MTSC 887 - Image Processing
Digital Image Fundamentals
Lab Session

Sokratis Makrogiannis, Ph.D.

January 29, 2015

Contents

1 Image Reading/Writing

- Read and display image information.

```
imfinfo('Fig0221(a)(ctskull-256).tif')
```

or

```
imfinfo('rice.png')
```

You should receive the following output for the skull CT image as shown in Figure 1.

- Read image file.

```
ans =

          Filename: [1x114 char]
       FileModDate: '22-Feb-2001 00:13:14'
          FileSize: 144254
            Format: 'tif'
     FormatVersion: []
             Width: 374
            Height: 452
          BitDepth: 8
         ColorType: 'grayscale'
   FormatSignature: [73 73 42 0]
         ByteOrder: 'little-endian'
    NewSubFileType: 0
     BitsPerSample: 8
       Compression: 'PackBits'
PhotometricInterpretation: 'BlackIsZero'
       StripOffsets: [1x22 double]
    SamplesPerPixel: 1
       RowsPerStrip: 21
    StripByteCounts: [1x22 double]
        XResolution: 72
        YResolution: 72
     ResolutionUnit: 'Inch'
           Colormap: []
  PlanarConfiguration: 'Chunky'
         TileWidth: []
        TileLength: []
       TileOffsets: []
    TileByteCounts: []
       Orientation: 1
          FillOrder: 1
  GrayResponseUnit: 0.0100
     MaxSampleValue: 255
     MinSampleValue: 0
      Thresholding: 1
            Offset: 143888
```

Figure 1: Image file info.

```
Image = imread('Fig0221(a)(ctskull-256).tif');
```

or

```
Image = imread('rice.png');
```

- Display matlab's workspace variables.

```
whos
```

- Display Image Size.

```
size(Image)
```

- Display image file using 'imshow'.

```
figure,imshow(Image)
```

The output is shown in Figure 2.

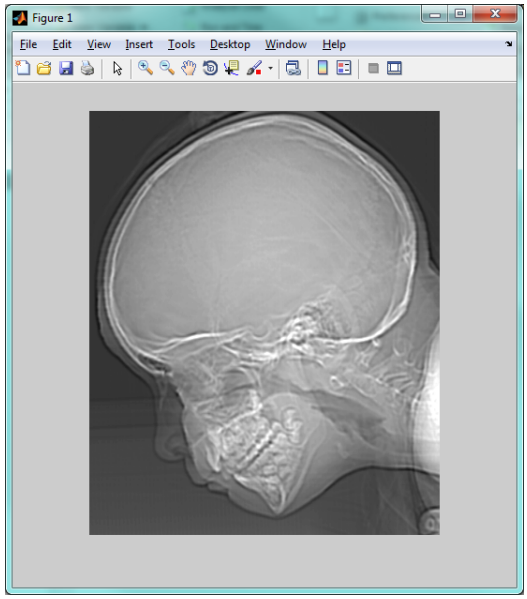


Figure 2: Matlab's imshow output.

- Display image file using 'imagesc'.

```
figure, imagesc(Image), colormap gray, axis image
```

- Write image to file using 'imwrite'.

```
imwrite(Image, 'Ex1-1.png');
```

- Clear matlab's memory, close figures, and clear the command window.

```
clear all, close all, clc
```

Exercise 1. *Read, display, and write three random images from the test set.*

2 Image Representation

2.1 Grayscale image

- Read image file.

```
Image = imread('Fig0221(a)(ctskull-256).tif');
```

- Display part of the image as a numerical array.

```
Image(80:90, 80:90)
```

The output is displayed in Figure 3.

```
ans =

  140  137  140  143  144  144  141  140  137  137  137
  143  141  141  143  143  143  143  143  141  141  140
  146  144  143  141  141  143  144  146  144  143  141
  146  144  141  140  140  143  146  147  146  144  143
  144  143  141  140  140  144  146  147  147  147  144
  141  141  141  143  144  146  147  147  147  147  144
  137  140  143  144  146  147  147  147  147  147  144
  136  140  143  144  146  147  147  147  147  147  146
  137  140  141  143  144  146  147  150  150  147  146
  141  141  141  143  144  147  150  152  152  150  147
  143  143  141  144  146  150  152  153  153  152  150
```

Figure 3: Numeric image representation.

- Display by different colormaps.

```
figure, imagesc(Image), axis image, colormap gray, colorbar

figure, imagesc(Image), axis image, colormap jet, colorbar
```

```
figure, imagesc(Image), axis image, colormap hot, colorbar
```

The output is displayed in Figure 4.

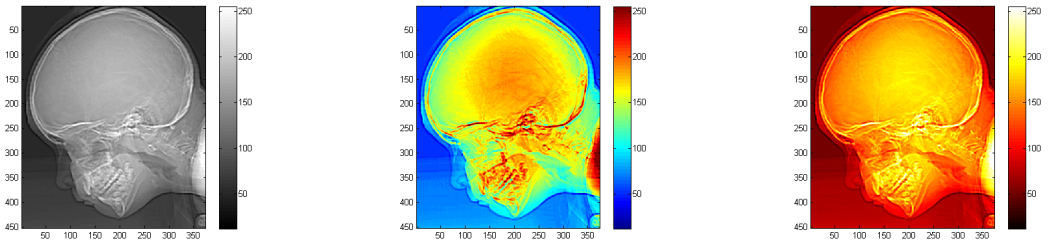


Figure 4: Displaying images using gray, jet and hot colormaps from left to right.

- Clear matlab's memory, close figures, and clear the command window.

```
clear all, close all, clc
```

Exercise 2. *Use matlab's help function to find the number of available colormaps. Why should we use one over the other?*

2.2 Color converted to grayscale image

- Read color image file.

```
RGB = imread('saturn.png');
```

- Convert to grayscale.

```
GrayImage = rgb2gray(RGB);
```

- Display part of the image as a numerical array.

```
GrayImage(80:90, 80:90)
```

- Display image.

```
figure, imagesc(GrayImage), axis image, colormap gray ,
    colorbar
```

- Clear matlab's memory, close figures, and clear the command window.

```
clear all, close all, clc
```

Exercise 3. *Use matlab's surf and mesh functions to display the grayscale image as a surface.*

3 Color Image Representation

- Read image file.

```
Image = imread('saturn.png');
```

- Display part of the image as a numerical array.

```
Image(80:90, 80:90, :)
```

- Display image.

```
figure, imagesc(Image), axis image, colorbar
```

- Clear matlab's memory, close figures, and clear the command window.

```
clear all, close all, clc
```

The color image and the converted to grayscale are displayed in Figure 5.

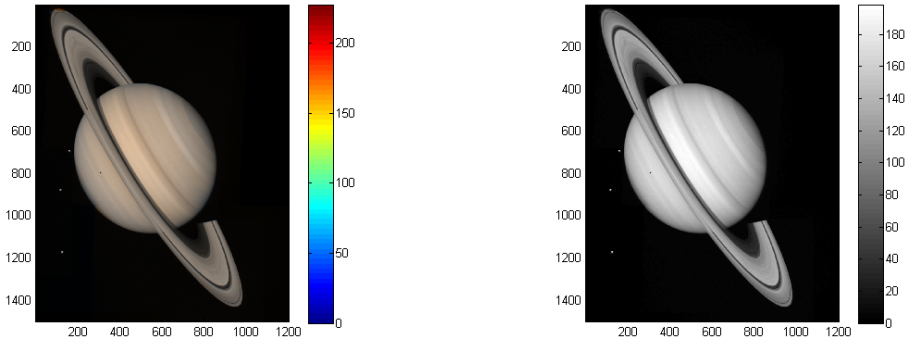


Figure 5: Displaying a color image and the converted to grayscale (left to right).

Exercise 4. *Use matlab's figure functionalities to read the (R,G,B) values of specific pixels.*

MTSC 887 - Image Processing
Digital Image Fundamentals
Lab Session

Sokratis Makrogiannis, Ph.D.

January 29, 2015

Contents

1 Quantization

1. Read image file.

```
Image = imread('jetplane.png');
```

2. Change number of bits per pixel from 8 to 5 to 4.

```
V = 1:8:255;
I_32 = imquantize(Image, V);
V = 1:16:255;
I_16 = imquantize(Image, V);
```

3. Display all images in the same figure. Result is shown in Figure 1.

```
figure, subplot(221), imagesc(Image), axis image, colormap
    gray, colorbar, title('Original');

subplot(222), imagesc(I_32), axis image, colormap gray,
    colorbar, title('32 levels');

subplot(223), imagesc(I_16), axis image, colormap gray,
    colorbar, title('16 levels');
```

4. Clear matlab's memory, close figures, and clear the command window.

```
clear all, close all, clc
```

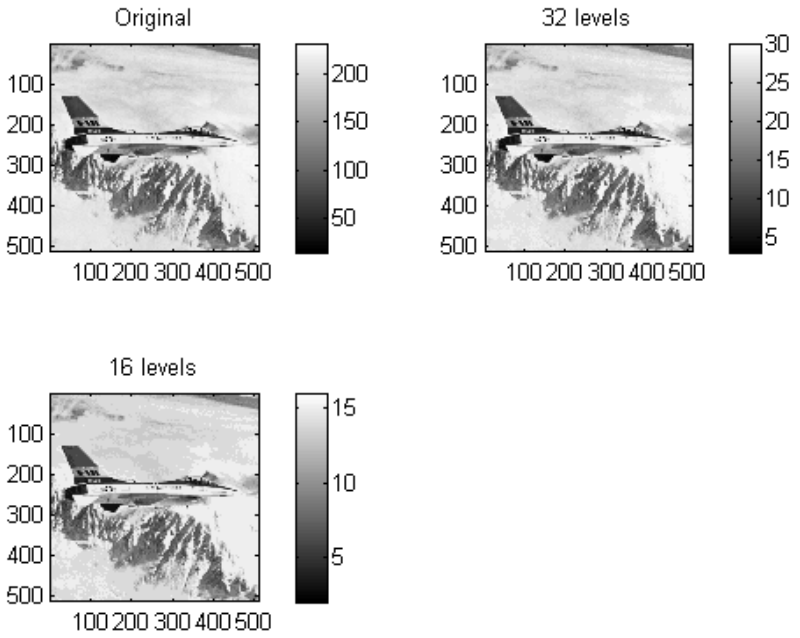


Figure 1: Jetplane image at different quantization levels.

Exercise 1. *Observe the quantized images and explain the effect of quantization.*

2 Spatial Relationships

1. Create a black and white image with three non-zero points.

```
bw = zeros(200,200); bw(50,50) = 1; bw(50,150) = 1;
bw(150,100) = 1;
```

2. Compute Euclidean distance transform.

```
D1 = bwdist(bw,'euclidean');
```

3. Compute City-block distance transform.

```
D2 = bwdist(bw,'cityblock');
```

4. Compute chessboard distance transform.

```
D3 = bwdist(bw,'chessboard');
```

5. Display all generated distance images. Result is shown in Figure 2.

```
figure
subplot(2,2,1), subimage(mat2gray(D1)), title('Euclidean'),
hold on, imcontour(D1);

subplot(2,2,2), subimage(mat2gray(D2)), title('City block'),
hold on, imcontour(D2);

subplot(2,2,3), subimage(mat2gray(D3)), title('Chessboard'),
hold on, imcontour(D3);
```

Exercise 2. *Create a zero-valued* 100×100 *image array and add a point at location* $(50, 50)$*. Calculate Euclidean, City-block, and chessboard distance*

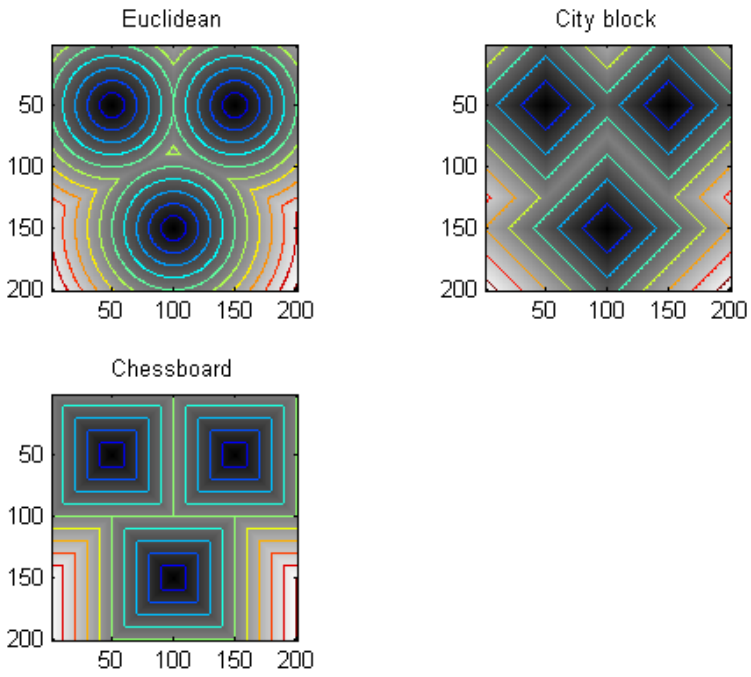


Figure 2: Distance maps created by three points.

maps using matlab. Confirm numerical values at location $(75, 75)$ *using manual calculations.*

3 Linear and Nonlinear Operations - Image Averaging

1. Read image file.

```
Image = imread('livingroom.png');
```

2. Add Gaussian noise with zero mean and 0.1 variance and create a noise corrupted image.

```
NoisyImage = imnoise(Image, 'gaussian', 0, 0.1);
```

3. Display original and noisy images.

```
figure, imagesc(Image), axis image, colormap gray, colorbar,
    title('Original Image');

figure, imagesc(NoisyImage), axis image, colormap gray,
    colorbar, title('Image Corrupted by Noise');
```

4. Create a sum image with zeros only.

```
SumImage = zeros(size(Image));
```

5. Write a loop structure that iteratively generates a noisy image, adds it to the original image and creates a sum image. Use 10 iterations.

```
for i=1:10,

NoisyImage = imnoise(Image, 'gaussian', 0, 0.1);

SumImage = SumImage + double(NoisyImage);

end
```

6. Calculate average image.

```
AverageImage = SumImage / 10;
```

7. Display average image. Observe that the noise has been reduced. Output is displayed Figure 3.

```
figure, imagesc(AverageImage), axis image, colormap gray,
    colorbar;
title('Average Image');
```

Exercise 3. *Repeat steps 4 to 7 for 50, 100 and 1000 iterations. Observe the improvement in image quality. Explain why there is a noise reduction. What is the probabilistic interpretation of the above procedure?*

Figure 3: Image averaging for noise removal.

4 Probabilistic Methods

1. Read an image files from our test set.

```
Image = imread('house.png');
```

2. Add Gaussian noise with zero mean and 0.1 variance and create a noise

corrupted image.

```
NoisyImage = imnoise(Image, 'gaussian', 0, 0.2);
```

3. Display original and noisy images. Results are displayed in Figure 4.

```
figure, imagesc(Image), axis image, colormap gray, colorbar;

figure, imagesc(NoisyImage), axis image, colormap gray,
    colorbar;
```

4. For each image:

   (a) Compute average pixel intensity.

```
ImageMean = mean(Image(:))

NoisyImageMean = mean(NoisyImage(:))
```

   (b) Compute standard deviation of pixel intensities.

```
ImageStandardDev = std(double(Image(:)))

NoisyImageStandardDev = std(double(NoisyImage(:)))
```

Exercise 4. *Repeat the above process using standard deviations* $\sigma = \{0.4, 0.8\}$. *Compare the measures and explain any similarities or differences.*

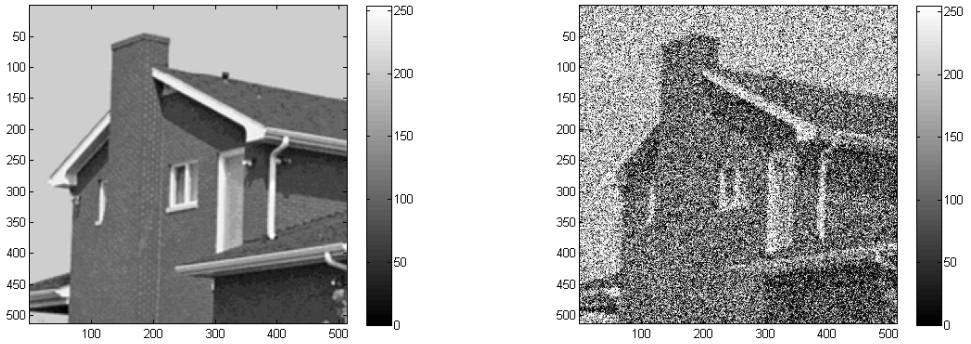


Figure 4: Displaying the original image and the image corrupted by Gaussian noise.(left to right).

5 How to convert a Matlab script to a function.

1. Create a new script in Matlab's workspace.
2. Copy all commands in the script window.
3. Create a function header in the first line of your program in the following form:

```
function [OutputArgument1, OutputArgument2] =
    FunctionName(InputArgument1, InputArgument2, ...,
    InputArgumentN);
```

4. Make sure to match the input and output arguments with variables in your script.
5. Save your file as FunctionName.m.
6. Well done!

Example of a Matlab function:

```
function AverageImage = ImageAveraging(filename, nIterations)
% syntax: AverageImage = ImageAveraging(filename, nIterations);
% Example: AverageImage = ImageAveraging('livingroom.png', 50);

% Read image file.
Image = imread(filename);

% Add Gaussian noise with zero mean and 0.1 variance and create a
    noise corrupted image.
NoisyImage = imnoise(Image, 'gaussian', 0, 0.1);


% Display original and noisy images.
figure, imagesc(Image), axis image, colormap gray, colorbar;
title('Original Image');
figure, imagesc(NoisyImage), axis image, colormap gray, colorbar;
title('Image Corrupted by Noise');

% Create a sum image with zeros only.
SumImage = zeros(size(Image));

% Write a loop structure that iteratively generates a noisy image,
    adds it to the original image and creates a sum image. Use 10
    iterations.

for i=1:nIterations,
NoisyImage = imnoise(Image, 'gaussian', 0, 0.1);
SumImage = SumImage + double(NoisyImage);
end

% Calculate average image.
AverageImage = SumImage / nIterations;

% Display average image. Observe that the noise has been reduced.
figure, imagesc(AverageImage), axis image, colormap gray, colorbar;
title('Average Image');

end
```

MTSC 887 - Image Processing
Intensity Transformations
Lab Session

Sokratis Makrogiannis, Ph.D.

February 5, 2015

Contents

1 Matlab's Demos

1. Check list of Matlab's image processing demos, sample mat-files and sample images.

```
help imdemos
```

2 Intensity Mapping

2.1 Image Negatives

1. Read image file.

```
Image = imread('peppers_gray.png');
```

2. Calculate negative image $s = (L - 1) - r$ for $L = 256$.

```
NegativeImage = 255 - Image;
```

3. Display original and negative images. Results are displayed in Figure 1.

```
figure, imagesc(Image), axis image, colormap gray, colorbar;
figure, imagesc(NegativeImage), axis image, colormap gray,
    colorbar;
```

4. Save negative image to file.

```
saveas(gcf, 'peppers_negative.png')
```

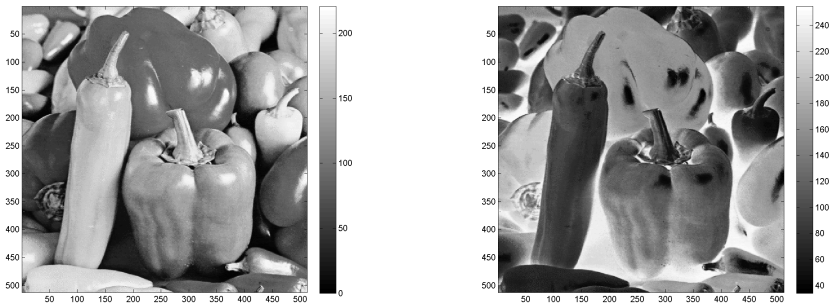


Figure 1: Original and negative images.

2.2 Log Transform

1. Read image file.

```
Image = imread('cameraman.png');
```

2. Convert pixel type to double.

```
Image = double(Image);
```

3. Apply log intensity transformation.

```
LogImage = 10*log10(1+Image);
```

4. Display results (also in Figure 2).

```
figure, imagesc(Image), axis image, colormap gray, colorbar;
figure, imagesc(LogImage), axis image, colormap gray,
    colorbar;
saveas(gcf, 'cameraman_log_transformed.png')
```

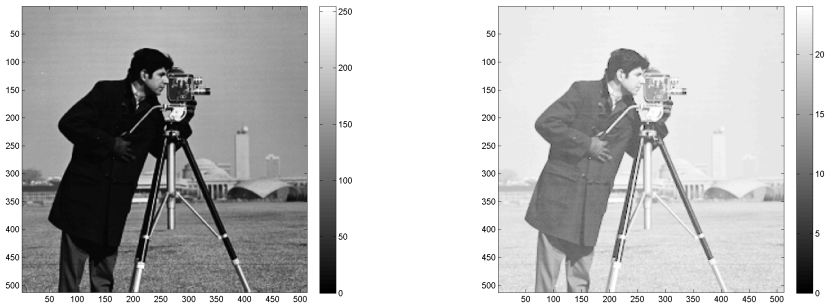


Figure 2: Original and log transformed images.

2.3 Gamma Transform

1. Read image and convert to double pixel type.

```
Image = imread('lena_gray_256.png');
Image = double(Image);
```

2. Set c and then iteratively set variable values of γ and calculate $s = c \cdot r^{\gamma}$. Results are also displayed in Figure 3.

```
c = 1;
for gamma = 0.05:2:10, GammaImage = c * Image.^gamma; figure,
    imagesc(GammaImage), axis image, colormap gray,
    colorbar;end
```

Figure 3: Gamma transformed images for $\gamma = \{0.05, 2.05, 4.05\}$.

Exercise 1. *Implement the following processes: 1) read the cameraman image, 2) calculate negative image, 3) add a constant value of 50, 4) apply gamma intensity transform with* $\gamma \in [0, 2.0]$ *with a step of 0.4. Report your observations with respect to contrast stretching.*

3 Piecewise-linear Transforms

3.1 Contrast Stretching

1. Read image file and convert to double.

```
Image = imread('woman_darkhair.png');
Image = double(Image);
figure, imagesc(Image), axis image, colormap gray, colorbar;
```

2. Define the piecewise linear function.

```
a = [0 50 150 255];
b = [0 30 200 255];
figure, plot(a, b, 'linewidth', 2), title('Piecewise linear
    intensity mapping function.'), grid on;
saveas(gcf, 'piecewise_linear_mapping.png');
```

3. Create output image.

```
Image2 = zeros(size(Image));
```

4. For each linear interval

   (a) Find the pixels with intensities in the interval.

   (b) Calculate the equation of line.

   (c) Assign the transformed intensities to output image. Results are displayed in Figure 4.

```
i=1;
indices = find(Image>=a(i) & Image <= a(i+1));
Image2(indices) = (Image(indices) -
    a(i))*(b(i+1)-b(i))/(a(i+1)-a(i)) + b(i);
figure, imagesc(Image2), axis image, colormap gray, colorbar;
i=2;
indices = find(Image>=a(i) & Image <= a(i+1));
Image2(indices) = (Image(indices) -
    a(i))*(b(i+1)-b(i))/(a(i+1)-a(i)) + b(i);
figure, imagesc(Image2), axis image, colormap gray, colorbar;
```

```
i=3;
indices = find(Image>=a(i) & Image <= a(i+1));
Image2(indices) = (Image(indices) -
    a(i))*(b(i+1)-b(i))/(a(i+1)-a(i)) + b(i);
figure, imagesc(Image2), axis image, colormap gray, colorbar;
saveas(gcf, 'woman_darkhair_post_mapping.png');
```

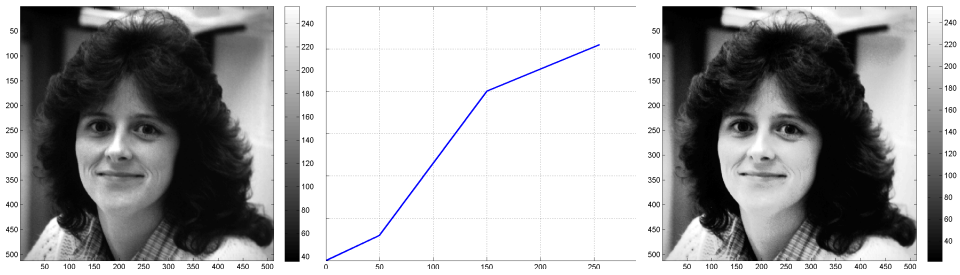


Figure 4: Original and contrast-stretched images.

Exercise 2. *Write the mathematical expression for the above piece-wise linear contrast stretching function.*

3.2 Bit-plane Slicing

1. Read and display image file.

```
Image = imread('pirate.tif.png');
Image = double(Image);
figure, imagesc(Image), axis image, colormap gray, colorbar;
```

2. Show help entry for modulus function 'mod'.

```
help mod
```

3. Iteratively shift intensity bits to the right by division and retrieve the bit value using the modulus of division by 2 to create bit-planes.

```
I0 = mod(Image, 2);
I1 = mod(floor(Image/2), 2);
```

```
I2 = mod(floor(Image/4), 2);
I3 = mod(floor(Image/8), 2);
I4 = mod(floor(Image/16), 2);
I5 = mod(floor(Image/32), 2);
I6 = mod(floor(Image/64), 2);
I7 = mod(floor(Image/128), 2);
```

4. Reconstruct the image using diferent combinations of bit-planes. Results from decomposition and reconstruction are displayed in Figure 5 and Figure 6.

```
CC1 = 2*I7;
figure, imagesc(CC1), axis image, colormap gray, colorbar;
CC2 = 2*(2*I7+I6);
figure, imagesc(CC2), axis image, colormap gray, colorbar;
CC3 = 2*(2*(2*I7+I6)+I5);
figure, imagesc(CC3), axis image, colormap gray, colorbar;
```

Exercise 3. *Repeat the bit-plane decomposition and reconstruction processes for the jetplane image. Describe the steps of this method, its implementation and the output.*

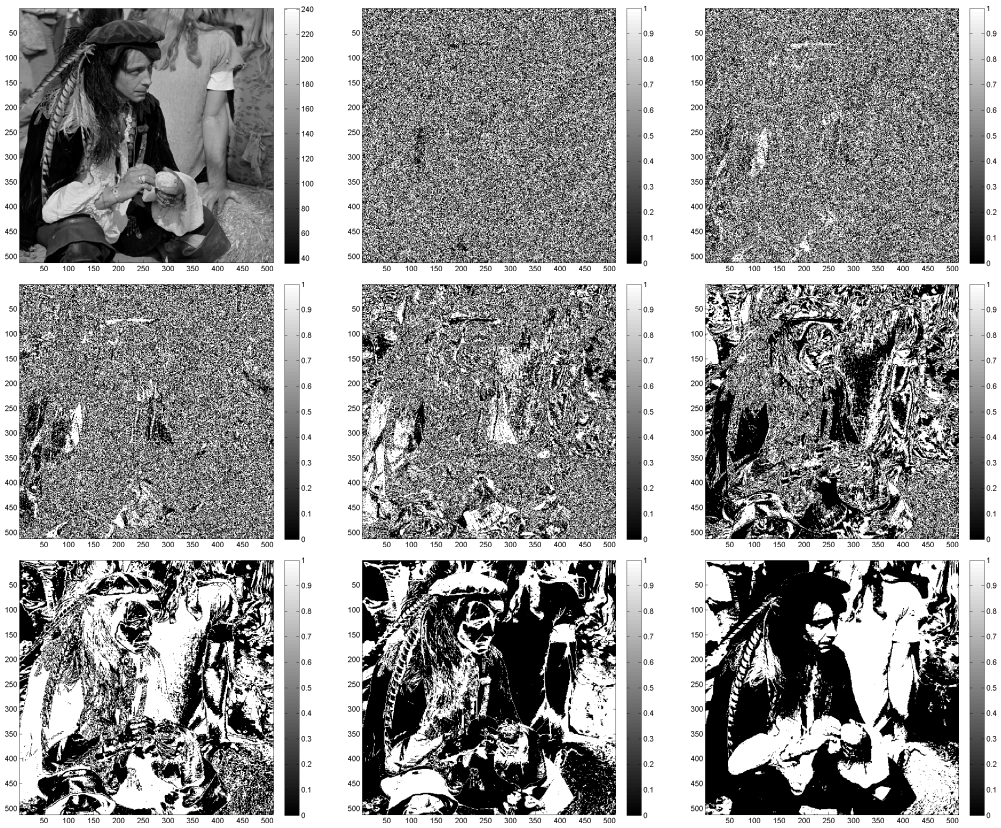


Figure 5: Original and bit-plane sliced images.

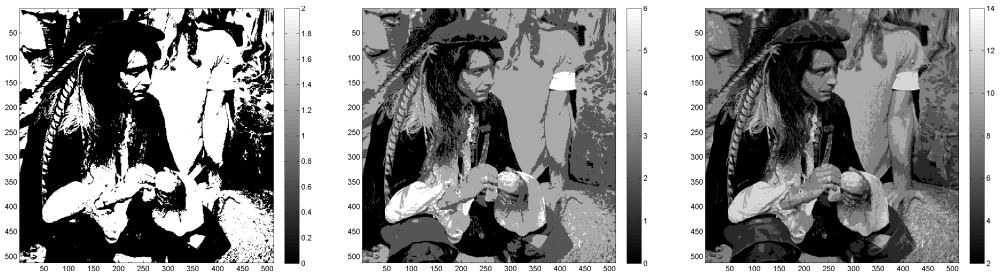


Figure 6: Reconstructed images using i) bit plane 8, ii) bitplanes 8 and 7, and iii) bitplanes 8,7 and 6 (left to right).

4 Histogram Processing

4.1 Histogram Calculation

1. Read and display input image.

```
Image = imread('peppers_gray.png');
figure, imagesc(Image), axis image, colormap gray, colorbar;
```

2. Compute and display histogram using default binning, or binning defined by vector. Results are displayed in Figure 7.

```
figure, hist(double(Image(:))), grid on;
xlabel('Intensity');
ylabel('N');
i=0:1:255;
figure, hist(double(Image(:)), i), grid on;
title('Histogram of peppers image')
xlabel('Intensity');
ylabel('N');
saveas(gcf, 'histogram_peppers_gray_original.png');
```

3. Display intensity histogram after dividing intensities by a constant.

```
figure, hist(double(Image(:)/5), i), grid on;
title('Histogram of peppers image intensities divided by 5.')
```

4. Display intensity histogram after applying the negative transform.

```
figure, hist(255-double(Image(:)), i), grid on;
title('Histogram of negative of peppers image.');
```

Exercise 4. *How do the arithemtic operations of division and inversion affect the intensity histogram? Explain why.*

4.2 Histogram Equalization

1. Read and display input image.

```
Image = imread('mandril_gray.png');
Image = double(Image);
figure, imagesc(Image), axis image, colormap gray, colorbar;
saveas(gcf, 'mandril_gray_original.png');
```

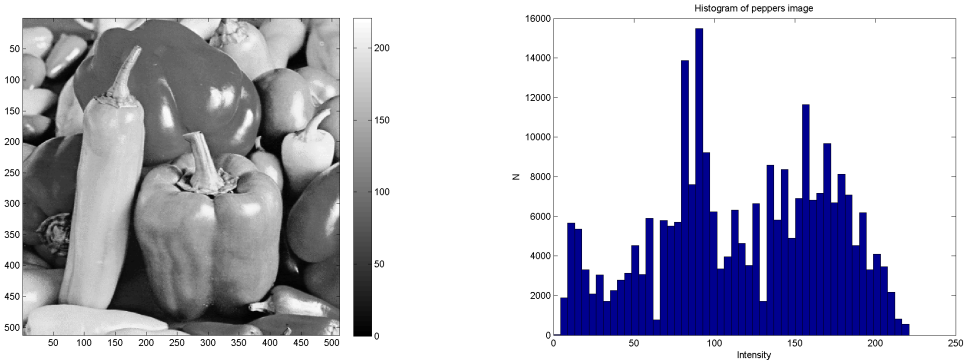


Figure 7: Original image and its intensity histogram.

2. Calculate histogram and pdf.

```
i = 0:1:255;
n = hist(Image(:), i);
pdf = n /sum(n);
figure, plot(i, pdf, 'linewidth', 2), axis tight, grid on,
    title('PDF');
saveas(gcf, 'mandril_gray_original_pdf.png');
```

3. Calculate the cdf and intensity mapping function.

```
cdf = cumsum(pdf);
figure, plot(i, cdf, 'linewidth', 2), axis tight, grid on,
    title('CDF');
saveas(gcf, 'mandril_gray_original_cdf.png');
mapping = 255 * cdf;
figure, plot(i, mapping, 'linewidth', 2), axis tight, grid
    on, title('Mapping');
saveas(gcf, 'mandril_gray_mapping.png');
```

4. Apply intensity mapping.

```
Image2 = zeros(size(Image));
for i=1:256, indices = find(Image == i-1); Image2(indices) =
    round(mapping(i));, end;
```

```
figure, imagesc(Image2), axis image, colormap gray, colorbar;
saveas(gcf, 'mandril_gray_equalized.png');
```

5. Calculate and display new histogram, pdf and cdf to verify the histogram equalization operation. Results are displayed in Figure 8.

```
i = 0:1:255;
n2 = hist(Image2(:), i);
pdf2 = n2 /sum(n2);
figure, plot(i, pdf2, 'linewidth', 2), axis tight, grid on,
    title('Equalized PDF');
saveas(gcf, 'mandril_gray_equalized_pdf.png');
cdf2 = cumsum(pdf2);
figure, plot(i, cdf2, 'linewidth', 2), axis tight, grid on,
    title('CDF');
saveas(gcf, 'mandril_gray_equalized_cdf.png');
```

Exercise 5. *Apply histogram equalization to the cameraman image. Observe and explain differences between the original and transformed histograms. Give a visual interpretation of this process. Explain why the pdf of output image may not be uniform for some images.*

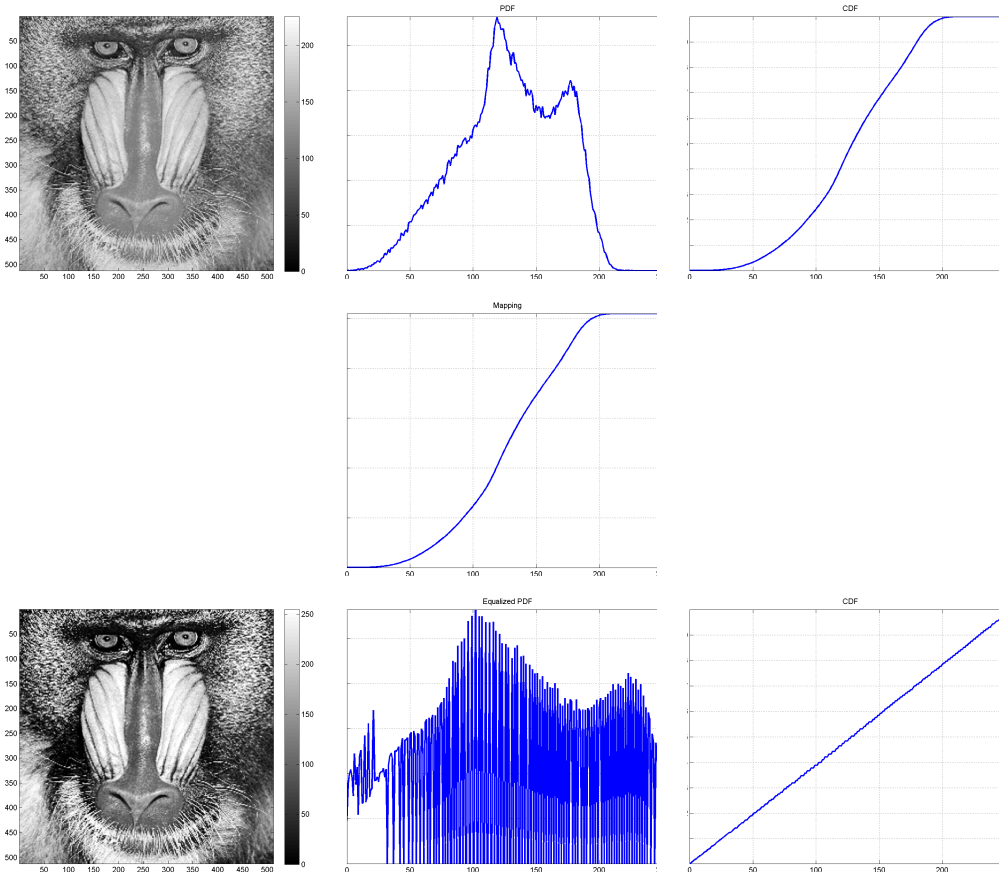


Figure 8: Histogram equalization process.

MTSC 887 - Image Processing
Fourier Transform
Lab Session

Sokratis Makrogiannis, Ph.D.

February 12, 2015

Contents

1 Spatial Filtering Fundamentals

1.1 1-D Convolution

Apply 1-D convolution between a noisy signal and an averaging filter to reduce noise.

1. Generate a ramp function and Gaussian noise.

2. Add noise to ramp and plot signals.

3. Construct an averaging filter.
4. Apply filtering by convolution.
5. Display all signals.

```
% Create data set.
Data = [1:0.2:10];

% Create Gaussian noise.
Noise = randn(1, 46);

% Add noise to data,
NoisyData = Data + Noise;

% Plot all signals.
figure, plot(Data), grid on, hold on
plot(NoisyData, 'k-'), grid on, hold on

% Create an averaging filter.
windowSize = 5;
h = (ones(1,windowSize)/windowSize)';

% Apply filtering by convolution.
% Out = filter(h,1,NoisyData);
Out = conv(NoisyData, h, 'same');
plot(Out, 'g'), legend('Original', 'Noisy', 'Filtered',
    'location', 'southeast');
```

Exercise 1. *What kind of noise did we just add? Plot the histogram to confirm. Identify the averaging mask.*

1.2 2-D Correlation Mechanics

Exercise 2. *The following function implements a correlation operation. Please explain code and suggest improvements.*

```
function FilteredImage = Correlation2D(Image, Mask)
% syntax: FilteredImage = Correlation2D(Image, Mask);
```

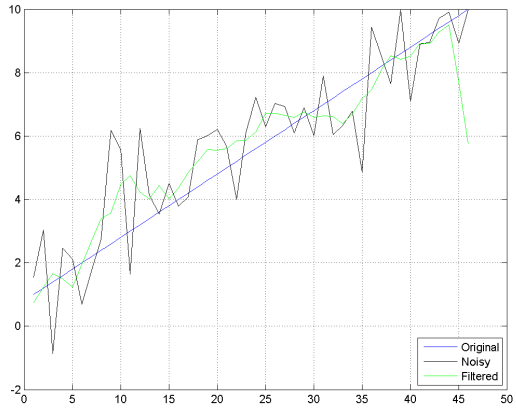


Figure 1: Example of convolution of a noisy signal with a box filter in 1-D for noise reduction.

```
% Input arguments.
% Image: input image.
% Mask: filter mask.
% Output arguments.
% FilteredImage: image after filtering.

% Get mask size.
MaskSize = size(Mask);
ImageSize = size(Image);

% Set the range of indices for filtering.
alpha = floor((MaskSize(1)-1)/2);
beta = floor((MaskSize(2)-1)/2);

% Allocate padded image.
PaddedImage = zeros(ImageSize + 2*[alpha, beta]);
Mask = double(Mask);

% Copy input image intensities to the padded image.
PaddedImage(1+alpha:ImageSize(1)+alpha, 1+beta:ImageSize(2)+beta)
    = Image;

% Allocate filtered image.
```

```
FilteredImage = zeros(ImageSize);

% Main loop for spatial correlation.
for i=1:ImageSize(1)
    for j=1:ImageSize(2)
        % Determine range for double sum.
        xIndices = i:i+2*alpha;
        yIndices = j:j+2*beta;
        % Multiply image pixels with filter mask and sum.
        FilteredImage(i,j) = sum( sum( PaddedImage( xIndices,
            yIndices ) .* Mask ) );
    end
end

end
```

Exercise 3. *Use above function to apply convolution of jetplane with an averaging filter.*

2 Spatial Filters

2.1 Building Filter Kernels: Gaussian, Laplacian, etc.

Exercise 4.

1. *Create a Gaussian kernel* $h(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$ *with* σ*: standard deviation and* $x, y \in \mathbb{Z}$.
2. *Select a test image from our data set.*
3. *Apply correlation using the custom function (Correlation2D) and Matlab's filter2.*
4. *Explain possible differences.*
5. *Repeat above steps to implement image sharpening by a Laplacian kernel* $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ *approximated by* $\nabla^2 f = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$.

2.2 Median Filtering

Exercise 5. *1. Propose modifications to the Correlation2D function, needed to implement median filtering.*

2. Implement the modifications and test on a noisy image.

3 Unsharp Masking

1. Blur the original image $f(x, y)$ to produce $\bar{f}(x, y)$.
2. Subtract blurred image from original (result is mask): $g_{mask}(x, y) = f(x, y) - \bar{f}(x, y)$.
3. Add mask to original image: $g(x, y) = f(x, y) + k \cdot g_{mask}(x, y)$.

- For $k = 1$, the above process is called *unsharp masking*.
- For $k > 1$, the above process is called *highboost filtering*.

```
A = imread('pirate.tif.png');
figure, imagesc(A), axis image, colormap gray, colorbar
H = fspecial('gaussian',3,1);
B = filter2(H,double(A));
figure, imagesc(B), axis image, colormap gray, colorbar
C = double(A) - B;
figure, imagesc(C), axis image, colormap gray, colorbar
D = double(A) + C;
figure, imagesc(D), axis image, colormap gray, colorbar
figure, imagesc(D, [0, 255]), axis image, colormap gray, colorbar
D = double(A) + 1.5 * C;
figure, imagesc(D, [0, 255]), axis image, colormap gray, colorbar
```

4 Correlation for Matching

Use correlation for template matching.

1. Read image and pattern.

Figure 2: Example of an original image, Gaussian smoothed, it's g-mask (top row, left to right), the result of unsharp masking, and the result of highboost filtering with $k = 1.5$ (bottom row, left to right).

2. Calculate correlation between image and pattern using i) matlab's pre-packaged function, and ii) custom function.

3. Identify peaks of correlation function by thresholding.

4. Overlay peaks onto image to confirm matching.

```
A = imread('pattern.pgm');
B = imread('image.pgm');
figure, imagesc(A), axis image, colormap gray, colorbar
figure, imagesc(B), axis image, colormap gray, colorbar
Out = filter2(A, B, 'same');
figure, imagesc(Out), axis image, colormap gray, colorbar
Out2 = Correlation2D(B, A);
figure, imagesc(Out2), axis image, colormap gray, colorbar
figure, imagesc(Out2>5*10e6), axis image, colormap gray, colorbar
figure, imshowpair(B, Out2>5*10e6), colorbar
```

Exercise 6. *Repeat the above process for a grayscale image and one part of it. Can we match a template with the image? If not, how would you improve*

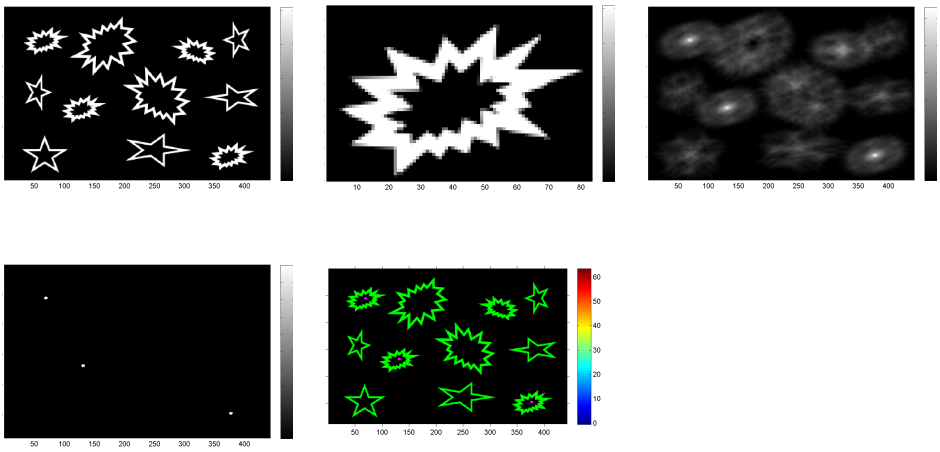


Figure 3: Example of an original image, and the pattern we are searching for, the result of correlation of image with pattern (top row, left to right), detection of peaks by thresholding and peaks overlaid on image (bottom row, left to right).

matching accuracy?

MTSC 887 - Image Processing
Fourier Transform
Lab Session

Sokratis Makrogiannis, Ph.D.

March 19, 2015

Contents

1 Discrete Fourier Transform

1.1 DFT and the Convolution Theorem in 1-D

This exercise shows the process of convolution and the convolution theorem in 1-D in the following steps.

1. Generate a periodic signal and a rectangular pulse.
2. Convolve signal with pulse and display all three signals.
3. Apply FFT to noisy signal and pulse.
4. Multiply the transformed signals.
5. Apply inverse FFT.

6. Compare the produced signals to confirm the convolution theorem.

```
f = randn(1,201);
t = 0:0.05:10;
f = sin(pi*t) .* sin(3*pi*t);
g = ones(1,15);
fgc = conv(f, g);
figure, plot(f)
figure, plot(g)
figure, plot(fgc)
F = fft(f, length(f)+length(g));
figure, plot(abs(fftshift(F)))
G = fft(g, length(f)+length(g));
figure, plot(abs(fftshift(G)))
FG = F.*G;
fgc2 = ifft(F.*G);
figure, plot(fgc2)
sum(abs(fgc(1:215)-fgc2(1:215)))
```

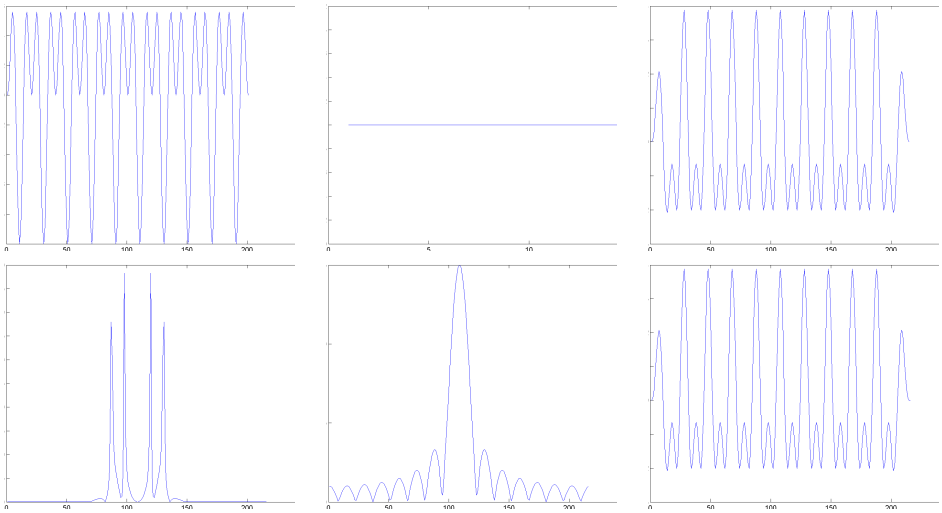


Figure 1: Example of a discrete function $f(t)$, a discrete function $g(t)$, the convolution $f * g$(top row, left to right), $F(\mu)$, $G(\mu)$, and $\mathcal{F}^{-}1(F \cdot G)$(bottom row, left to right).

Exercise 1. *What did you expect to observe based on the convolution theorem?*

1.2 Discrete Fourier Transform in 2-D

This exercise applies the forward and inverse Fast Fourier Transform (FFT), which is a fast implementation of Discrete Fourier Transform (DFT). We examine the magnitude and phase of the Fourier trasformed images after rotation.

1. Generate synthetic image with black and white squares.
2. Apply 2-D FFT followed by shifting.
3. Apply log transform to display magnitude.
4. Display phase.
5. Generate an image with rhombus.
6. Calculate 2-D FFT and display magnitude and phase.
7. Rotate rhombus and repeat previous step.
8. Observe differences.

```
a=[zeros(256,128) ones(256,128)];
figure, imagesc(a), axis image, colormap gray, colorbar
af=fftshift(fft2(a));
af1=log(1+abs(af));
figure, imagesc(log(1+abs(af))), axis image, colormap gray,
    colorbar
figure, imagesc(angle(af)), axis image, colormap gray, colorbar
[x,y]=meshgrid(1:256,1:256);

a=zeros(256,256);
a(78:178,78:178)=1;
figure, imagesc(a), colormap gray, colorbar
af=fftshift(fft2(a));
figure, imagesc(log(1+abs(af))), axis image, colormap gray,
    colorbar
```

```
figure, imagesc(angle(af)), axis image, colormap gray, colorbar

b=(x+y<329)&(x+y>182)&(x-y>-67)&(x-y<73);
figure, imagesc(b), colormap gray, colorbar
bf=fftshift(fft2(b));
figure, imagesc(log(1+abs(bf))), axis image, colormap gray,
    colorbar
figure, imagesc(angle(bf)), axis image, colormap gray, colorbar

c = imrotate(b, 30, 'nearest', 'crop');
figure, imagesc(c), axis image, colormap gray, colorbar
cf=fftshift(fft2(c));
figure, imagesc(log(1+abs(cf))), axis image, colormap gray,
    colorbar
figure, imagesc(angle(cf)), axis image, colormap gray, colorbar
```

Exercise 2. *Explain the steps in above process. Rotate square by 45 degrees and calculate 2-D FFT. Identify and explain the effect of rotation on the spectrum and phase.*

1.3 Sampling Theorem and Aliasing in 2-D

Here we simulate the aliasing effect via skipping rows and columns to downsample an image. We then upsample the image by pixel replication. Observe the aliasing effect on fine detail information.

```
A = imread('barbara.tif');
figure, imagesc(A), colormap gray, axis image, colorbar;
[row, col] = size(A);
% Downsampling by pixel replication.
X=1:4:row;
Y=1:4:col;
B = A(X,Y);
figure, imagesc(B), colormap gray, axis image, colorbar;
C = imresize(B, 4, 'nearest');
figure, imagesc(C), colormap gray, axis image, colorbar;

N = 9;
sigma = 1;
```

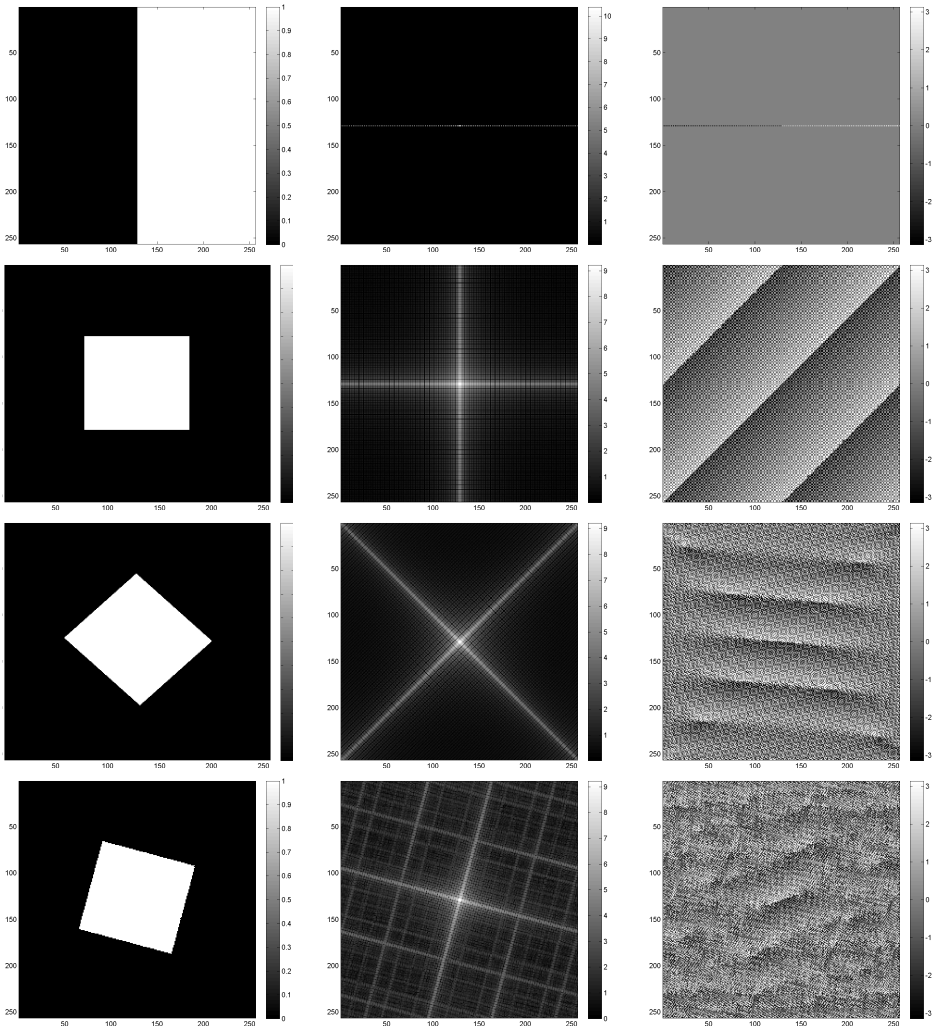


Figure 2: Examples of an original image, its DFT magnitude, and its DFT phase (left to right).

```
for x=1:N, for y=1:N,
    h(x,y)=(1/(2*pi*sigma^2))*exp((-1)*((x-(N+1)/2)^2+(y-(N+1)/2)^2)/(2*sigma^2));,
    end, end
A2 = Correlation2D(A, h);
B2 = A2(X,Y);
figure, imagesc(B2), colormap gray, axis image, colorbar;
C2 = imresize(B2, 4, 'nearest');
```

```
figure, imagesc(C2), colormap gray, axis image, colorbar;
```

Exercise 3. *Why does aliasing happen? How does it manifest itself? Create image using every eighth row and column and explain any differences in aliasing.*

1.4 Displaying an image as a surface

In this exercise we learn how to display an image as a surface.

```
A = imread('barbara.tif');
[row, col] = size(A);
figure, imagesc(A), colormap gray, axis image, colorbar;
X=1:row;
Y=1:col;
[YY,XX]=meshgrid(Y,X);
figure, mesh(XX,YY,double(A)), shading interp
```

Exercise 4 (DFT translation property)**.** *Show the translation property of DFT in the following steps.*

1. *Read a test image.*
2. *Apply 2-D FFT to original image.*
3. *Display spectrum using log transform.*
4. *Use the command fftshift to shift spectrum.*
5. *Display spectrum using log transform.*
6. *Multiply image by* $(-1)^{(x+y)}$.
7. *Apply 2-D FFT to the previous image.*
8. *Display second spectrum using log transform.*
9. *Compare the three spectra. What do you observe?*

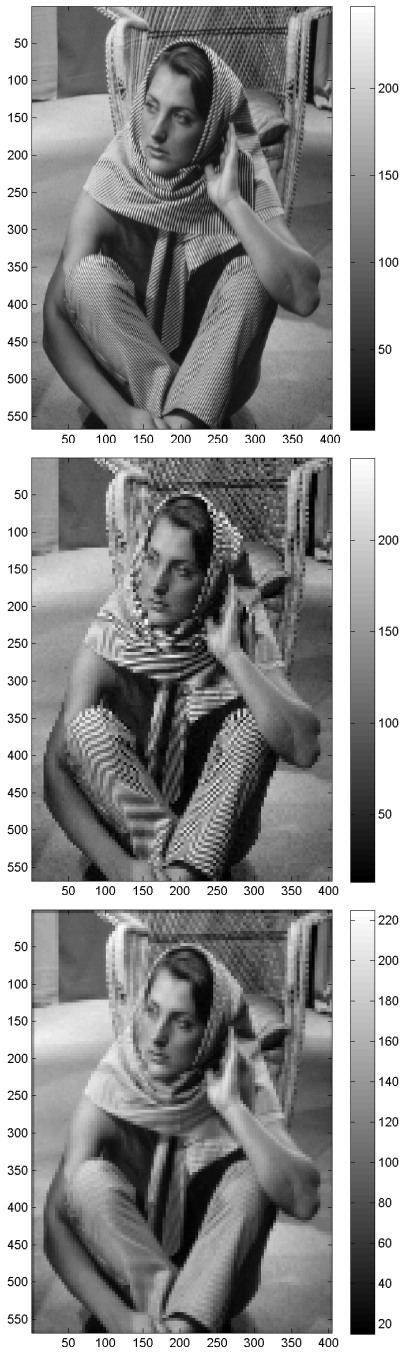


Figure 3: Original image, aliasing after downsampling by pixel replication, and reduction of aliasing by Gaussian blurring before downsampling. (left to right).

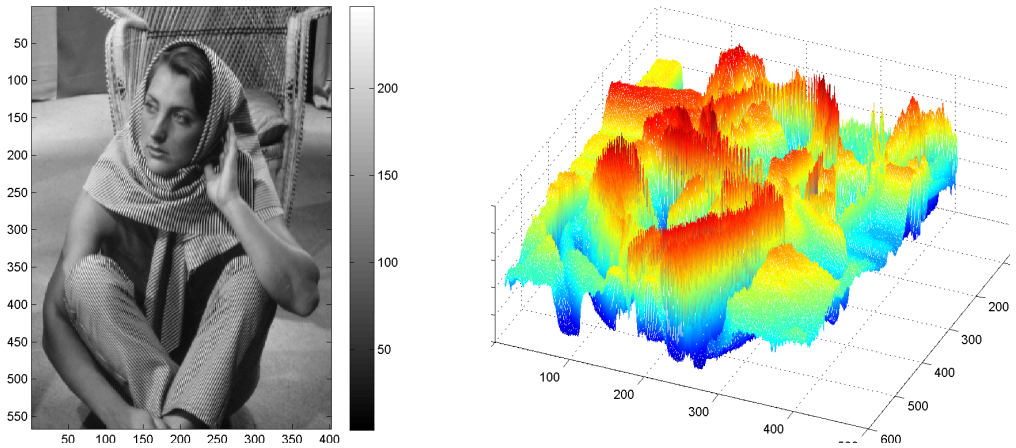


Figure 4: Original image, and image representation by a topographic surface (left to right).

Exercise 5 (DFT magnitude characteristics)**.** *The goal of this exercise is to interpret the visual content of an image in the frequency domain.*

Original image and edge image *Here we will observe differences between the spectra of an image and its gradient magnitude.*

1. *Read a test image.*
2. *Apply 2-D FFT to original image and display centered and log scaled spectrum.*
3. *Compute the gradient magnitude of the image* $\nabla f = \sqrt{{f_x}^2 + {f_y}^2}$ *(use the gradient command).*
4. *Apply 2-D FFT and display centered spectrum and log scaled spectrum of the gradient magnitude.*
5. *Compare the two spectra. What do you observe?*

Original image and padded image *We observe and comment on the effect of padding on an image spectrum.*

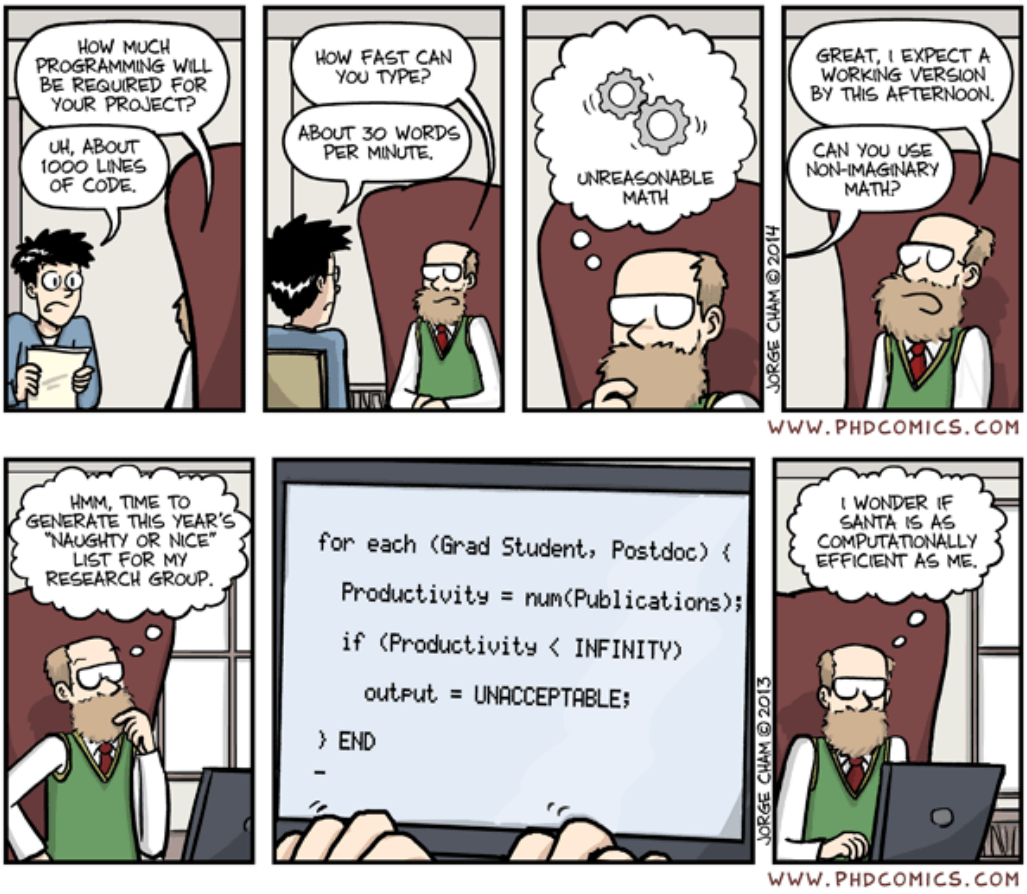
1. *Read a test image.*
2. *Apply 2-D FFT to original image and display centered and log scaled spectrum.*

3. *Pad image in the spatial domain (use the padarray command).*

4. *Apply 2-D FFT to padded image and display centered spectrum and log scaled spectrum.*

5. *Compare the two spectra. What do you observe?*

...and some project advice



MTSC 887 - Image Processing
Frequency Domain Filtering
Lab Session

Sokratis Makrogiannis, Ph.D.

March 27, 2015

Contents

1 Filtering in the Frequency Domain

1.1 The filtering process

This section deals with the filtering process and the steps to be followed.

1. Given image $f(x, y)$ with size $M \times N$, we append zeros to image matrix to increase size to $2M \times 2N$. Let padded image be $f_p(x, y)$.

2. Multiply $f_p(x, y)$ by $(-1)^{x+y}$ to shift DC frequency to the center.

3. Compute DFT: $F(u,v) = \mathcal{F}[f(x,y)]$.

4. Build a real and symmetric $2M \times 2N$ filter transfer function $H(u,v)$ in the frequency domain. Center function at (M,N).

5. Apply array multiplication: $F(u,v) \cdot H(u,v)$.

6. Transform back to spatial domain: $g_p(x,y) = Re\{\mathcal{F}^{-1}[(F(u,v) \cdot H(u,v)]\}(-1)^{x+y}$.

7. Extract the top left quadrant of $g_p(x,y)$ to remove padding. The result is the filtered image $g(x,y)$.

We apply the above process to perform Ideal Lowpass Filtering (ILPF) with $D_0 = 15$ on the test image $'lena\_gray\_256'$.

- ILPFs fully pass all frequencies within a radius D_0, and fully attenuate all frequencies outside of this circle.
- D_0 is called cut-off frequency.
- The filter transfer function is

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \leq D_0 \\ 0 & \text{if } D(u,v) > D_0. \end{cases}$$

  $D(u,v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2}$, D_0: cut-off frequency, (P,Q): padded image matrix size.

```
% Ideal lowpass filtering example.
A = imread('lena_gray_256.png');
figure, imagesc(A), axis image, colormap gray, colorbar;
B = zeros(2*size(A));
B(1:256, 1:256) = A;
figure, imagesc(B), axis image, colormap gray, colorbar;
FB = fftshift(fft2(B));
figure, imagesc(log(1+abs(FB))), axis image, colormap gray,
    colorbar;
[x,y]=meshgrid(-256:255,-256:255);
z=sqrt(x.^2+y.^2);
ILPF = (z < 15);
```

```
figure, imagesc(ILPF), axis image, colormap gray, colorbar;
FFB = FB .* ILPF;
figure, imagesc(log(1+abs(FFB))), axis image, colormap gray,
    colorbar;
IFFB = ifft2(ifftshift(FFB));
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
IFFB = real(ifft2(ifftshift(FFB)));
IFFB = IFFB(1:256,1:256);
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
```

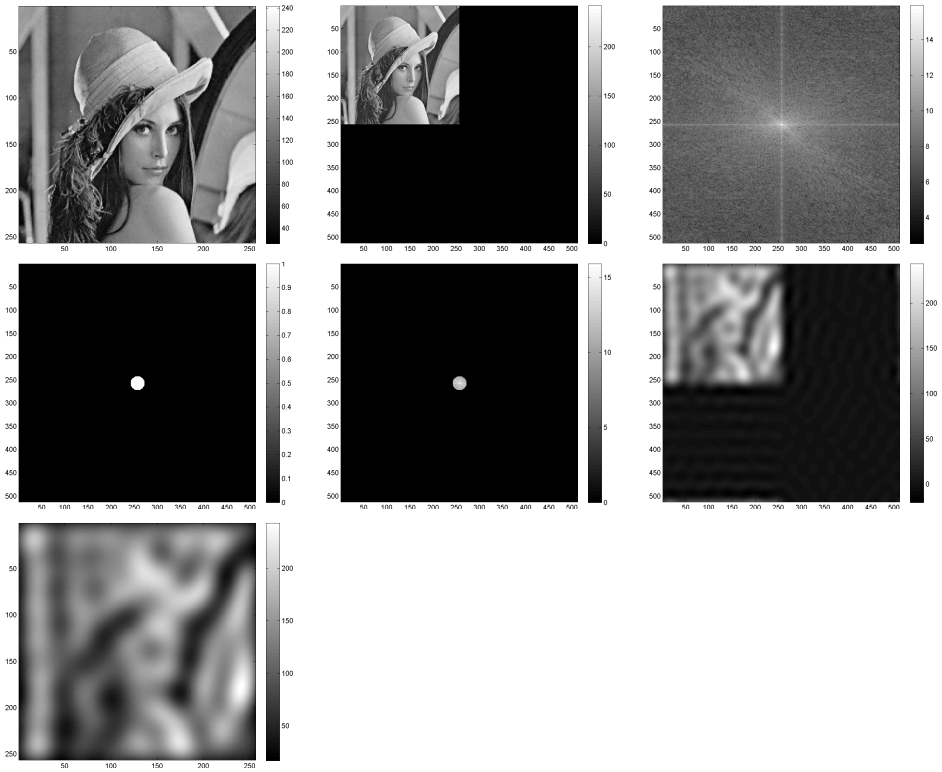


Figure 1: Ideal lowpass filtering process.

Exercise 1. *Try the following: in the above process skip the padding step and perform filtering. Observe any differences.*

1.2 Lowpass Filtering

- Sharp intensity changes in the image (for example noise, edges) corresond to high frequencies of the spectrum.
- Smoothing of such details corresponds to attenuation of the high frequency components.
- This process is known as lowpass filtering.
- Widely used lowpass filter types
  - Ideal Lowpass Filters
  - Butterworth Lowpass Filters
  - Gaussian Lowpass Filters

1.2.1 Ideal Lowpass Filter

Here we will experiment with different types of lowpass filters.

Exercise 2. *Please apply the following processes.*

1. *Open the test image house.*
2. *Generate ideal lowpass filters with radii 30, 50, 100.*
3. *Apply the filter and show results.*
4. *Make observations on the amount of blurring and presence of ringing artifacts.*

1.2.2 Gaussian Lowpass Filter

- The Gaussian lowpass filter transfer function is defined as

$$H(u,v) = \frac{1}{\sqrt{2\pi}D_0} e^{-D^2(u,v)/2D_0^2}.$$

- We can show that the IDFT of a Gaussian is also a Gaussian, implying that a Gaussian filter does not cause ringing.
- The following script applies Gaussian lowpass filtering.

```
% Gaussian lowpass filtering example.
A = imread('lena_gray_256.png');
figure, imagesc(A), axis image, colormap gray, colorbar;
B = zeros(2*size(A));
B(1:256, 1:256) = A;
figure, imagesc(B), axis image, colormap gray, colorbar;
FB = fftshift(fft2(B));
figure, imagesc(log(1+abs(FB))), axis image, colormap gray,
    colorbar;
[x,y]=meshgrid(-256:255,-256:255);
sigma = 50;
GLPF = (1/(sqrt(2*pi)*sigma)) * exp((-1)*(x.^2+y.^2)/(2*sigma^2));
figure, imagesc(GLPF), axis image, colormap gray, colorbar;
FFB = FB .* GLPF;
figure, imagesc(log(1+abs(FFB))), axis image, colormap gray,
    colorbar;
IFFB = real(ifft2(ifftshift(FFB)));
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
IFFB = IFFB(1:256,1:256);
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
```

1.2.3 Butterworth Lowpass Filter

- These filters can be seen as intermediate cases between the Ideal and Gaussian lowpass filters.
- The transfer function of a BLPF with order n is defined as

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}}.$$

```
% Butterworth lowpass filtering example.
A = imread('lena_gray_256.png');
figure, imagesc(A), axis image, colormap gray, colorbar;
B = zeros(2*size(A));
B(1:256, 1:256) = A;
figure, imagesc(B), axis image, colormap gray, colorbar;
FB = fftshift(fft2(B));
```

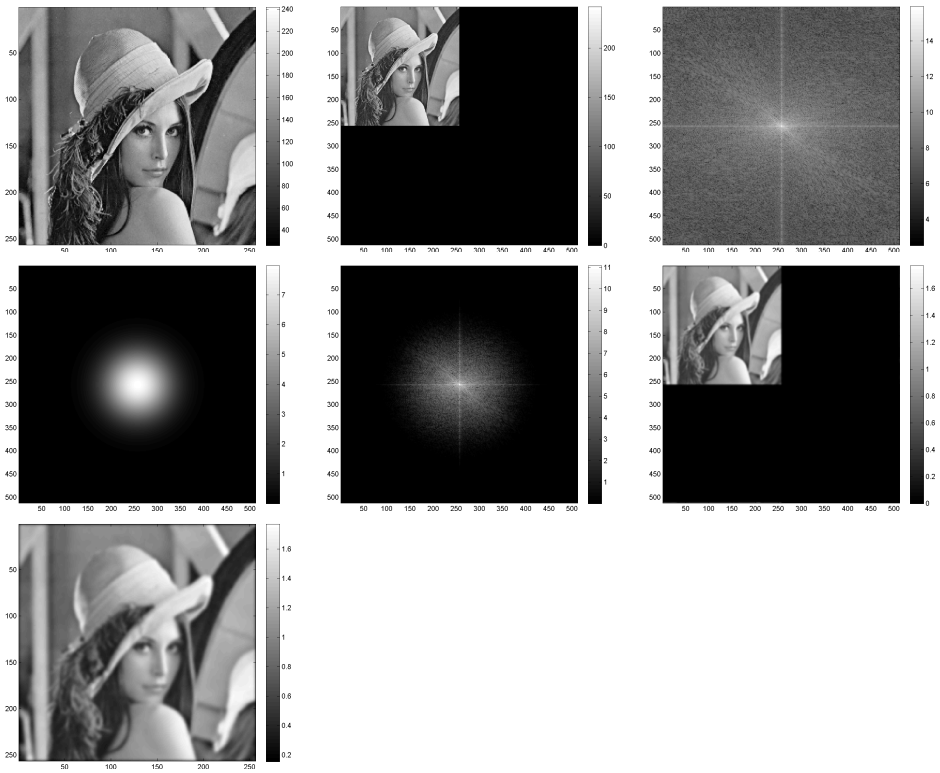


Figure 2: Gaussian lowpass filtering process.

```
figure, imagesc(log(1+abs(FB))), axis image, colormap gray,
    colorbar;
[x,y]=meshgrid(-256:255,-256:255);
n=1;
D0= 10;
BLPF=1./(1+(sqrt(x.^2+y.^2)/D0).^(2*n));
figure, imagesc(BLPF), axis image, colormap gray, colorbar;
FFB = FB .* BLPF;
figure, imagesc(log(1+abs(FFB))), axis image, colormap gray,
    colorbar;
IFFB = real(ifft2(ifftshift(FFB)));
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
IFFB = IFFB(1:256,1:256);
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
```

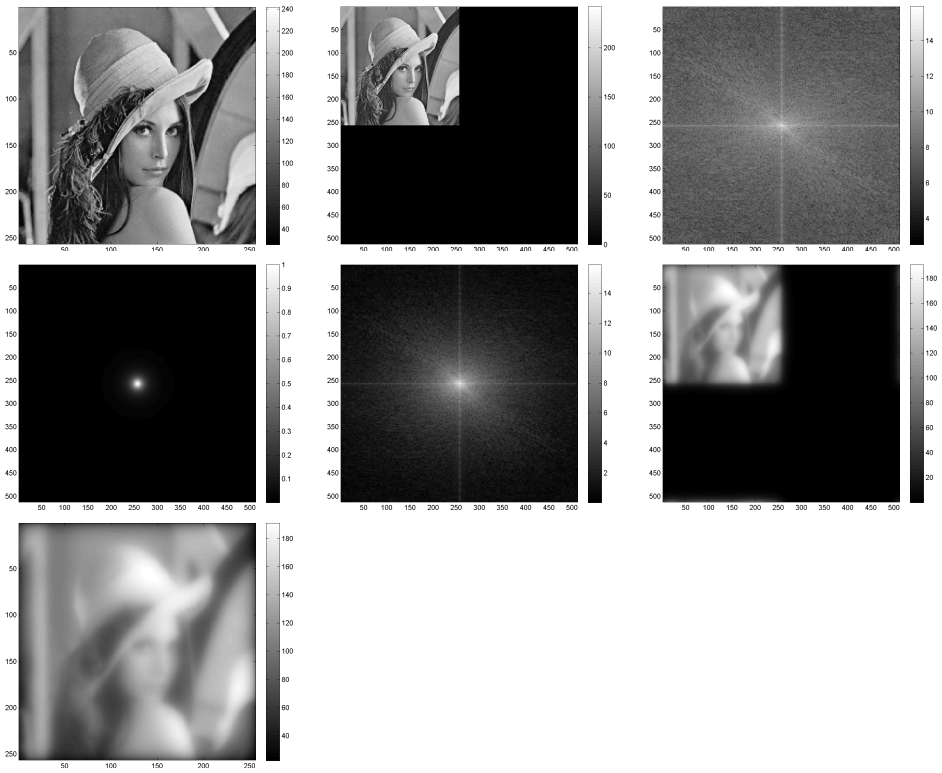


Figure 3: Butterworth lowpass filtering process.

Exercise 3. *Repeat above process for* $D_0 = 30, 50, 100$. *What do you observe?*

1.3 Highpass Filtering

Here we will experiment with different types of highpass filters.

- Fine image detail corresponds to high frequencies.
- Image sharpening can be achieved by highpass filtering.
- Highpass filtering attenuates low frequencies and retains high frequencies.
- We can simply obtain a highpass filter $H_{HP}(u, v)$ as the complementary of a lowpass filter $H_{LP}(u, v)$:

$$H_{HP}(u, v) = 1 - H_{LP}(u, v).$$

- We can design highpass filters in the frequency domain such as Ideal highpass, Gaussian highpass, Butterworth highpass, Laplacian highpass, Unsharp masking, and homomorphic filters.

1.3.1 Ideal Highpass Filter

```
% Ideal highpass filtering example.
A = imread('lena_gray_256.png');
figure, imagesc(A), axis image, colormap gray, colorbar;
B = zeros(2*size(A));
B(1:256, 1:256) = A;
figure, imagesc(B), axis image, colormap gray, colorbar;
FB = fftshift(fft2(B));
figure, imagesc(log(1+abs(FB))), axis image, colormap gray,
    colorbar;
[x,y]=meshgrid(-256:255,-256:255);
z=sqrt(x.^2+y.^2);
ILPF = (z >= 15);
figure, imagesc(ILPF), axis image, colormap gray, colorbar;
FFB = FB .* ILPF;
```

```
figure, imagesc(log(1+abs(FFB))), axis image, colormap gray,
    colorbar;
IFFB = ifft2(ifftshift(FFB));
figure, imagesc(IFFB, [0, 255]), axis image, colormap gray,
    colorbar;
IFFB = real(ifft2(ifftshift(FFB)));
IFFB = IFFB(1:256,1:256);
figure, imagesc(IFFB, [0, 255]), axis image, colormap gray,
    colorbar;
```

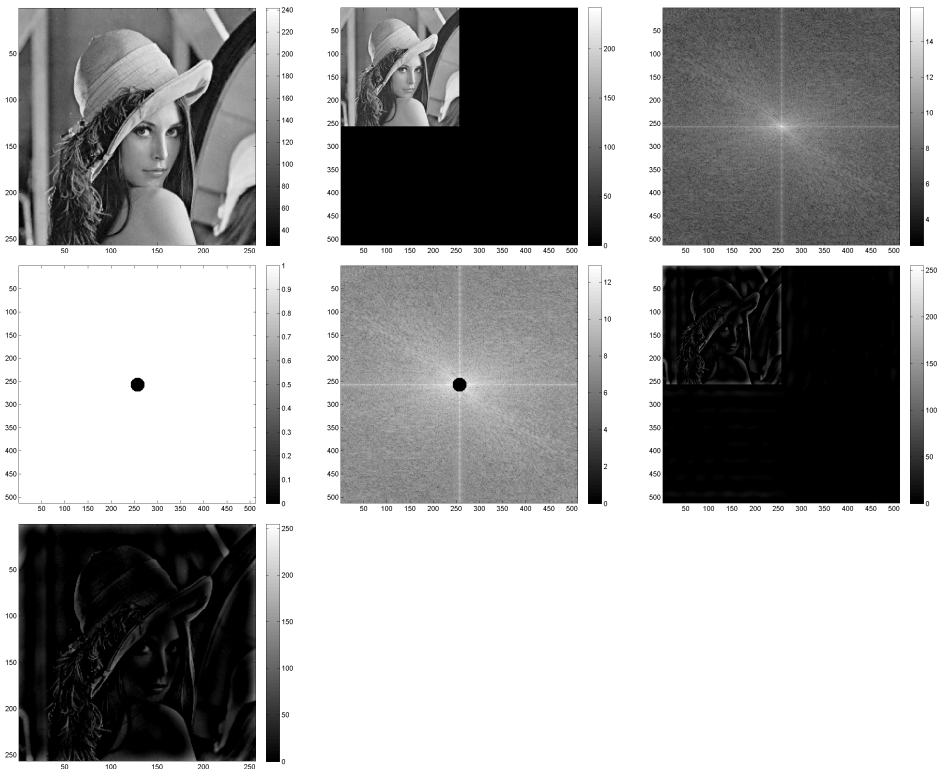


Figure 4: Ideal highpass filtering process.

1.4 Bandpass Filtering

- Frequently, we are interested in filters that attenuate or pass a specific range of frequencies.

- Bandreject and bandpass filters attenuate or pass specific bands of frequencies.
- We can design bandpass filters based on previous filter definitions.
- Notch filters pass or attenuate small regions of the frequency rectangle.

1.4.1 Gaussian Bandpass Filter

- Gaussian bandpass filter:

$$H(u,v) = e^{-[D(u,v)^2 - D_0^2]^2/(DW)^2}.$$

- $D(u,v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2}$, D_0: radial center of the band, W: width of band (or bandwidth).

```
% Gaussian bandpass filtering example.
A = imread('lena_gray_256.png');
figure, imagesc(A), axis image, colormap gray, colorbar;
B = zeros(2*size(A));
B(1:256, 1:256) = A;
figure, imagesc(B), axis image, colormap gray, colorbar;
FB = fftshift(fft2(B));
figure, imagesc(log(1+abs(FB))), axis image, colormap gray,
    colorbar;
[x,y]=meshgrid(-256:255,-256:255);
D = sqrt(x.^2+y.^2);
D0 = 100;
W = 50;
GBPF = (1/(sqrt(2*pi)*D0)) * exp((-1)*(D.^2 - D0^2).^2 ./ (D.^2 *
    W^2));
figure, imagesc(GBPF), axis image, colormap gray, colorbar;
FFB = FB .* GBPF;
figure, imagesc(log(1+abs(FFB))), axis image, colormap gray,
    colorbar;
IFFB = real(ifft2(ifftshift(FFB)));
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
IFFB = IFFB(1:256,1:256);
figure, imagesc(IFFB), axis image, colormap gray, colorbar;
```

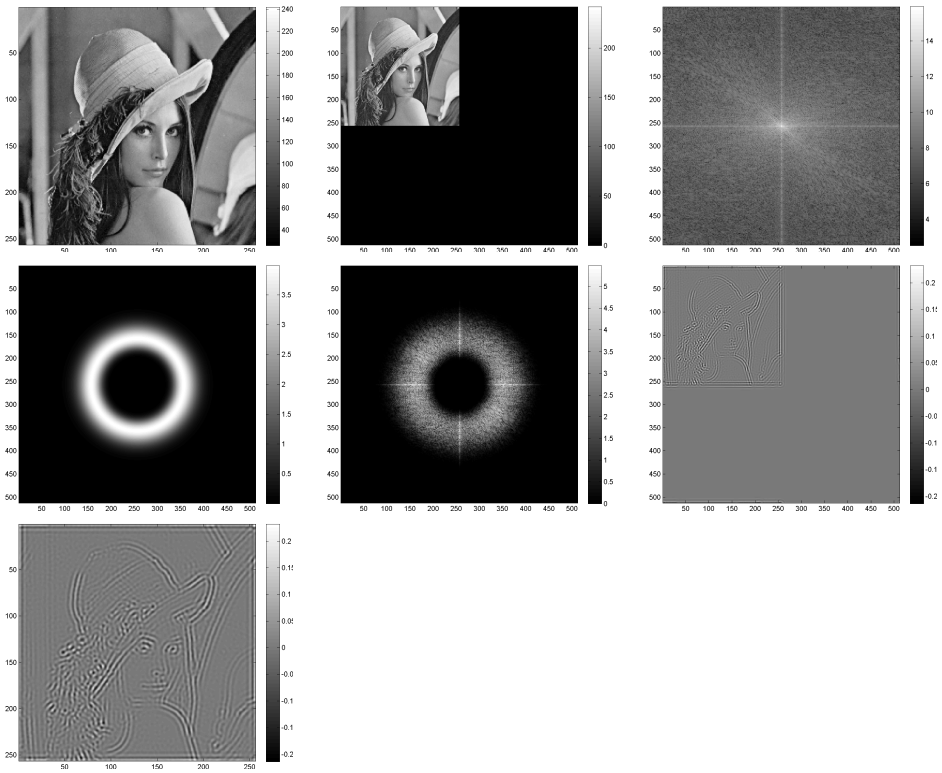


Figure 5: Gausian bandpass filtering process.

Exercise 4. *Apply the above bandpass filter for different center frequencies and bandwidths to extract different characteristics of the image.*

MTSC 887 - Image Processing
Image Restoration and Reconstruction
Lab Session

Sokratis Makrogiannis, Ph.D.

April 1, 2015

Contents

1 The Degradation and Restoration Process

- Here we assume that an image $f(x, y)$ undergoes a degradation process modeled by function $h(x, y)$ followed by corruption by additive noise $n(x, y)$.

$$g(x, y) = h(x, y) * f(x, y) + n(x, y).$$

- In the frequency domain this becomes

$$G(u,v) = H(u,v)F(u,v) + N(u,v).$$

2 Restoration in the Presence of Noise

2.1 Noise Models

- Most types of noise are modeled by a probability density function.
- The noise models are typically chosen based on some understanding of the noise source.

2.1.1 Gaussian Noise

- The pdf is

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(z-\bar{z})^2}{2\sigma^2}}.$$

- Gaussian noise is caused by
  - electronic circuit noise
  - sensor noise due to poor illumination and/or high temperature.

Adding Gaussian noise to image

1. Read test image.
2. Calculate and display histogram.
3. Add Gaussian noise to image.
4. Calculate and display histogram.
5. Compare the two histograms.

```
% Gaussian noise example.
A = imread('peppers_gray.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
figure, hist(double(A(:)), 50);
B = imnoise(A, 'gaussian', 0, 0.01);
```

```
figure, imagesc(B), colormap gray, colorbar, axis image;
figure, hist(double(B(:)), 50);
```

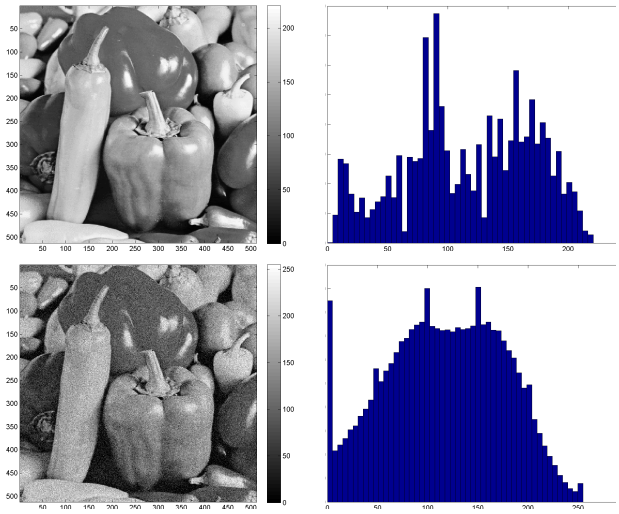


Figure 1: Corruption by Gaussian noise.

2.1.2 Salt and Pepper Noise

- The pdf is given by

$$p(z) = \begin{cases} P_a & z = a \\ P_b & z = b \end{cases}$$

- Common when quick transients (eg, faulty switching) takes places during imaging.

Adding Salt and Pepper noise to image

1. Create Salt and Pepper noise.
2. Read test image.
3. Add noise to image.
4. Calculate and display histogram.

```
% Salt and pepper noise example.
A = imread('peppers_gray.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
figure, hist(double(A(:)), 50)
B = imnoise(A, 'salt & pepper', 0.05);
figure, imagesc(B), colormap gray, colorbar, axis image;
figure, hist(double(B(:)), 50);
```

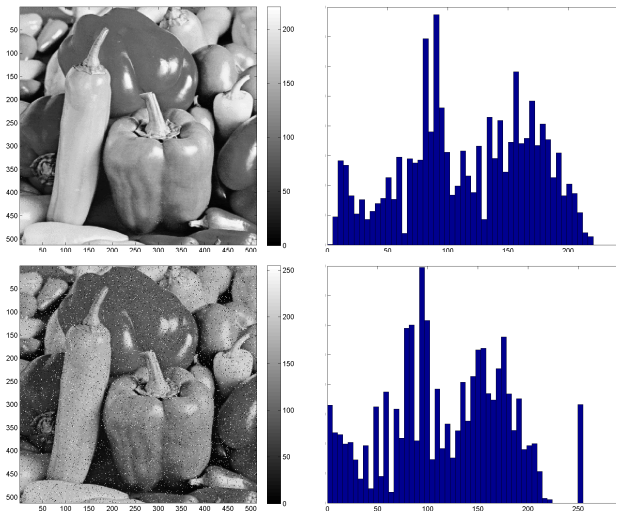


Figure 2: Corruption by Salt and Pepper noise.

Exercise 1. *Generate uniform noise and calculate and display histogram. Calculate noise average and standard deviation using a patch for each case.*

2.2 Denoising

- If noise can not be estimated, filtering methods are used to suppress noise
- The main types of filters for denoising are
  - Mean filters (arithmetic, geometric, harmonic, etc)
  - Order statistics filters (median, min, max)
  - Frequency-domain filters.

2.2.1 Reducing Gaussian Noise

1. Build a 3×3 averaging filter.
2. Apply filtering to noisy image.
3. Display filtering output.
4. Calculate and display histograms of original, noisy, and restored image.

```
% Gaussian denoising example.
A = imread('peppers_gray.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
B = imnoise(A, 'gaussian', 0, 0.01);
figure, imagesc(B), colormap gray, colorbar, axis image;
a3=fspecial('average');
C = filter2(a3, B);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, hist(double(A(:)), 50)
figure, hist(double(B(:)), 50)
figure, hist(double(C(:)), 50)
```

Exercise 2. *Repeat averaging with filters of kernels* 5×5 *and* 9×9 *and display restored images and histograms. Make comments on results. Compute SNR between the original and restored images for all cases.*

2.2.2 Reducing Salt and Pepper Noise

Use averaging filter

1. Build a 3×3 averaging filter.
2. Apply filtering to noisy image.
3. Display filtering output.
4. Calculate and display histograms of original, noisy, and restored image.
5. Increase kernel to 7×7 and repeat above process.

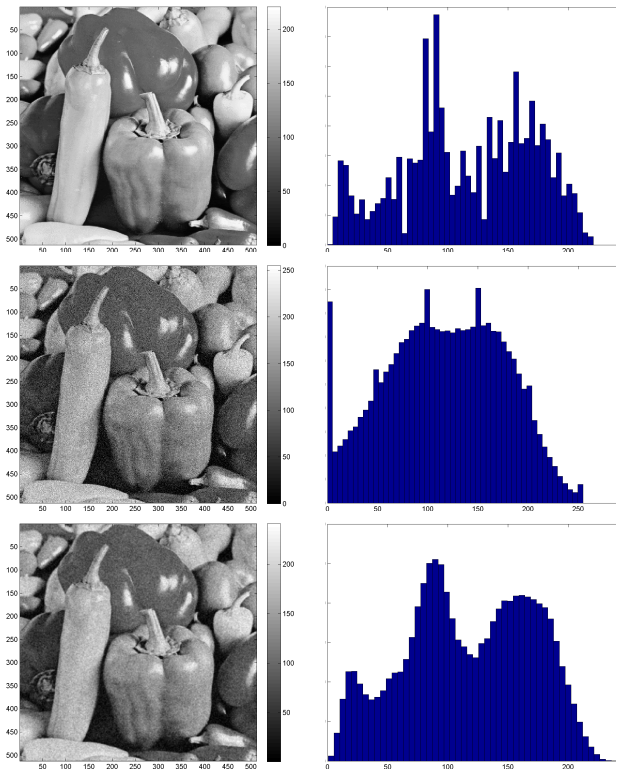


Figure 3: Corruption by Gaussian noise and denoising by averaging.

```
% Salt and pepper denoising.
A = imread('peppers_gray.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
B = imnoise(A, 'salt & pepper', 0.2);
figure, imagesc(B), colormap gray, colorbar, axis image;
a3=fspecial('average', [3 3]);
C = filter2(a3, B);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, hist(double(A(:)), 50)
figure, hist(double(B(:)), 50)
figure, hist(double(C(:)), 50)
a7=fspecial('average', [7 7]);
D = filter2(a7, B);
figure, imagesc(D), colormap gray, colorbar, axis image;
figure, hist(double(D(:)), 50);
```

Use median filter

1. Build a median filter.
2. Apply filtering to noisy image.
3. Display filtering output.
4. Calculate and display histograms of original, noisy, and restored image.
5. Increase kernel to 5×5 and repeat above process.

```
% Salt and pepper denoising.
A = imread('peppers_gray.png');
B = imnoise(A, 'salt & pepper', 0.2);
figure, imagesc(B), colormap gray, colorbar, axis image;
C = medfilt2(B, [3 3]);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, hist(double(A(:)), 50)
figure, hist(double(B(:)), 50)
figure, hist(double(C(:)), 50)
C = medfilt2(B, [5 5]);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, hist(double(C(:)), 50);
```

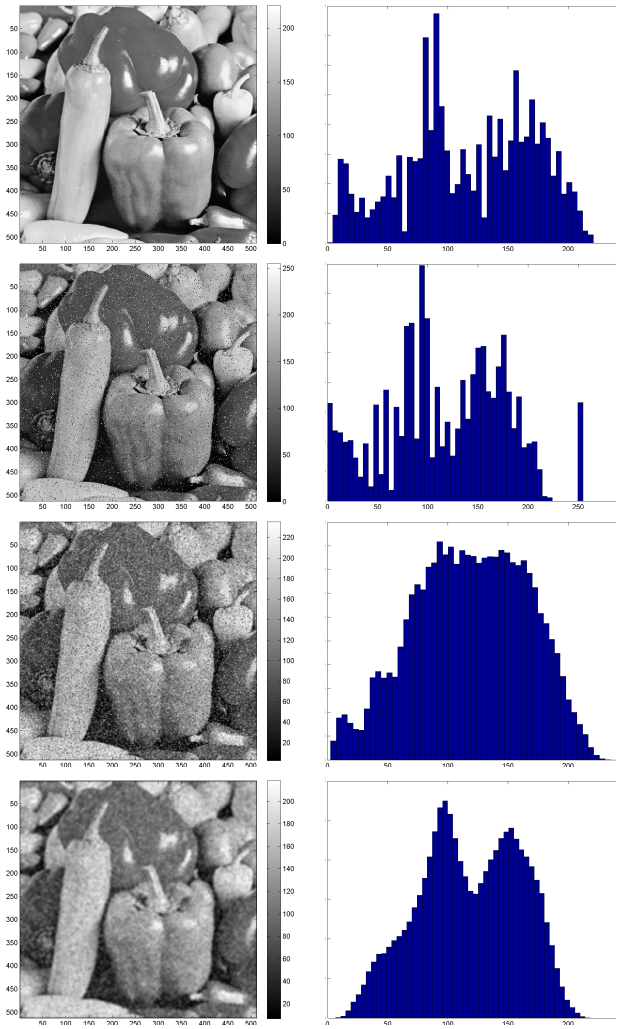


Figure 4: Corruption by salt and pepper noise and denoising by averaging filter.

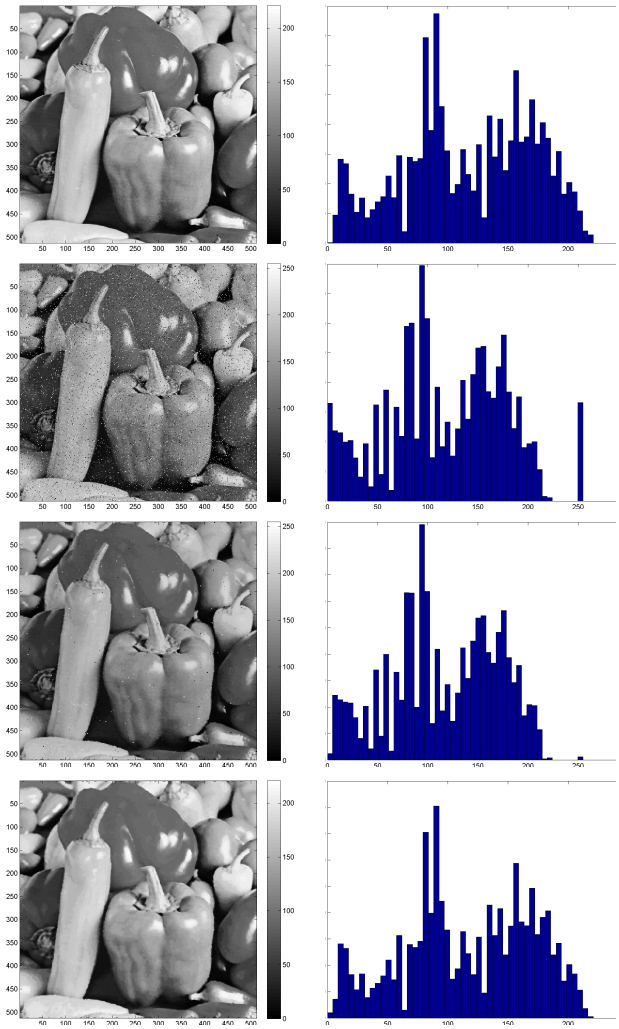


Figure 5: Corruption by salt and pepper noise and denoising by median filter.

Exercise 3. *Test other order statistics, and midpoint filters. Compute SNR between the original and restored images.*

3 Restoration in the Presence of Degradation

- We assume the following degradation and noise corruption scenario

$$G(u,v) = H(u,v)F(u,v) + N(u,v).$$

- H can be modeled mathematically. Two cases are atmospheric turbulence and motion.

3.1 Creating Degradation

- The atmospheric turbulence can be modeled as

$$H(u,v) = e^{-k(u^2+v^2)^{5/6}}.$$

1. Create atmospheric turbulence type of degradation. Use function $H(u,v) = e^{-k(u^2+v^2)^{5/6}}$
2. Apply degradation to a test image.

```
% Modeling atmospheric turbulence.
A = imread('37073.jpg');
A = rgb2gray(A);
figure, imagesc(A), colormap gray, colorbar, axis image;
mid = floor(size(A)/2);
[y, x]=meshgrid(-mid(2):mid(2),-mid(1):mid(1));
k = 0.0025;
H = exp((-1)*k*(y.^2+x.^2).^(5/6));
figure, imagesc(log(1+abs(H))), axis image, colormap gray,
    colorbar;
FA = fftshift(fft2(double(A)));
figure, imagesc(log(1+abs(FA))), axis image, colormap gray,
    colorbar;
HFA = H .* FA;
figure, imagesc(log(1+abs(HFA))), axis image, colormap gray,
    colorbar;
```

```
IHFA = real(ifft2(ifftshift(HFA)));
figure, imagesc(IHFA), colormap gray, colorbar, axis image;
```

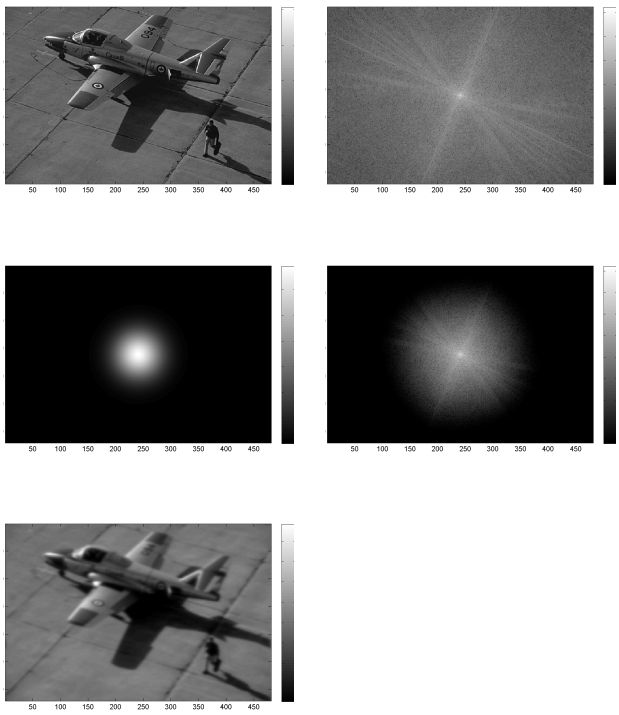


Figure 6: Simulating atmospheric turbulence.

Exercise 4. *Repeat previous process for smaller values of K. Restore image using inverse filtering and regularization methods. Compute SNR between the original and restored images.*

MTSC 887 - Image Processing
Morphological Image Processing
Lab Session

Sokratis Makrogiannis, Ph.D.

April 16, 2015

Contents

1 Morphological Image Processing

This session deals with mathematical morphology operators and their use for image processing. Morphological image processing deals with the extraction and analysis of shape patterns in digital imaging.

1.1 Erosion

- Let A and B be sets of pixels in $\mathbb{Z}^2$.

- Then the erosion of A by B written $A \ominus B$ is defined as

$$A \ominus B = \{w|(B)_w \subseteq A\}.$$

- To perform erosion we can move B over A and find all the locations it will fit. The set of all such locations forms $A \ominus B$.

Morphological erosion process

1. Read test image.
2. Create a square 3×3 structuring element.
3. Erode image by element.
4. Display original and eroded images.
5. Repeat erosion process using a cross-shaped 3×3 structuring element.
6. Display difference between the two eroded images.

```
% Erosion example.
A = imread('circles.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
se = ones(3,3);
B = imerode(A, se);
figure, imagesc(B), colormap gray, colorbar, axis image;
se2 = [0 1 0; 1 1 1; 0 1 0];
C = imerode(A, se2);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, imagesc((B-C)), colormap gray, colorbar, axis image;
```

1.2 Dilation

- Let A and B be sets of pixels in $\mathbb{Z}^2$.
- Then the dilation of A by B written $A \oplus B$ is defined as

$$A \oplus B = \{(x, y) + (u, v)|(x, y) \in A, (u, v) \in B\}.$$

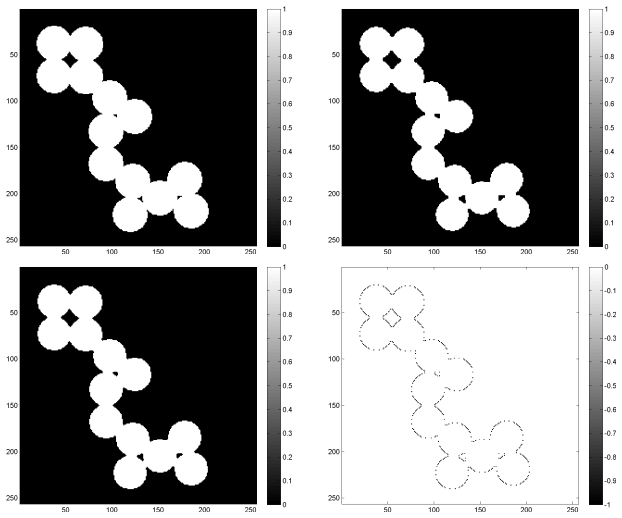


Figure 1: Erosion example.

- To perform dilation we can move B over A and replace every point $(x, y) \in A$ with a copy of B centered at (x, y). The set of all such locations forms $A \oplus B$.

Morphological dilation process

1. Read test image.
2. Create a square 3×3 structuring element.
3. Dilate image by element.
4. Display original and dilated images.
5. Repeat dilation process using a cross-shaped 3×3 structuring element.
6. Display difference between the two dilated images.

```
% Dilation example.
A = imread('circles.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
se = ones(3,3);
B = imdilate(A, se);
figure, imagesc(B), colormap gray, colorbar, axis image;
```

```
se2 = [0 1 0; 1 1 1; 0 1 0];
C = imdilate(A, se2);
figure, imagesc(C), colormap gray, colorbar, axis image;
figure, imagesc((B-C)), colormap gray, colorbar, axis image;
```

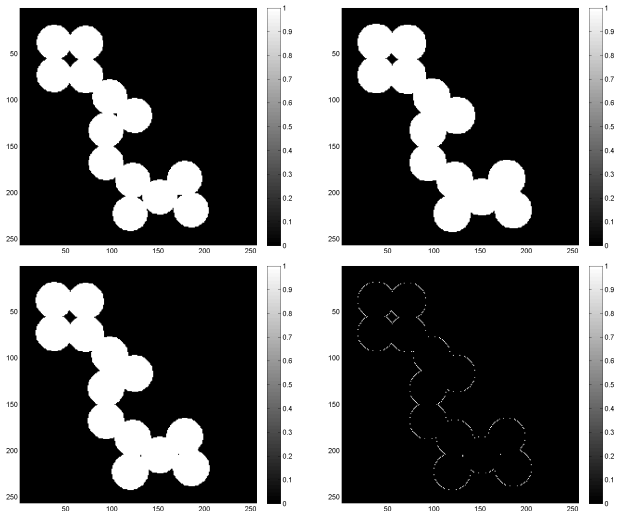


Figure 2: Dilation example.

Exercise 1. *Use matlab's strel command to create structuring elements of various shapes and sizes. Then use these elements to apply dilation. Observe and explain differences.*

1.3 Duality between Erosion and Dilation

- Erosion and dilation are duals of each other with respect to set complement and reflection:

  $$(A \ominus B)^c = A^c \oplus \hat{B}$$

  and

  $$(A \oplus B)^c = A^c \ominus \hat{B}.$$

- This is particularly useful when the structuring element is symmetric with respect to its origin.

Exercise 2. *Write a script that will verify that erosion and dilation are duals.*

1.4 Opening

- Opening of a set A by a structuring element B is symbolized by $A \circ B$ and given by
$$A \circ B = (A \ominus B) \oplus B.$$
- So A is first eroded by B and the result is dilated by B.

1.5 Closing

- Closing of a set A by a structuring element B is symbolized by $A \bullet B$ and given by
$$A \bullet B = (A \oplus B) \ominus B.$$
- So A is first dilated by B and the result is eroded by B.

Morphological closing process

1. Read test image.
2. Create a square 3×3 structuring element.
3. Dilate image by structuring element.
4. Erode last output by structuring element to complete closing operation.
5. Display original and closed images.
6. Repeat closing process using a cross-shaped 3×3 structuring element.
7. Display difference between the two closed images.

```
% Closing example.
A = imread('text.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
se = ones(3,3);
B = imdilate(A, se);
C = imerode(B, se);
figure, imagesc(C), colormap gray, colorbar, axis image;
se2 = [0 1 0; 1 1 1; 0 1 0];
D = imdilate(A, se2);
E = imerode(D, se2);
```

```
figure, imagesc(E), colormap gray, colorbar, axis image;
figure, imagesc((C-E)), colormap gray, colorbar, axis image;
```

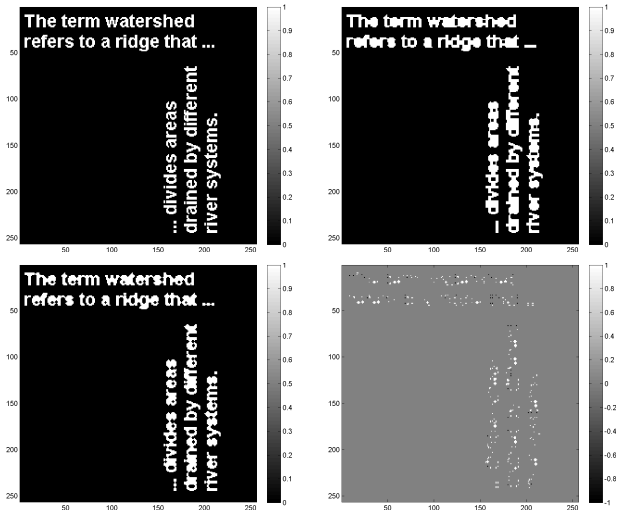


Figure 3: Closing example.

1.6 Duality between Opening and Closing

- The duality between opening and closing can be expressed by

$$(A \bullet B)^c = (A^c \circ \hat{B})$$

and

$$(A \circ B)^c = (A^c \bullet \hat{B})$$

Duality between opening and closing experiment

1. Read test image.
2. Create a square 3×3 structuring element.
3. Apply closing to original image by structuring element and calculate complement of closing output.

4. Calculate complement of original image and open complement by structuring element.

5. Display results from steps 3 and 4.

6. Display difference between the images produced in steps 3 and 4.

```
% Opening-closing duality example.
A = imread('text.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
se = ones(3,3);
B = imdilate(A, se);
C = imerode(B, se);
D = ~C;
figure, imagesc(D), colormap gray, colorbar, axis image;
E = imerode(~A, se);
F = imdilate(E, se);
figure, imagesc(F), colormap gray, colorbar, axis image;
figure, imagesc((D-F)), colormap gray, colorbar, axis image;
```

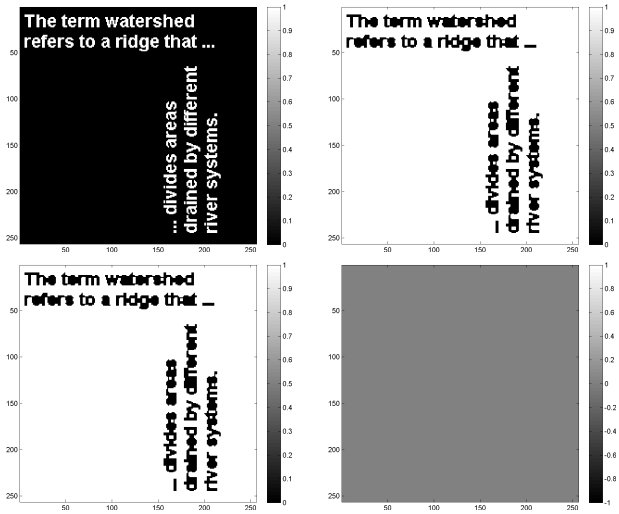


Figure 4: Example of duality between closing and opening.

1.7 Opening Properties

Opening satisfies the following properties

1. $A \circ B$ is a subset (subimage) of A.
2. If C is a subset of D, then $C \circ B$ is a subset of $D \circ B$.
3. $(A \circ B) \circ B = A \circ B$. After the first opening, any additional opening operations have no additional effect.

Exercise 3. *Write a script that verifies the above properties of opening operation.*

1.8 Hit-or-Miss Transform

- This method is used to detect shapes.
- Let a set A consisting of sets C, D, and E, such that $A = (C \cup D \cup E)$.
- Let B denote the set containing D and its background W.
- Now we assume that we are looking to find the location of D.
- The location of D is given by the intersection of the erosion of A by D with the erosion of A^c by $W - D$. This is expressed as

$$A \circledast B = (A \ominus D) \cap (A^c \ominus (W - D)).$$

- If we let $B_1 = D$ and $B_2 = W - D$, it follows that

$$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2).$$

- By utilizing duality between erosion and dilation we get

$$A \circledast B = (A \ominus B_1) - (A \oplus \hat{B}_2).$$

Hit-or-miss transformation process

1. Read test image.
2. Select the pattern to be matched.

3. Erode image with foreground.
4. Erode complementary of image with background.
5. Find set intersection from previous two steps.
6. Convolve the location of match with the pattern and translate origin.
7. Display matched pattern on image.

```
% Hit-or-miss example.
A = imread('text.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
B = A(12:24,93:105);
figure, imagesc(B), colormap gray, colorbar, axis image;
TB1 = imerode(A, B);
figure, imagesc(TB1), colormap gray, colorbar, axis image;
B2 = ones(size(B)) & ~B;
TB2 = imerode(~A,B2);
figure, imagesc(TB2), colormap gray, colorbar, axis image;
HORM = TB1 & TB2;
figure, imagesc(HORM), colormap gray, colorbar, axis image;
VF = conv2(double(B), double(HORM));
for i=1:size(VF,1), for j=1:size(VF,2),
i2 = i - floor(size(B,1)/2);
j2 = j - floor(size(B,2)/2);
if i2>0 & i2<size(VF,1) & j2>0 & j2<size(VF,1)
VF2(i2,j2) = VF(i,j);
end
end, end
figure, imshowpair(A, VF2);
```

1.9 Boundary Extraction

- Boundary extraction is used very frequently in image processing and analysis.
- The boundary of a set A denoted by $\beta(A)$, is obtained by erosion by an SE element B followed by the difference between the original image and the eroded image.

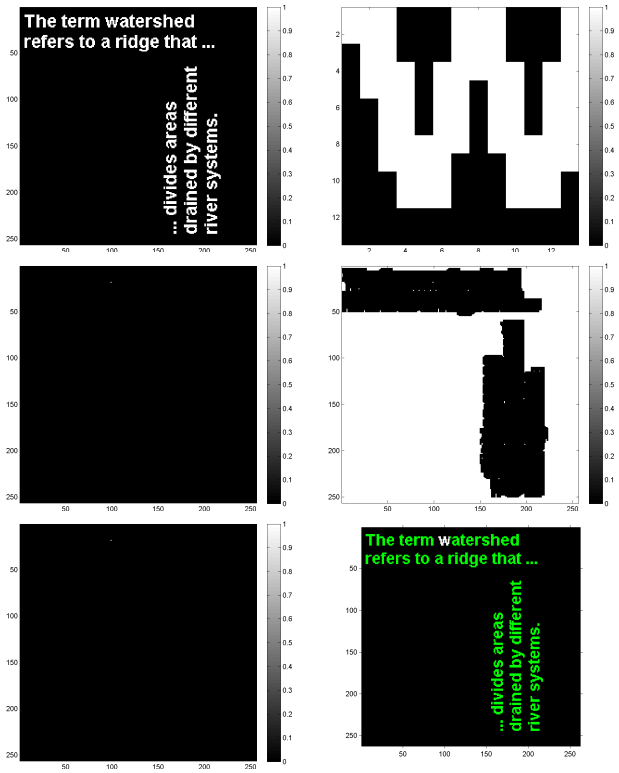


Figure 5: Example of duality between closing and opening.

- This is expressed by

$$\beta(A) = A - (A \ominus B).$$

- The boundary thickness depends on the structuring element.

Boundary extraction process

1. Read test image.
2. Create a square 3×3 structuring element.
3. Erode image by element.
4. Find the intersection between the original image and the complementary of eroded image.
5. Display boundary image.

```
% Boundary extraction example.
A = imread('circles.png');
figure, imagesc(A), colormap gray, colorbar, axis image;
se = ones(3,3);
B = imerode(A, se);
figure, imagesc(B), colormap gray, colorbar, axis image;
C = A & ~B;
figure, imagesc(C), colormap gray, colorbar, axis image;
```

Exercise 4. *Write a program that will find the outer boundary of a shape.*

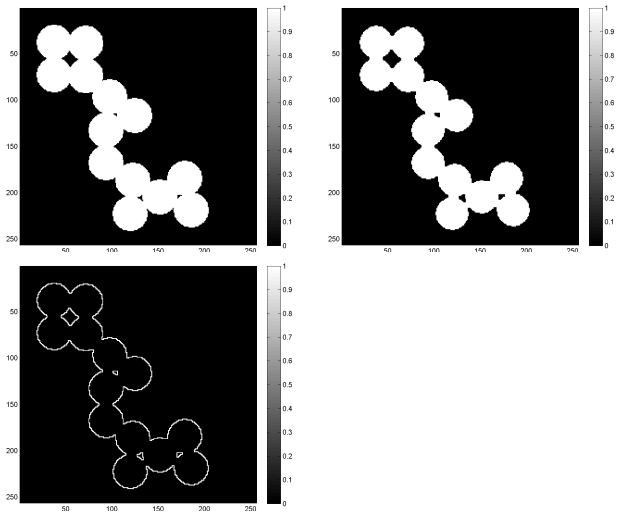


Figure 6: Boundary extraction example.