\*img\_fft;

BLF10 = ifft2(G\_blf10);

figure;

imshow(BLF10,[]);

title('Butterworth lowpass filter (10)');

H\_blf20 = (1 + ((D./20).^(2\*n))).^(-1);

G\_blf20 = H\_blf20.\*img\_fft;

BLF20 = ifft2(G\_blf20);

figure;

imshow(BLF20,[]);

title('Butterworth lowpass filter (20)');

H\_blf30 = (1 + ((D./30).^(2\*n))).^(-1);

G\_blf30 = H\_blf30.\*img\_fft;

BLF30 = ifft2(G\_blf30);

figure;

imshow(BLF30,[]);

title('Butterworth lowpass filter (30)');

H\_blf40 = (1 + ((D./40).^(2\*n))).^(-1);

G\_blf40 = H\_blf40.\*img\_fft;

BLF40 = ifft2(G\_blf40);

figure;

imshow(BLF40,[]);

title('Butterworth lowpass filter (40)');

H\_blf50 = (1 + ((D./50).^(2\*n))).^(-1);

G\_blf50 = H\_blf50.\*img\_fft;

BLF50 = ifft2(G\_blf50);

figure;

imshow(BLF50,[]);

title('Butterworth lowpass filter (50)');

% RMS ideal low pass %

ideal\_er10 = (single(img\_ori) - single(ideal10)).^2;

ideal\_mean10 = sum(ideal\_er10(:)) / numel(img\_ori);

ideal\_rmsError10 = sqrt(ideal\_mean10);

ideal\_er20 = (single(img\_ori) - single(ideal20)).^2;

ideal\_mean20 = sum(ideal\_er20(:)) / numel(img\_ori);

ideal\_rmsError20 = sqrt(ideal\_mean20);

ideal\_er30 = (single(img\_ori) - single(ideal30)).^2;

ideal\_mean30 = sum(ideal\_er30(:)) / numel(img\_ori);

ideal\_rmsError30 = sqrt(ideal\_mean30);

ideal\_er40 = (single(img\_ori) - single(ideal40)).^2;

ideal\_mean40 = sum(ideal\_er40(:)) / numel(img\_ori);

ideal\_rmsError40 = sqrt(ideal\_mean40);

ideal\_er50 = (single(img\_ori) - single(ideal50)).^2;

ideal\_mean50 = sum(ideal\_er50(:)) / numel(img\_ori);

ideal\_rmsError50 = sqrt(ideal\_mean50);

% RMS gaussian lowpass filter %

GLF\_er10 = (single(img\_ori) - single(GLF10)).^2;

GLF\_mean10 = sum(GLF\_er10(:)) / numel(img\_ori);

GLF\_rmsError10 = sqrt(GLF\_mean10);

GLF\_er20 = (single(img\_ori) - single(GLF20)).^2;

GLF\_mean20 = sum(GLF\_er20(:)) / numel(img\_ori);

GLF\_rmsError20 = sqrt(GLF\_mean20);

GLF\_er30 = (single(img\_ori) - single(GLF30)).^2;

GLF\_mean30 = sum(GLF\_er30(:)) / numel(img\_ori);

GLF\_rmsError30 = sqrt(GLF\_mean30);

GLF\_er40 = (single(img\_ori) - single(GLF40)).^2;

GLF\_mean40 = sum(GLF\_er40(:)) / numel(img\_ori);

GLF\_rmsError40 = sqrt(GLF\_mean40);

GLF\_er50 = (single(img\_ori) - single(GLF50)).^2;

GLF\_mean50 = sum(GLF\_er50(:)) / numel(img\_ori);

GLF\_rmsError50 = sqrt(GLF\_mean50);

% RMS butterworth lowpass filter %

BLF\_er10 = (single(img\_ori) - single(BLF10)).^2;

BLF\_mean10 = sum(BLF\_er10(:)) / numel(img\_ori);

BLF\_rmsError10 = sqrt(BLF\_mean10);

BLF\_er20 = (single(img\_ori) - single(BLF20)).^2;

BLF\_mean20 = sum(BLF\_er20(:)) / numel(img\_ori);

BLF\_rmsError20 = sqrt(BLF\_mean20);

BLF\_er30 = (single(img\_ori) - single(BLF30)).^2;

BLF\_mean30 = sum(BLF\_er30(:)) / numel(img\_ori);

BLF\_rmsError30 = sqrt(BLF\_mean30);

BLF\_er40 = (single(img\_ori) - single(BLF40)).^2;

BLF\_mean40 = sum(BLF\_er40(:)) / numel(img\_ori);

BLF\_rmsError40 = sqrt(BLF\_mean40);

BLF\_er50 = (single(img\_ori) - single(BLF50)).^2;

BLF\_mean50 = sum(BLF\_er50(:)) / numel(img\_ori);

BLF\_rmsError50 = sqrt(BLF\_mean50);

% Median filter %

img\_noise\_pad = padarray(img\_noise , [1 1] , 'both');

[X,Y] = size(img\_noise);

img\_noise\_new = img\_noise\_pad(2:end-1,2:end-1);

arr = ones(9,1);

for i = 1:X

for j = 1:Y

current = 1;

for m = 1:3

for n = 1:3

arr(current) = img\_noise\_pad(i+m-1,j+n-1);

current = current + 1;

end

end

MF(i,j) = median(arr);

end

end

figure;

imshow(MF,[]);

title('median filter');

% RMS Median filter %

MF\_er = (single(img\_ori) - single(MF)).^2;

MF\_mean = sum(MF\_er(:)) / numel(img\_ori);

MF\_rmsError = sqrt(MF\_mean);