Experiment No 06

BECS 32461

Paper A

LINEAR AND CIRCULAR CONVOLUTION

Student Name: W. K. G. K. Jayawardana

Student No: EC/2021/006

Date Performed: 2025/10/31

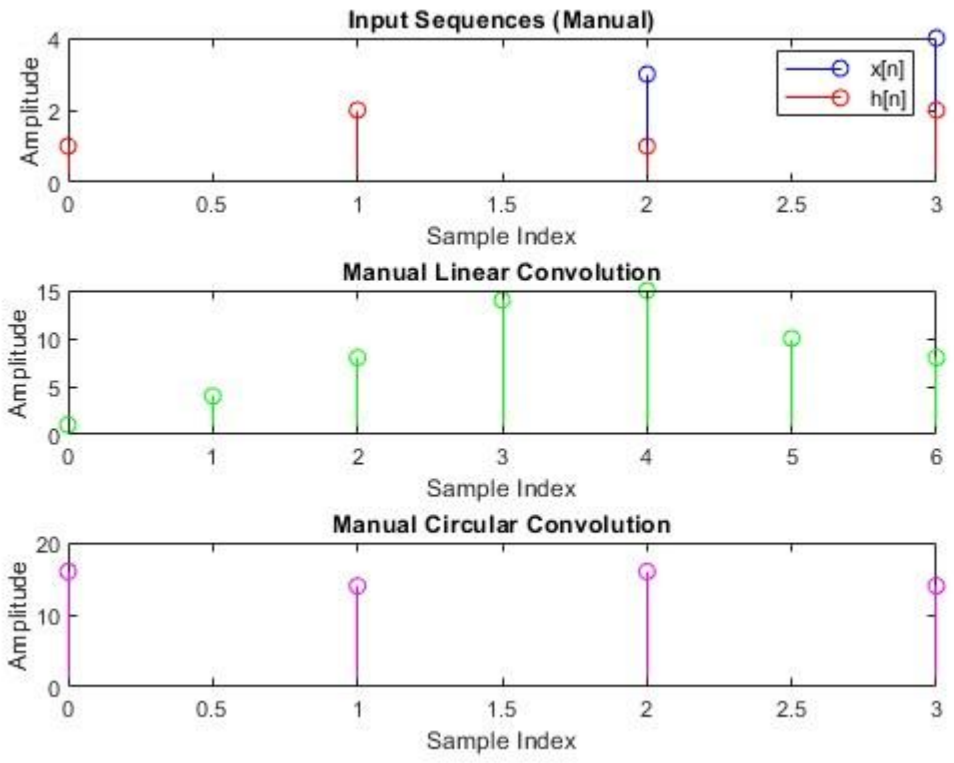
Date Submitted: 2025/10/31

PROCEDURE

F01.

```
x = [1 2 3 4];
h = [1 2 1 2];

m = length(x);
n = length(h);

L = m + n - 1;
N = max(m, n);

if m < n
x = [x zeros(1, N - m)];
h = [h zeros(1, N - n)];
end

ylc_manual = zeros(1, length(x) + length(h) - 1);
for n = 1:length(ylc_manual)
    for k = 1:length(x)
     if (n - k + 1) > 0 && (n - k + 1) <= length(h)
 ylc_manual(n) = ylc_manual(n) + x(k) * h(n - k + 1);
        end
    end
end

ycc_manual = zeros(1, N);
for k = 0:N-1
    sum = 0;
    for j = 0:N-1
        sum = sum + x(mod(k - j, N) + 1) * h(j + 1);
    end
    ycc_manual(k + 1) = sum;
end

figure;

subplot(3, 1, 1);
stem(0:length(x)-1, x, 'b', 'DisplayName', 'x[n]');
hold on;
stem(0:length(x)-1, h, 'r', 'DisplayName', 'h[n]');
hold off;
title('Input Sequences (Manual)');
xlabel('Sample Index');
ylabel('Amplitude');
legend;


subplot(3, 1, 2);
stem(0:L-1, ylc_manual, 'g');
title('Manual Linear Convolution');
xlabel('Sample Index');
ylabel('Amplitude');
```

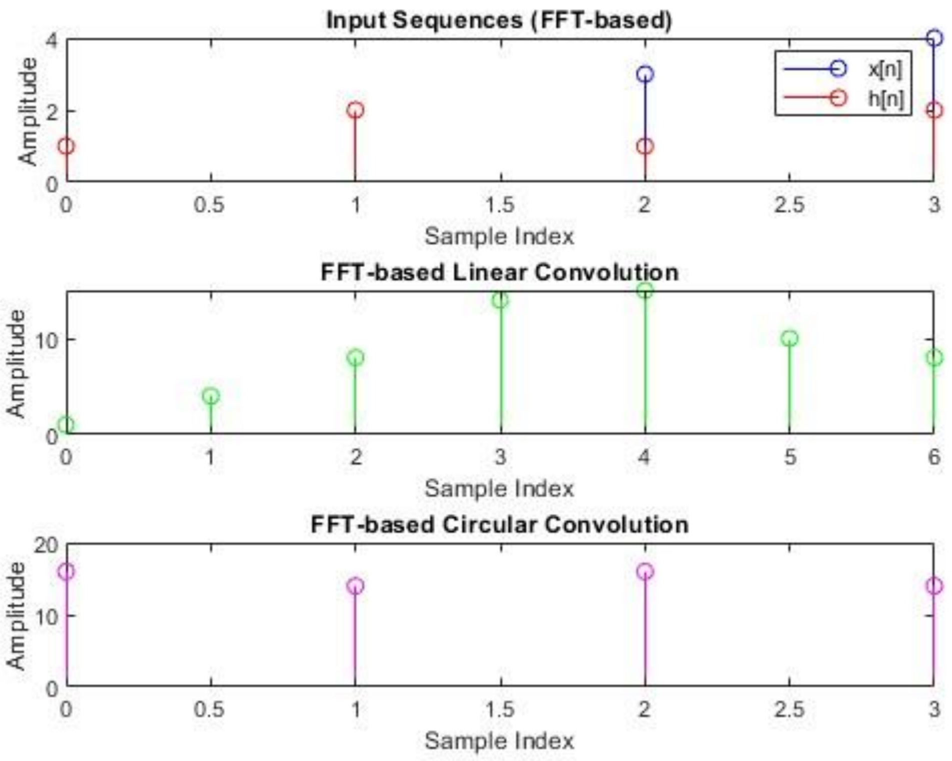
```
subplot(3, 1, 3);
stem(0:N-1, ycc_manual, 'm');
title('Manual Circular Convolution');
xlabel('Sample Index');
ylabel('Amplitude');
```



F02.

```
x = [1 2 3 4];
h = [1 2 1 2];

m = length(x);
n = length(h);

L = m + n - 1;
N = max(m, n);

x_padded = [x zeros(1, L - N)];
h_padded = [h zeros(1, L - N)];

ylc_fft = ifft(fft(x_padded) .* fft(h_padded));

ycc_fft = ifft(fft(x).* fft(h));

figure;
```

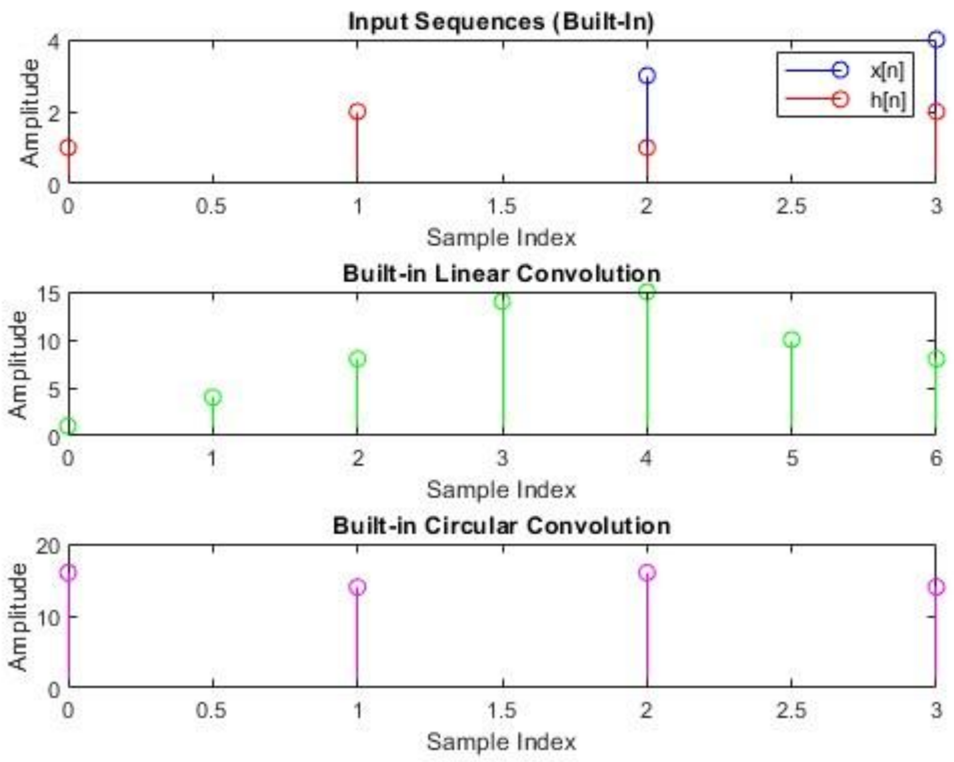
```
subplot(3, 1, 1);
stem(0:m-1, x, 'b', 'DisplayName', 'x[n]');
hold on;
stem(0:m-1, h, 'r', 'DisplayName', 'h[n]');
hold off;
title('Input Sequences (FFT-based)');
xlabel('Sample Index');
ylabel('Amplitude');
legend;

subplot(3, 1, 2);
stem(0:L-1, ylc_fft, 'g');
title('FFT-based Linear Convolution');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(3, 1, 3);
stem(0:N-1, ycc_fft, 'm');
title('FFT-based Circular Convolution');
xlabel('Sample Index');
ylabel('Amplitude');
```



F03.

```
x = [1 2 3 4];
h = [1 2 1 2];

ylc_builtin = conv(x, h);
```

```
N = max(length(x), length(h));
ycc_builtin = cconv(x, h, N);

figure;

subplot(3, 1, 1);
stem(0:length(x)-1, x, 'b', 'DisplayName', 'x[n]');
hold on;
stem(0:length(h)-1, h, 'r', 'DisplayName', 'h[n]');
hold off;
title('Input Sequences (Built-In)');
xlabel('Sample Index');
ylabel('Amplitude');
legend;


subplot(3, 1, 2);
stem(0:length(ylc_builtin)-1, ylc_builtin, 'g');
title('Built-in Linear Convolution');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(3, 1, 3);
stem(0:N-1, ycc_builtin, 'm');
title('Built-in Circular Convolution');
xlabel('Sample Index');
ylabel('Amplitude');
```

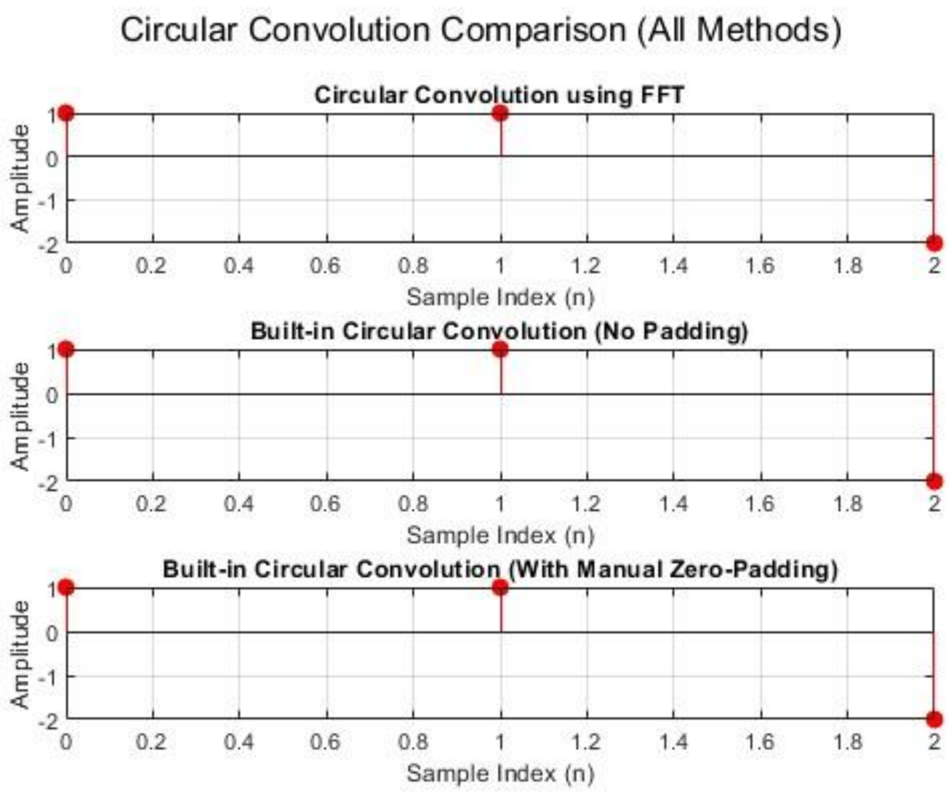


EXERCISE

E01.

```
x = [1 0 -1];
h = [2 1];

N = max(length(x), length(h));

% --- Zero-padding for FFT method ---
x_padded_fft = [x, zeros(1, N - length(x))];
h_padded_fft = [h, zeros(1, N - length(h))];

% --- (a) Circular Convolution using FFT ---
y_fft = ifft(fft(x_padded_fft) .* fft(h_padded_fft));

% --- Built-in circular convolution (no manual padding) ---
y_builtin_no_padding = cconv(x, h, N);

% --- (c) Built-in circular convolution with manual zero-padding ---
h_padded = [h, zeros(1, length(x) - length(h))];
y_builtin_padded = cconv(x, h_padded, length(x));

% --- Display numerical results ---
disp('Circular Convolution using FFT:');
disp(y_fft);
disp('Built-in Circular Convolution (without manual padding):');
disp(y_builtin_no_padding);
disp('Built-in Circular Convolution with Manual Zero-Padding:');
disp(y_builtin_padded);

% --- (b) Plot all results using subplots ---
figure;

% Subplot 1: FFT Method
subplot(3,1,1);
stem(0:N-1, y_fft, 'r', 'filled');
title('Circular Convolution using FFT');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;

% Subplot 2: Built-in cconv (No Padding)
subplot(3,1,2);
stem(0:N-1, y_builtin_no_padding, 'r', 'filled');
title('Built-in Circular Convolution (No Padding)');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;

% Subplot 3: Built-in cconv (With Manual Zero-Padding)
subplot(3,1,3);
```

```
stem(0:length(y_builtin_padded)-1, y_builtin_padded, 'r', 'filled');
title('Built-in Circular Convolution (With Manual Zero-Padding)');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;

sgtitle('Circular Convolution Comparison (All Methods)');
```



```
>> E01
Circular Convolution using FFT:
     1     1    -2

Built-in Circular Convolution (without manual padding):
     1     1    -2

Built-in Circular Convolution with Manual Zero-Padding:
     1     1    -2
```

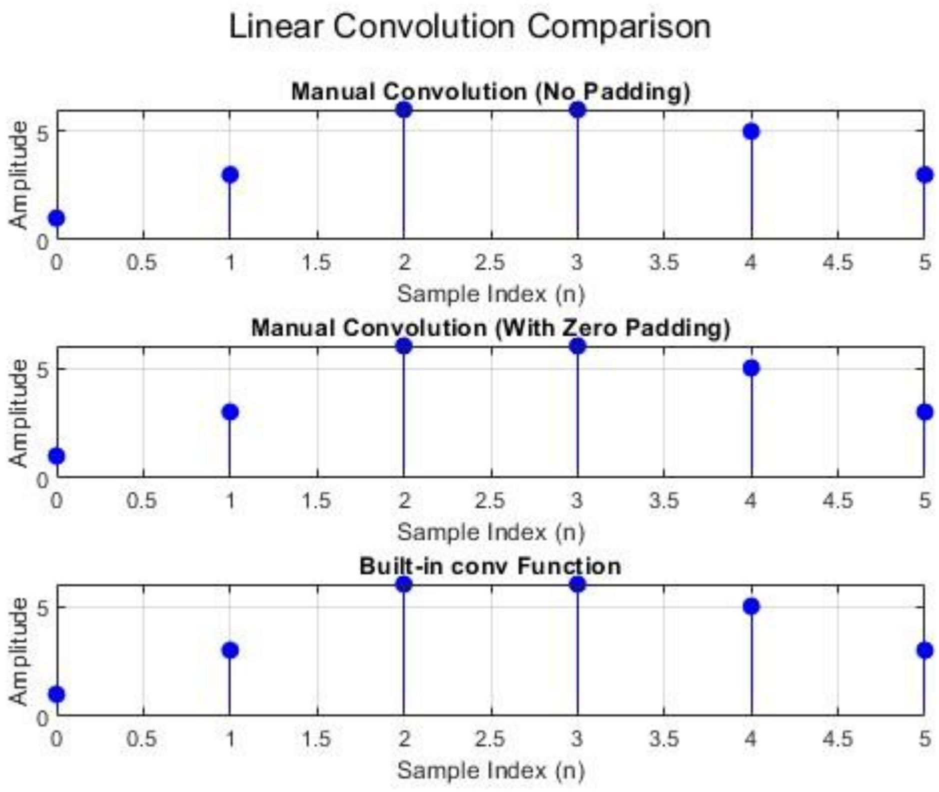
Circular convolution treats signals as if they are repeating in a circle. For this to work correctly, both signals need to be the same length so they can "wrap around" and align properly.

If one signal is shorter than the other, we add zeros to the end of its process called **zero-padding**. This makes both signals equal in length and ensures the calculation is accurate. Without padding, the result would be incorrect because the signals wouldn't align properly.

E02.

```
% --- (a) ---
x = [1 2 3];
h = [1 1 1 1];

m = length(x);
n = length(h);
L = m + n - 1; % Output length for linear convolution

% (i) Manual Convolution without Zero Padding (Correct Implementation)
ylc_manual_no_pad = zeros(1, L);
for i = 1:L
    for k = 1:m
        if (i - k + 1) > 0 && (i - k + 1) <= n
            ylc_manual_no_pad(i) = ylc_manual_no_pad(i) + x(k) * h(i - k + 1);
        end
    end
end

% (ii) Manual Convolution with Zero Padding
x_padded = [x, zeros(1, L - m)];
h_padded = [h, zeros(1, L - n)];
ylc_manual_padded = zeros(1, L);
for i = 1:L
    for k = 1:i
        ylc_manual_padded(i) = ylc_manual_padded(i) + x_padded(k) * h_padded(i - k +
1);
    end
end

% (iii) Built-in Convolution
ylc_builtin = conv(x, h);

disp('Manual Convolution (No Padding):');
disp(ylc_manual_no_pad);
disp('Manual Convolution (With Padding):');
disp(ylc_manual_padded);
disp('Built-in conv Function:');
disp(ylc_builtin);

% --- (b) Plotting the results ---
figure;

subplot(3,1,1);
stem(0:L-1, ylc_manual_no_pad, 'b', 'filled');
title('Manual Convolution (No Padding)');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;

subplot(3,1,2);
stem(0:L-1, ylc_manual_padded, 'b', 'filled');
title('Manual Convolution (With Zero Padding)');
xlabel('Sample Index (n)');
```

```
ylabel('Amplitude');
grid on;

subplot(3,1,3);
stem(0:L-1, ylc_builtin, 'b', 'filled');
title('Built-in conv Function');
xlabel('Sample Index (n)');
ylabel('Amplitude');
grid on;

sgtitle('Linear Convolution Comparison');
```



```
>> E02
Manual Convolution (No Padding):
     1     3     6     6     5     3

Manual Convolution (With Padding):
     1     3     6     6     5     3

Built-in conv Function:
     1     3     6     6     5     3
```

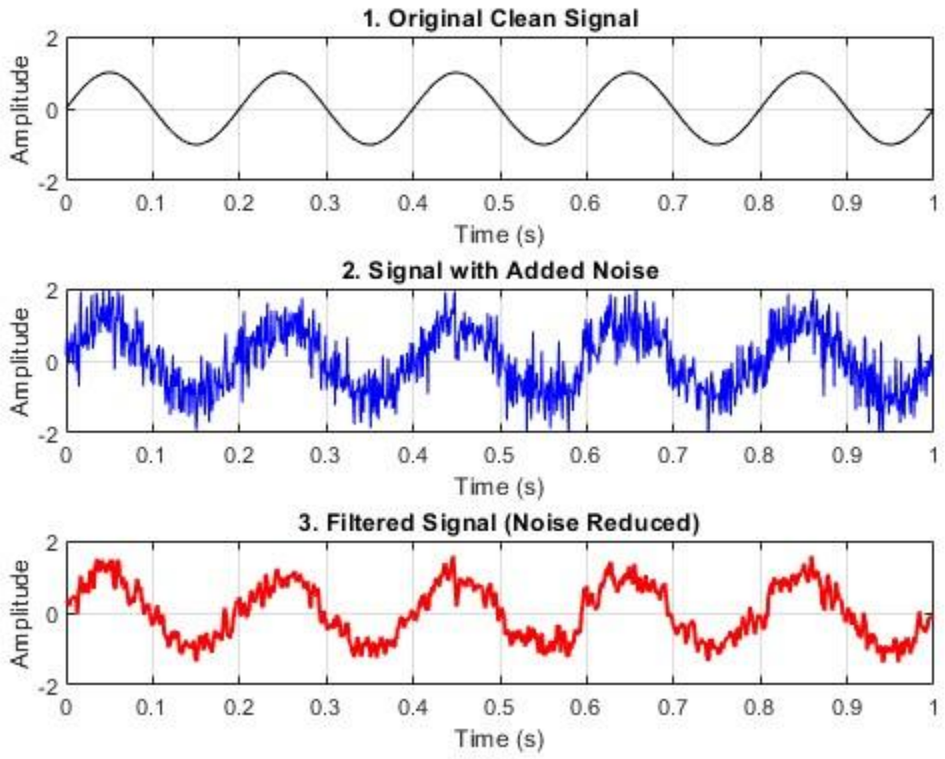
When calculating linear convolution, all the methods in the exercise give the same correct result. However, zero-padding is **extremely important** when you use the **FFT (Fast Fourier Transform)** method to find the linear convolution.

Here's why: using the FFT to multiply signals is a shortcut that naturally produces a *circular* convolution result. To get the correct *linear* convolution result, you must first pad both signals with zeros to a combined length of at least $M + N - 1$ (where M and N are the original lengths). This padding prevents the "wrap-around" effect of circular convolution from corrupting the answer.

E03.

```
% --- (i) ---
fs = 1000;
f = 5;
t = 0:1/fs:1-1/fs;

sine_wave = sin(2 * pi * f * t);

noise = 0.5 * randn(size(t));
noisy_signal = sine_wave + noise;


% --- (ii)  ---
h_avg = [0.25 0.25 0.25 0.25];


% --- (iii)---

filtered_signal = conv(noisy_signal, h_avg, 'same');


% --- (iv)---
figure;

subplot(3, 1, 1);
plot(t, sine_wave, 'k');
title('1. Original Clean Signal');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
ylim([-2, 2]);


subplot(3, 1, 2);
plot(t, noisy_signal, 'b');
title('Signal with Added Noise');
xlabel('Time (s)');
ylabel('Amplitude');
```

```
grid on;
ylim([-2, 2]);


subplot(3, 1, 3);
plot(t, filtered_signal, 'r', 'LineWidth', 1.5);
title('Filtered Signal (Noise Reduced)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
ylim([-2, 2]); n
```



A moving average filter smooths out a signal by reducing sharp spikes. It works by making each new point in the signal an **average** of the last few points from the original noisy signal.

Random noise is made of quick, random ups and downs. When you average a small group of these noisy points, the random positive and negative values tend to cancel each other out. This process removes the sharp, jerky parts of the noise while keeping the main, underlying signal (like the sine wave) intact. The result is a much cleaner and smoother signal.

E04.

```
% --- (i) ---

try
    img_orig = imread('cameraman.tif');
catch
    disp('Error: Could not find "cameraman.tif".');
    disp('Please ensure the Image Processing Toolbox is installed and the file is 
accessible.');
    return;
end

img_double = double(img_orig);

% --- (ii) ---
h_avg = (1/9) * ones(3, 3);

% Sharpening filter
h_sharp = [ 0 -1  0;
           -1  5 -1;
            0 -1  0];

% --- (iii) Perform 2D Convolution ---
img_blurred = conv2(img_double, h_avg, 'same');

% Apply sharpening filter
img_sharpened = conv2(img_double, h_sharp, 'same');


% --- (iv) ---
img_blurred = uint8(img_blurred);
img_sharpened = uint8(img_sharpened);

figure;
% Original Image
subplot(1, 3, 1);
imshow(img_orig);
```

```
title('Original Image');

% Blurred Image
subplot(1, 3, 2);
imshow(img_blurred);
title('Filtered (Averaging)');

% Sharpened Image
subplot(1, 3, 3);
imshow(img_sharpened);
title('Filtered (Sharpening)');
```