Image Enhancement in Frequency Domain Miscellaneous Processes

Image Processing

Project Intro Video

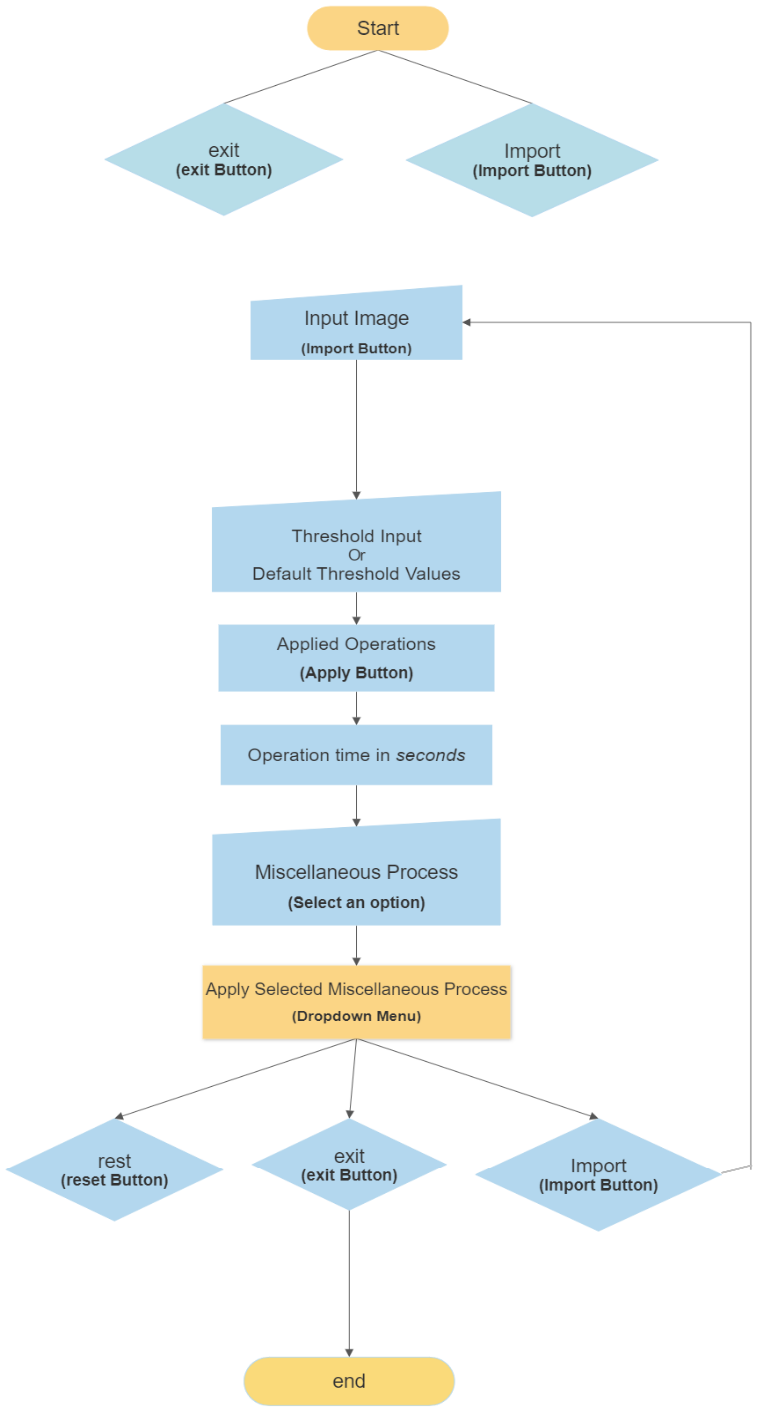
CONTENTS

Project Documentation

※ Project description and used functions:

- ▣ Implemented using *MATLAB R2016a*
- ▣ Preview Section (Original Image ⇒ Filtered Image).
- ▣ Menu Section (Import, apply, reset and exit).
- ▣ Operation time in seconds.
- ▣ Applied Operations section:
  - ↪ Main Operations Sub-section
    - ✓ High-pass filter
    - ✓ Low-pass filter
    - ✓ Band-pass filter
    - ✓ Homomorphic High-pass filter
    - ✓ Noise Removal
  - + Threshold values for each filter
  - + Default threshold values
- ▣ Miscellaneous Operations:
  - ✓ Histogram Equalization
  - ✓ Contrast Stretching
  - ✓ Robert Edge Detection
  - ✓ Sobel Edge Detection
  - ✓ Prewitt Edge Detection
  - ✓ Laplacian Edge Detection
  - ✓ Canny Edge Detection
  - ✓ Zerocross Edge Detection
  - ✓ Laplacian of Gaussian (LoG) Edge Detection
  - ✓ Difference of Gaussian (DoG) Edge Detection
  - ✓ Lossy JPEG Compression
- ▣ Time complexity of the program: $O(n^{2.37188})\ time$

※ Flow chart of program steps:



※ Algorithm of program steps ※

Step (1): Start, open the program GUI.

Step (2): From the top left corner menu, choose either Import option to select an image, or exit to close the program.

Step (3): In case you choose import option, you are prompted to input threshold values manually or check the 'Default threshold values' button for default threshold values.

Step (4): Afterwards, you may proceed and click on the 'apply' button to process the input image.

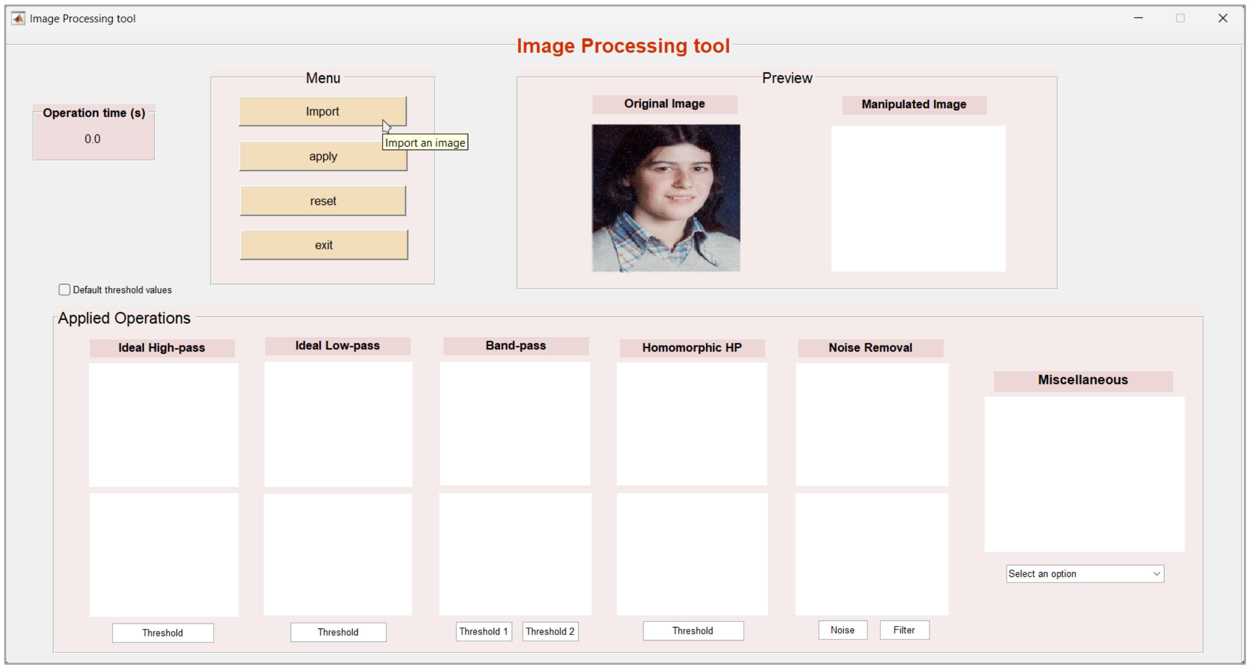
Step (5): Ability to live change the thresholds by edition the threshold values and clicking 'apply' to apply changes.

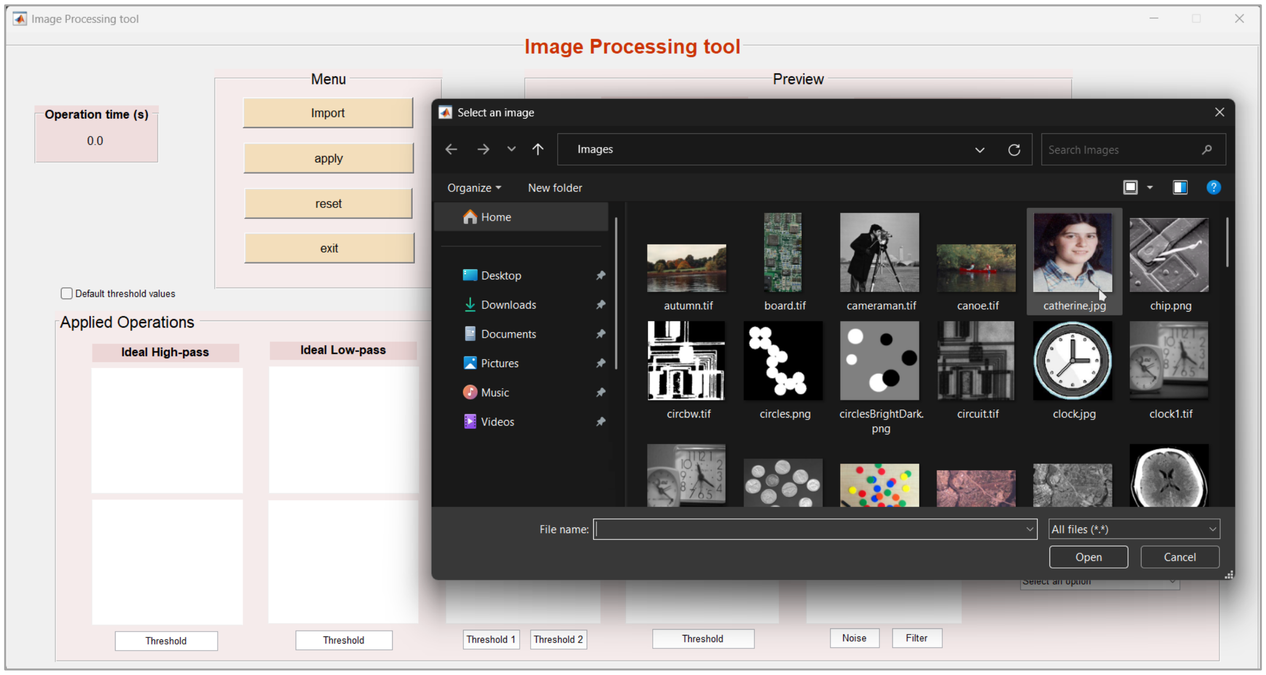
Step (6): An optional part is to select an option from the 'Miscellaneous' dropdown menu and it will be auto applied on the input image.

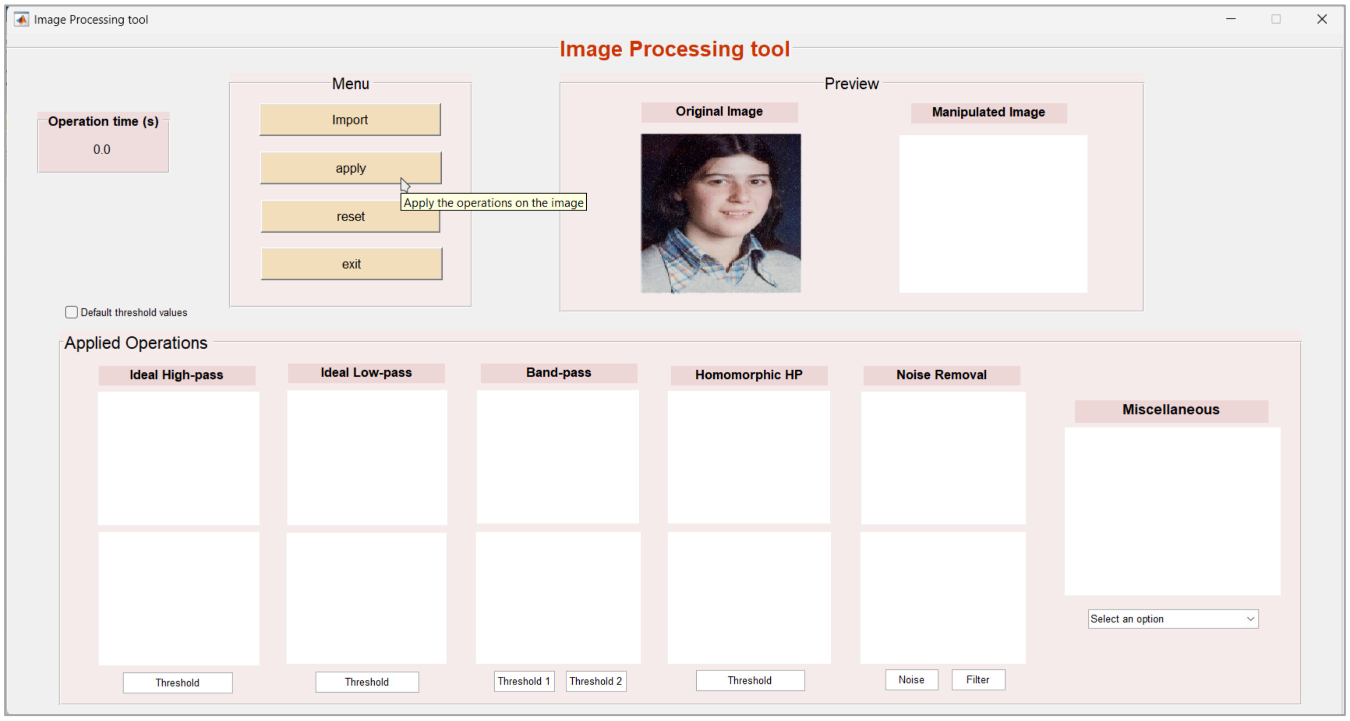
Step (7): Either reset everything by clicking the 'reset' button, or import another image by clicking the 'import' button and going to step 2, or even exit the program by clicking the 'exit' button.

Step (8): End, program termination.

※ Screenshots of the program ※

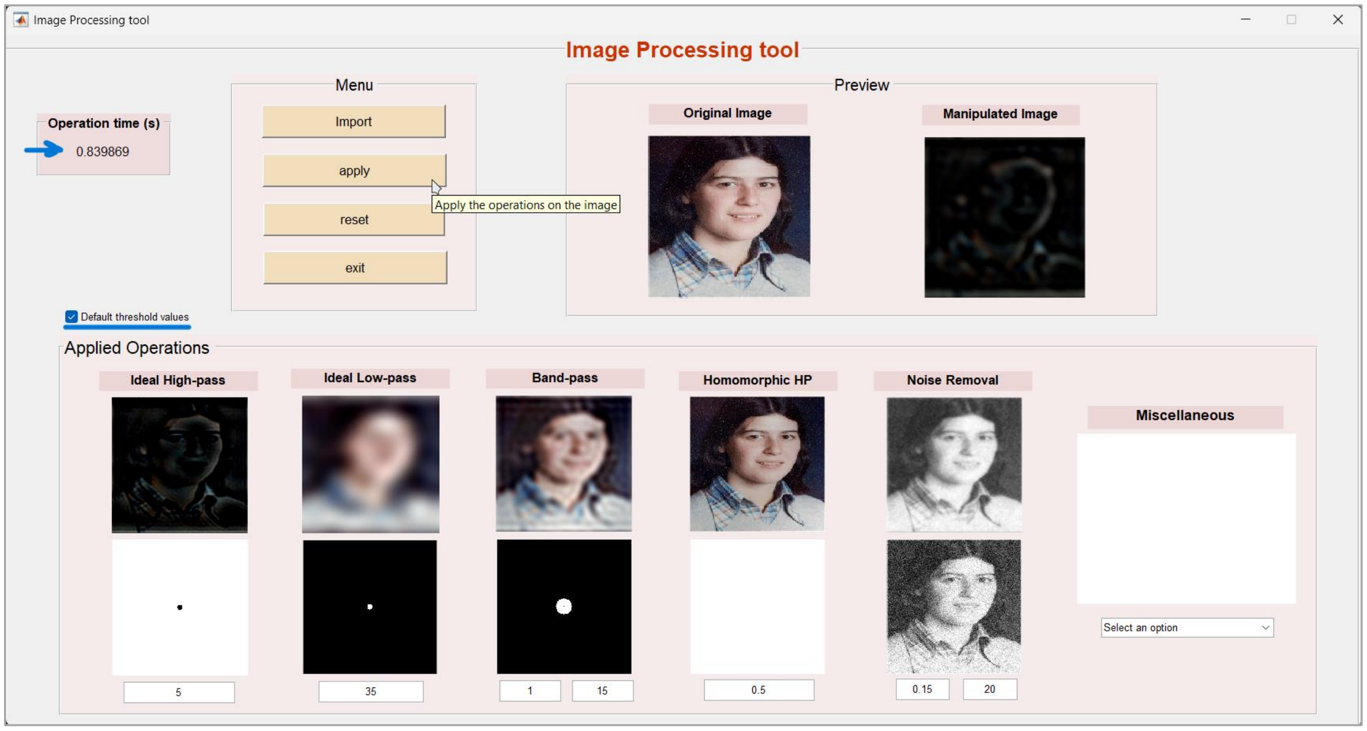






Operation time (s)
0.0

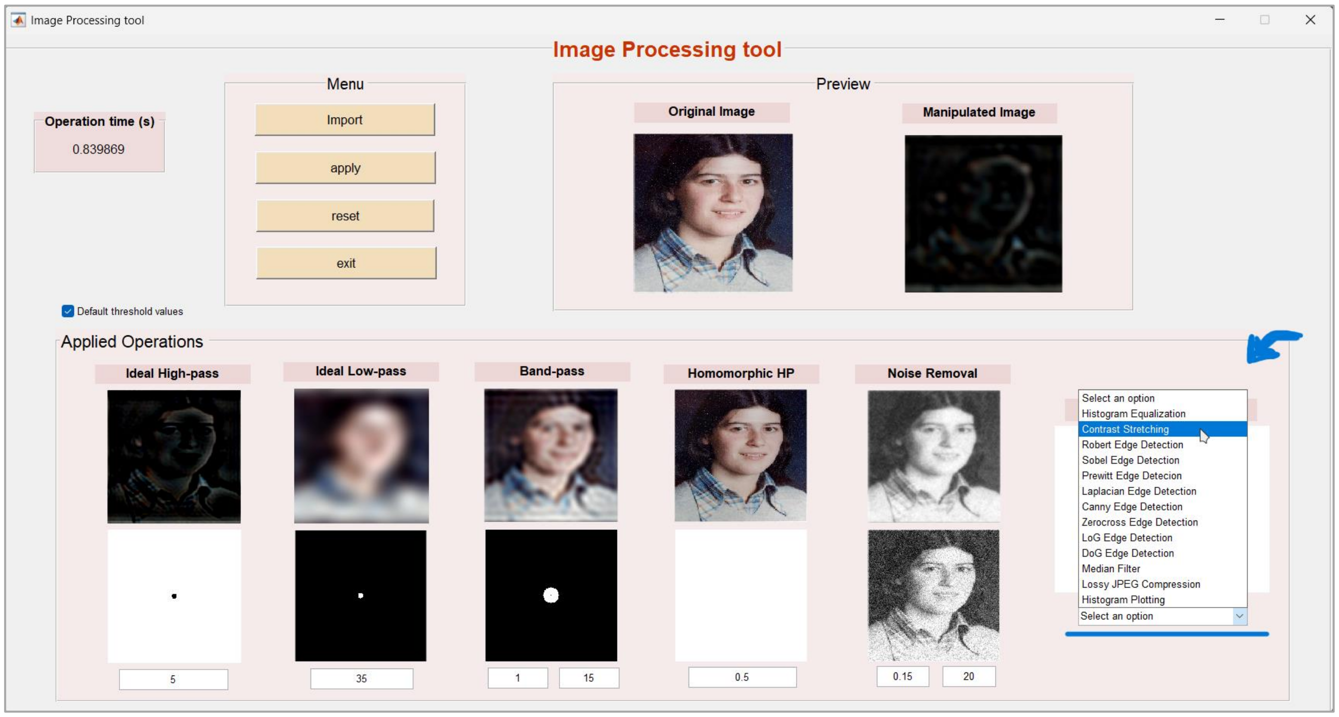
Default threshold values



Operation time (s)
0.839869

[✓ Default threshold values]

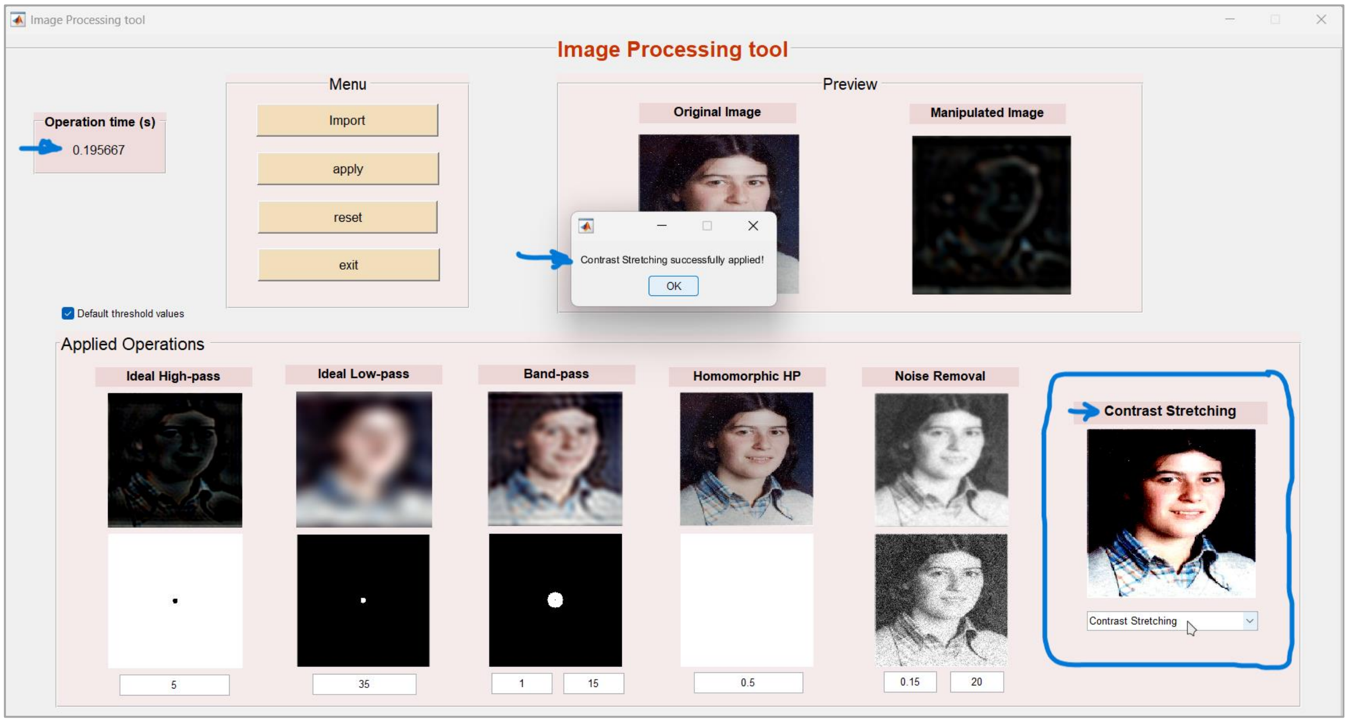
Ideal High-pass	Ideal Low-pass	Band-pass		Homomorphic HP	Noise Removal	
5	35	1	15	0.5	0.15	20



Operation time (s): 0.839869

Default threshold values ✓

Ideal High-pass: 5 | Ideal Low-pass: 35 | Band-pass: 1, 15 | Homomorphic HP: 0.5 | Noise Removal: 0.15, 20



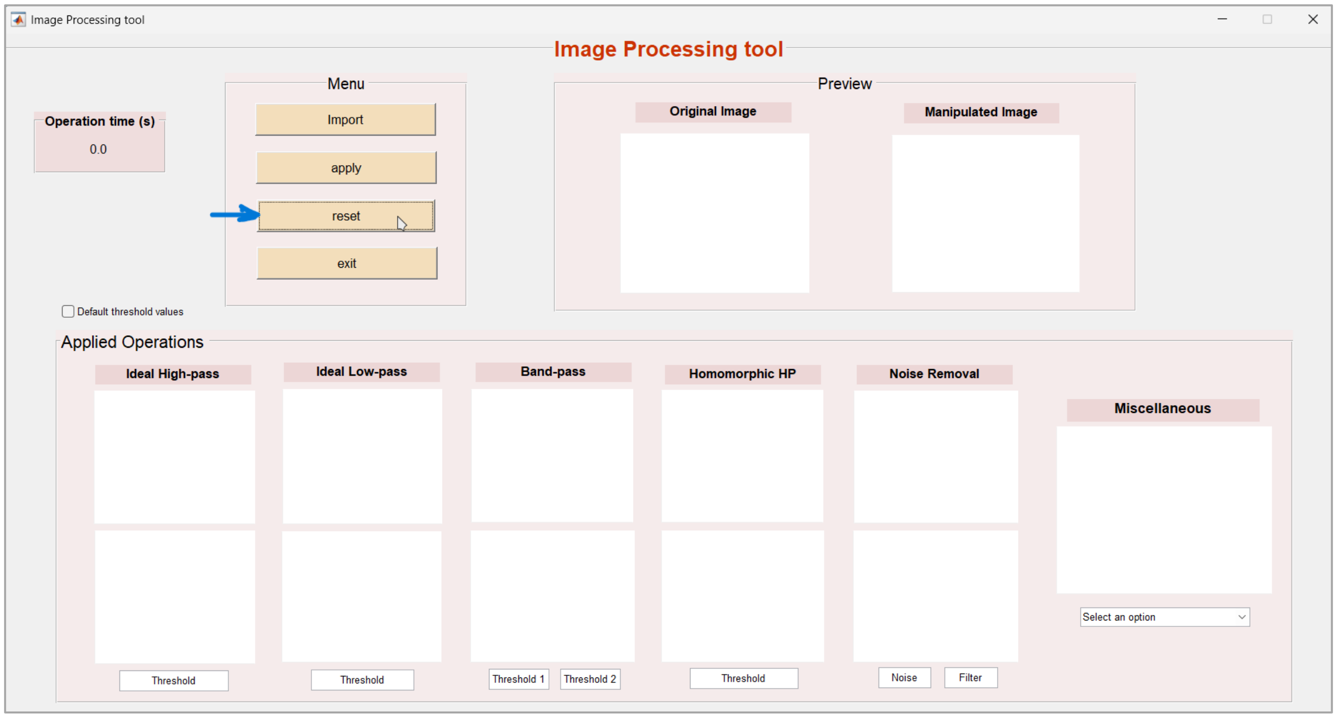
Operation time (s): 0.195667

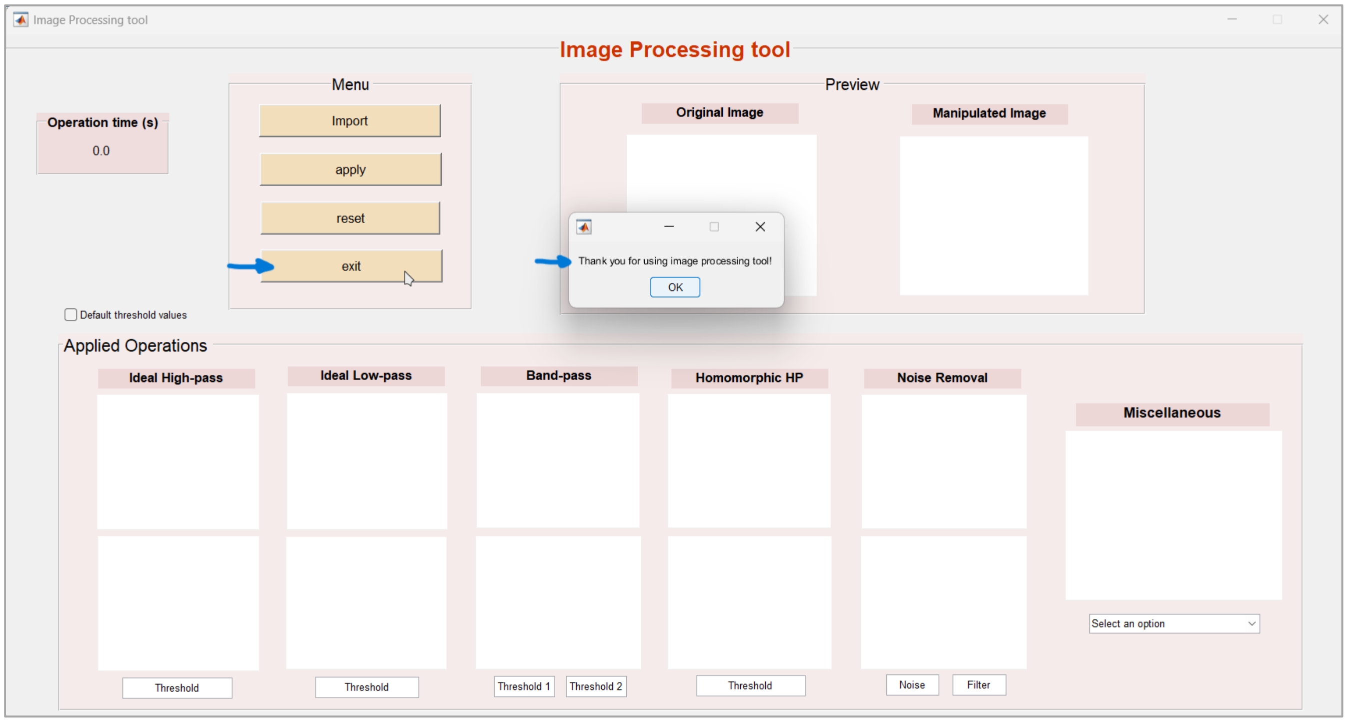
Contrast Stretching successfully applied!

Default threshold values ✓

Ideal High-pass: 5 | Ideal Low-pass: 35 | Band-pass: 1, 15 | Homomorphic HP: 0.5 | Noise Removal: 0.15, 20

Contrast Stretching





Thank you for using image processing tool!

※ Screenshots of the code ※

```
% --------------------------------------------------------------------
%%%***** Main buttons functions *****%%%
% --------------------------------------------------------------------

% --- Executes on 'import button' press
function import_Callback(hObject, eventdata, handles)

global i
global filename


[filename, pathname] = uigetfile({
    '*.*', 'All files (*.*)';
    '*.jpg','jpg Files (*.jpg)';
    '*.JPG','JPG Files (*.JPG)';
    '*.png','png Files (*.png)';
    '*.PNG','PNG Files (*.PNG)';
    '*.jpeg','jpeg Files (*.jpeg)';
    '*.JPEG','JPEG Files (*.JPEG)';
    '*.img','img Files (*.img)';
    '*.IMG','IMG Files (*.IMG)';
    '*.tif','tif Files (*.tif)';
    '*.TIF','TIF Files (*.TIF)';
    '*.tiff','tiff Files (*.tiff)';
    '*.TIFF','TIFF Files (*.TIFF)'}, 'Select an image',
'C:\Users\admin\Desktop\Image Processing\Practical\Images');

% Error check - if no filename, there is an error
if isequal(filename,0)
    msgbox('Load Error. No files selected!', 'Error', 'error');
else
    i = imread(fullfile(pathname, filename));
    axes(handles.axes1);
    imshow(i);
end


% --- Executes on 'reset button' press
function reset_Callback(hObject, eventdata, handles)
clearAxis(handles.axes1);
clearAxis(handles.axes2);
clearAxis(handles.axes3);
clearAxis(handles.axes4);
clearAxis(handles.axes5);
clearAxis(handles.axes6);
clearAxis(handles.axes7);
clearAxis(handles.axes8);
clearAxis(handles.axes9);
clearAxis(handles.axes10);
clearAxis(handles.axes11);
clearAxis(handles.axes12);
clearAxis(handles.axes13);

set(handles.lp_threshold, 'String', 'Threshold');
set(handles.hp_threshold, 'String', 'Threshold');
set(handles.bp_threshold1, 'String', 'Threshold 1');
set(handles.bp_threshold2, 'String', 'Threshold 2');
set(handles.homo_threshold, 'String', 'Threshold');
set(handles.noise_threshold, 'String', 'Noise');
set(handles.filter_threshold, 'String', 'Filter');

set(handles.miscellaneous, 'String', 'Miscellaneous');

set(handles.time, 'String', '0.0');


% --- Executes on 'exit button' press.
function exit_Callback(hObject, eventdata, handles)

msgbox('Thank you for using image processing tool!');
pause(1);
close();
close();



% --------------------------------------------------------------------
%%% ***** Main operations function *****%%%
% --------------------------------------------------------------------

% --- Executes on 'apply button' press
function apply_Callback(hObject, eventdata, handles)

% Check if the user has input a threshold value

% Thresholds array
t = [handles.hp_threshold,    ...
     handles.lp_threshold,    ...
     handles.bp_threshold1,   ...
     handles.bp_threshold2,   ...
     handles.homo_threshold,  ...
     handles.noise_threshold, ...
     handles.filter_threshold];

isEmptyThreshold = '';

for i=1:length(t)
    value = str2double(get(t(i), 'String'));

    if isnan(value)
        isEmptyThreshold = 'Empty';
    end
```

```
end


% Apply the operations only if a threshold value is provided or in case of a default
threshold
default_threshold = get(handles.default_thresholds, 'Value');

if default_threshold == 1

    % Noise threshold
    noise_threshold = 0.15;
    filter_threshold = 20;

    set(handles.noise_threshold, 'String', noise_threshold);
    set(handles.filter_threshold, 'String', filter_threshold);


    % Low-pass threshold
    lp_threshold = 35;
    set(handles.lp_threshold, 'String', lp_threshold);


    % High-pass threshold
    hp_threshold = 5;
    set(handles.hp_threshold, 'String', hp_threshold);


    % Band-pass threshold
    bp_threshold1 = 1;
    bp_threshold2 = 15;
    set(handles.bp_threshold1, 'String', bp_threshold1);
    set(handles.bp_threshold2, 'String', bp_threshold2);


    % Homomorphic threshold
    homo_threshold = 0.5;
    set(handles.homo_threshold, 'String', homo_threshold);

else

    % Noise threshold
    noise_threshold = str2double(get(t(6), 'String'));
    filter_threshold = str2double(get(t(7), 'String'));

    % Low-pass threshold
    lp_threshold = str2double(get(t(2), 'String'));

    % High-pass threshold
    hp_threshold = str2double(get(t(1), 'String'));

    % Band-pass threshold
    bp_threshold1 = str2double(get(t(3), 'String'));
    bp_threshold2 = str2double(get(t(4), 'String'));

    % Homomorphic threshold
    homo_threshold = str2double(get(t(5), 'String'));

end
```

```
% Apply the operations
if ~strcmp(isEmptyThreshold, 'Empty') || default_threshold ~= 0

    global i
    img = i;


    % Operations start time
    tStart = tic;


    % Noise removal (we use gaussian noise in our case)
    NF = NoiseRemoval(img, 'gaussian', noise_threshold, filter_threshold, handles);  %
Note: noise type is changeable

    % Low-pass filtering
    LPF = lpfilter(img, lp_threshold, handles);
    lpfilter(img, lp_threshold, handles);

    % High-pass filtering
    HPF = hpfilter(LPF, hp_threshold, handles);
    lpfilter(img, hp_threshold, handles);

    % Band-pass filtering
    BPF = bpfilter(HPF, bp_threshold1, bp_threshold2, handles);
    bpfilter(img, bp_threshold1, bp_threshold2, handles);

    % Homomorphic filtering
    HMF = homomorphic(BPF, homo_threshold, handles);
    homomorphic(img, homo_threshold, handles);

    % Show final manipulated image
    Res = HMF;
    axes(handles.axes2);
    imshow(Res, []);


    % Operations end time
    tEnd = toc(tStart);

    % Display operations time
    set(handles.time, 'String', tEnd);
else
  msgbox('A threshold value is required or check the default values!', 'Error', 'error');
end



% -----------------------------------------------------------------------------
%%% ***** Required functions *****%%%
% -----------------------------------------------------------------------------

% Noise removal function (using Butterworth LP filter)
function [out] = NoiseRemoval(input_image, type, noise_threshold,
filter_threshold, handle)
a = handleRGB(input_image);

% define noise
noisy_image = imnoise(uint8(a), type, noise_threshold);
```

```
axes(handle.axes13);
imshow(noisy_image);

% Apply Fourier Transform, then shift to the origin
ft = fftshift(fft2(noisy_image));

% use butterworth low-pass filter
H = butterlp(ft, filter_threshold, 2);

conv = H .* ft;

% Apply Inverse Fourier Transform
out_im = ifft2(conv);

% display denoised image
ifftshow(out_im, handle.axes12);


% return manipulated image
out = out_im;

% Ideal Low-Pass Filter
function [out] = lpfilter(input_image, threshold, handle)

I = input_image;

%------------- If an RGB image -------------
if ndims(I) == 3
   % Extract three images
    Red   =  I(: , : , 1);
    Green =  I(: , : , 2);
    Blue  =  I(: , : , 3);

    % Get the size of any channel (they'all are the same size)
    [r,c] = size(Red);


    % // Change - Transform each channel, then shift
    F_r = fftshift(fft2(Red));
    F_g = fftshift(fft2(Green));
    F_b = fftshift(fft2(Blue));



    %-- ILPF Filtering --

    % Find the center of the frequancy domain
    p = r ./ 2;
    q = c ./ 2;

    % Cut-off Frequancy
    d0 = threshold;

    % Initialize ILPF
    idealLP = zeros(c,r);
    % Create ILPF
    for i=1:r
        for j=1:c
            D = sqrt((i-p).^2 + (j-q).^2);
            idealLP(i,j) = D <= d0;
        end
    end

    % Display ILPF
    axes(handle.axes6);
    imshow(idealLP);
    % 3D: meshc(idealLP);


    %// Change - To match size
    idealLP = imresize(idealLP, [r c]);


    % // Now filter
    FF_R = idealLP .* F_r;
    FF_G = idealLP .* F_g;
    FF_B = idealLP .* F_b;


    %// Change - perform ifftshift, then ifft2, then cast to real
    % Inverse IFFT _RED
    Ir = ifftshift(FF_R);
    Irr = real(ifft2(Ir));


    % Inverse IFFT _Green
    Ig = ifftshift(FF_G);
    Igg = real(ifft2(Ig));


    % Inverse IFFT _Blue
    Ib = ifftshift(FF_B);
    Ibb = real(ifft2(Ib));


    % Combine the three components together
    %// Change - Removed fluff
    b = uint8(cat(3, Irr, Igg, Ibb));

    %// Visulaization - Display the final image in the spatial domain
    axes(handle.axes5);
    imshow(b, []);


    % return manipulated image
    out = b;

%------------- If not an RGB image -------------
else

    % find the size
    [r,c] = size(I);

    % Apply Fourier Transform to the original image
    im_f = fft2(I);

    % Shift image to the center
    f_shift = fftshift(im_f);

    % Find the center of the frequancy domain
    p = r ./ 2;
    q = c ./ 2;

    % Cut-off Frequancy
    d0 = threshold;

    % Initialize ILPF
    idealLP = zeros(c,r);

    % Create ILPF
    for i=1:r
        for j=1:c
            D = sqrt((i-p).^2 + (j-q).^2);
            idealLP(i,j) = D <= d0;
        end
    end


    %// Change - To match size
    idealLP = imresize(idealLP, [r c]);

    % Display the filter spectrum
    axes(handle.axes6);
    imshow(idealLP, []);


    % Convolve shifted image with ILPF
    convolveF = f_shift .* idealLP;

    % Shifted back the image
    original_image = ifftshift(convolveF);

    % Convert image to the spatial domain
    RImage = real(ifft2(original_image));

    % Display image in the spatial domain
    axes(handle.axes5);
    imshow(RImage, []);


    % return manipulated image
    out = RImage;
end


% Ideal High-Pass Filter
function [out] = hpfilter(input_image, threshold, handle)
```

```
I = input_image;

%------------- If an RGB image -------------
if ndims(I) == 3
   % Extract three images
    Red   =  I(: , : , 1);
    Green =  I(: , : , 2);
    Blue  =  I(: , : , 3);

    % Get the size of any channel (they'all are the same size)
    [r,c] = size(Red);


    % // Change - Transform each channel, then shift
    F_r = fftshift(fft2(Red));
    F_g = fftshift(fft2(Green));
    F_b = fftshift(fft2(Blue));



    %-- ILPF Filtering --

    % Find the center of the frequancy domain
    p = r ./ 2;
    q = c ./ 2;

    % Cut-off Frequancy
    d0 = threshold;

    % Initialize IHPF
    idealHP = zeros(c,r);

    % Create IHPF
    for i=1:r
        for j=1:c
            D = sqrt((i-p).^2 + (j-q).^2);
            idealHP(i,j) = D >= d0;
        end
    end

    %// Change - To match size
    idealHP = imresize(idealHP, [r c]);


    % Display IHPF
    axes(handle.axes4);
    imshow(idealHP);
    % 3D: meshc(idealHP);


    % // Now filter
    FF_R = idealHP .* F_r;
    FF_G = idealHP .* F_g;
    FF_B = idealHP .* F_b;
    %// Change - perform ifftshift, then ifft2, then cast to real
    % Inverse IFFT _RED
    Ir = ifftshift(FF_R);
    Irr = real(ifft2(Ir));

    % Inverse IFFT _Green
    Ig = ifftshift(FF_G);
    Igg = real(ifft2(Ig));

    % Inverse IFFT _Blue
    Ib = ifftshift(FF_B);
    Ibb = real(ifft2(Ib));

    % Combine the three components together
    %// Change - Removed fluff
    b = uint8(cat(3, Irr, Igg, Ibb));

    %// Visulaization - Display the final image in spatial domain
    axes(handle.axes3);
    imshow(b, []);

    % return manipulated image
    out = b;

%------------- If not an RGB image -------------
else

    % find the size
    [r,c] = size(I);

    % Apply Fourier Transform to the original image
    im_f = fft2(I);

    % Shift image to the center
    f_shift = fftshift(im_f);

    % Find the center of the frequancy domain
    p = r ./ 2;
    q = c ./ 2;

    % Cut-off Frequancy
    d0 = threshold;

    % Initialize IHPF
    idealHP = zeros(c,r);

    % Create IHPF
    for i=1:r
        for j=1:c
            D = sqrt((i-p).^2 + (j-q).^2);
            idealHP(i,j) = D >= d0;
        end
    end

    %// Change - To match size
    idealHP = imresize(idealHP, [r c]);

    % display the filter spectrum
    axes(handle.axes4);
    imshow(idealHP, []);


    % Convolve shifted image with IHPF
    convolveF = f_shift .* idealHP;

    % Shifted back the image
    original_image = ifftshift(convolveF);

    % Convert image to the spatial domain
    RImage = real(ifft2(original_image));

    % Display image in the spatial domain
    axes(handle.axes3);
    imshow(RImage, []);


    % return manipulated image
    out = RImage;
end


%-- Bandpass Filter
function [out] = bpfilter(input_image, threshold1, threshold2, handle)

I = input_image;

%------------- If an RGB image -------------
if ndims(I) == 3
   % Extract three images
    Red   =  I(: , : , 1);
    Green =  I(: , : , 2);
    Blue  =  I(: , : , 3);

    % Get the size of any channel (they'all are the same size)
    [r,c] = size(Red);


    % // Change - Transform each channel, then shift
    F_r = fftshift(fft2(Red));
    F_g = fftshift(fft2(Green));
    F_b = fftshift(fft2(Blue));


    %-- ILPF Filtering --
```

```
% Find the center of the frequancy domain
p = r ./ 2;
q = c ./ 2;

% Cut-off Frequancy
D01 = threshold1;
D02 = threshold2;

% Initialize BPF
bandPass = zeros(c,r);

% Create BPF
for i=1:r
    for j=1:c
        D = sqrt((i-p).^2 + (j-q).^2);
        bandPass(i,j) = (D >= D01 && D <= D02);
    end
end


%// Change - To match size
bandPass = imresize(bandPass, [r c]);

% Display BPF
axes(handle.axes8);
imshow(bandPass, []);
% 3D: meshc(bandPass);


% // Now filter
FF_R = bandPass .* F_r;
FF_G = bandPass .* F_g;
FF_B = bandPass .* F_b;


%// Change - perform ifftshift, then ifft2, then cast to real
% Inverse IFFT _RED
Ir = ifftshift(FF_R);
Irr = real(ifft2(Ir));


% Inverse IFFT _Green
Ig = ifftshift(FF_G);
Igg = real(ifft2(Ig));


% Inverse IFFT _Blue
Ib = ifftshift(FF_B);
Ibb = real(ifft2(Ib));


% Combine the three components together
%// Change - Removed fluff
b = uint8(cat(3, Irr, Igg, Ibb));
```

```
    %// Visulaization - Display the final image in spatial domain
    axes(handle.axes7);
    imshow(b, []);

    % return manipulated image
    out = b;

%------------- If not an RGB image -------------
else

    % find the size
    [r,c] = size(I);

    % Apply Fourier Transform to the original image
    im_f = fft2(I);

    % Shift image to the center
    f_shift = fftshift(im_f);

    % Find the center of the frequancy domain
    p = r ./ 2;
    q = c ./ 2;

    % Cut-off Frequancy
    D01 = threshold1;
    D02 = threshold2;

    % Initialize BPF
    bandPass = zeros(c,r);

    % Create BPF
    for i=1:r
        for j=1:c
            D = sqrt((i-p).^2 + (j-q).^2);
            bandPass(i,j) = (D >= D01 && D <= D02);
        end
    end


    %// Change - To match size
    bandPass = imresize(bandPass, [r c]);

    % Display BPF
    axes(handle.axes8);
    imshow(bandPass);


    % Convolve shifted image with IHPF
    convolveF = f_shift .* bandPass;

    % Shifted back the image
    original_image = ifftshift(convolveF);

    % Convert image to the spatial domain
    RImage = real(ifft2(original_image));
    % Display image in the spatial domain
    axes(handle.axes7);
    imshow(RImage, []);

    % return manipulated image
    out = RImage;
end
```

⋮ ⋮ ⋮

Thank you !