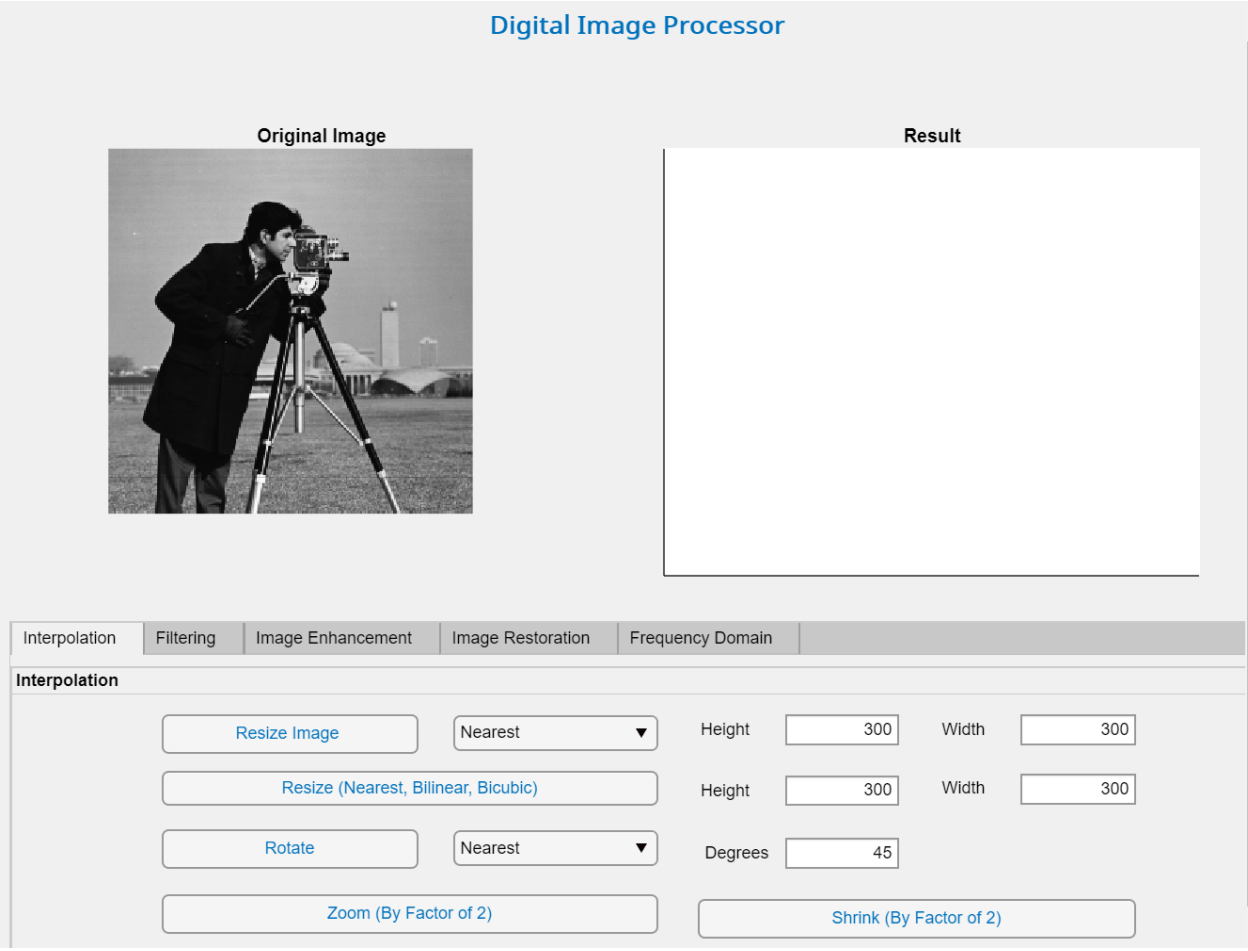
Digital Image Processing – Project Report

Ammar Ahmed (023-19-0107), Section A

Instructor: Dr. Ghulam Murtaza Memon

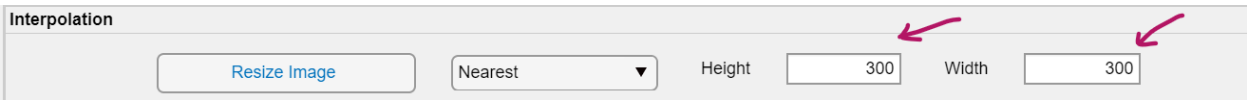
Task 1: Interpolation

Following is the initial interface of Interpolation tab. There are 4 interpolation operations the user can perform, let us go through each of them along with the demonstration of the output.

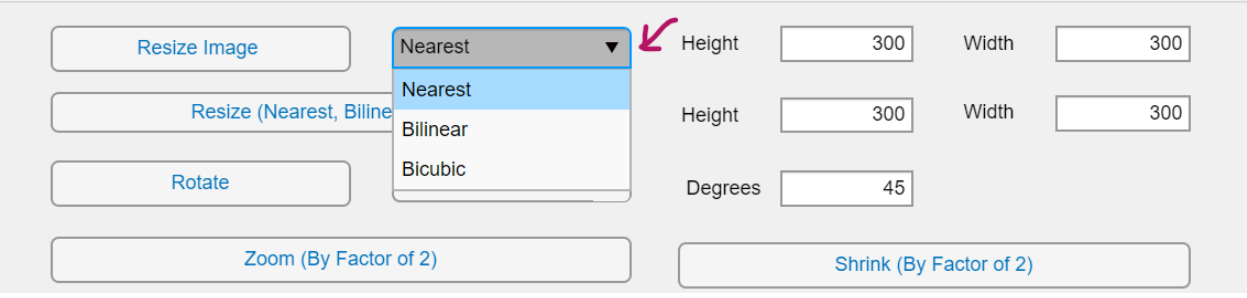


Resizing:

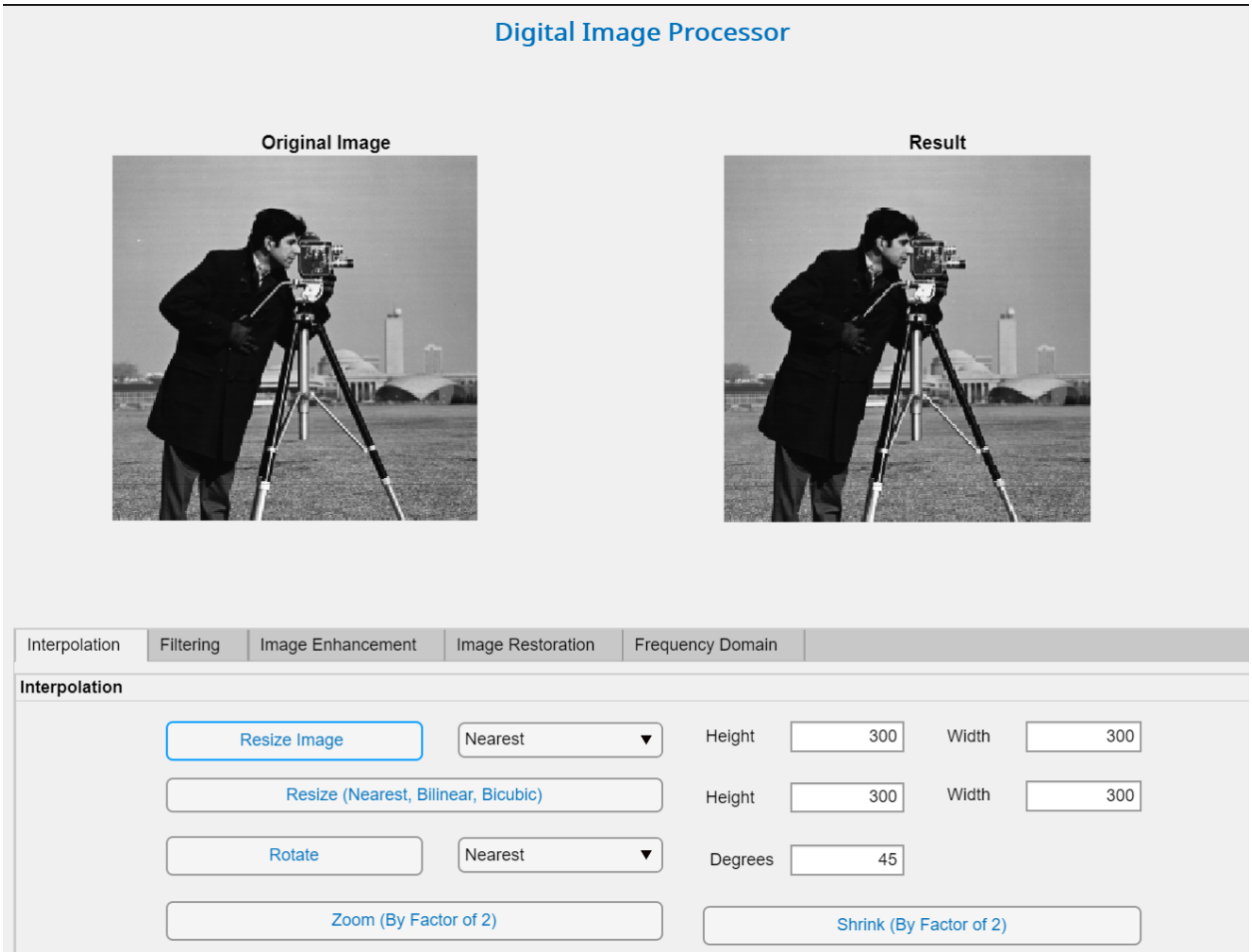
You can resize an image by specifying the height and the width of the image:



You can also specify the method of interpolation:

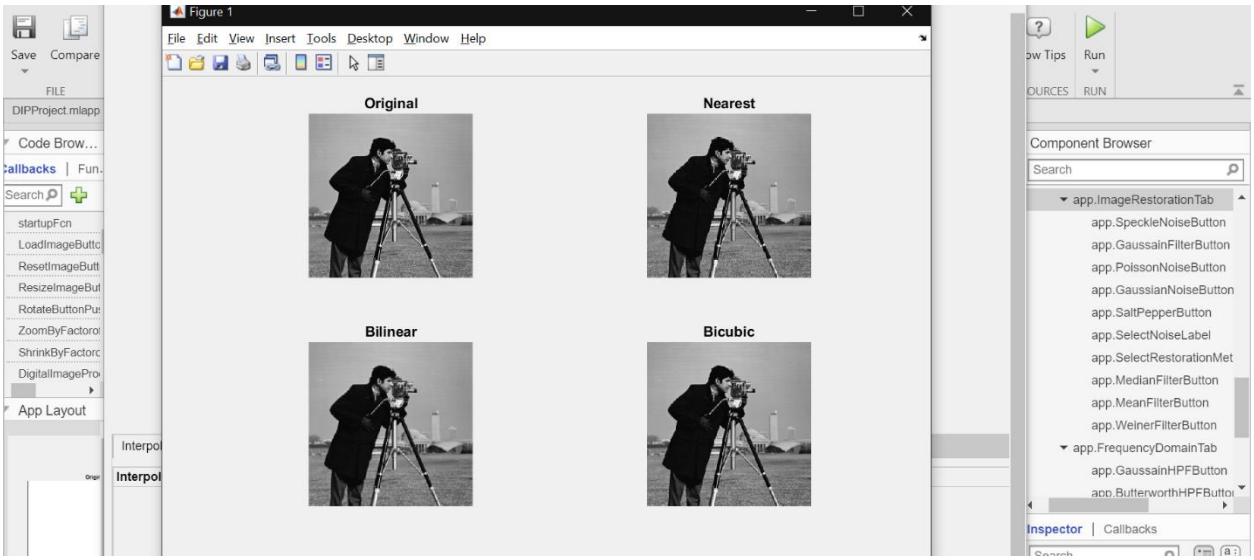


Output:



The user can also see the results of all methods (nearest, bilinear, bicubic) at once by clicking the following button.

Output:



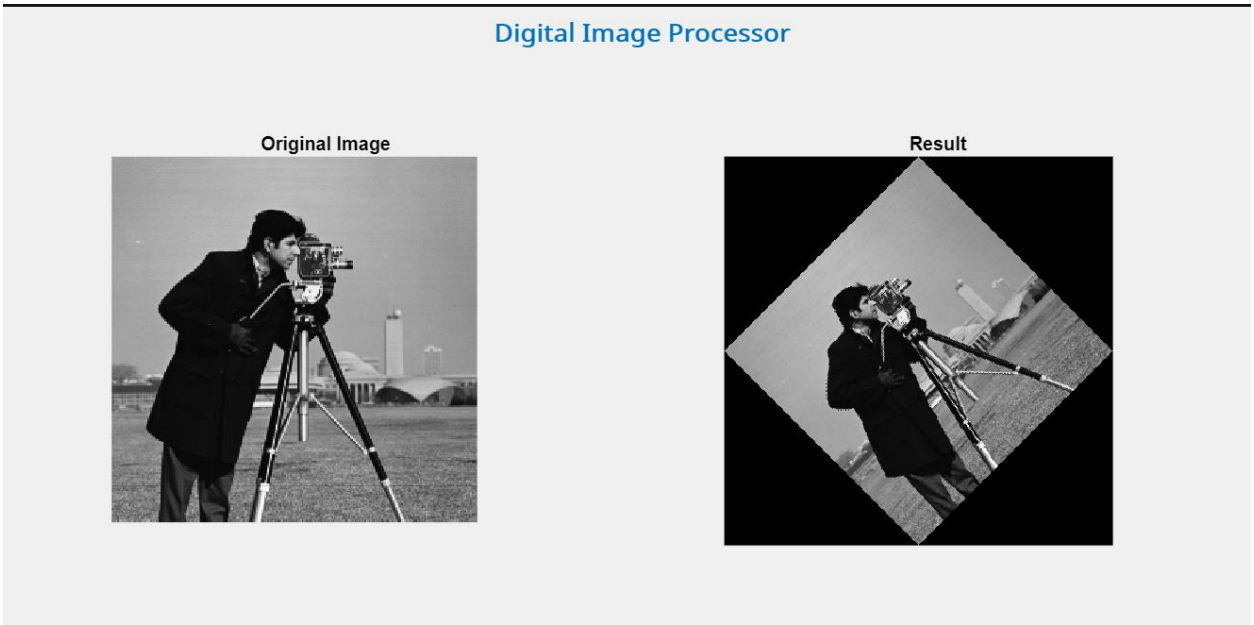
Code:

```
function ResizeImageButtonPushed(app, event)
    global a;
    img = a;
    val = app.DropDown.Value;
    h = app.HeightEditField.Value;
    w = app.WidthEditField.Value;

    if strcmp(val, 'Nearest')
        img = imresize(img, [h,w],'nearest');
    elseif strcmp(val, 'Bilinear')
        img = imresize(img, [h,w],'bilinear');
    else
        img = imresize(img, [h,w],'bicubic');
    end

    imshow(img, 'parent', app.UIAxes_2);
end
```

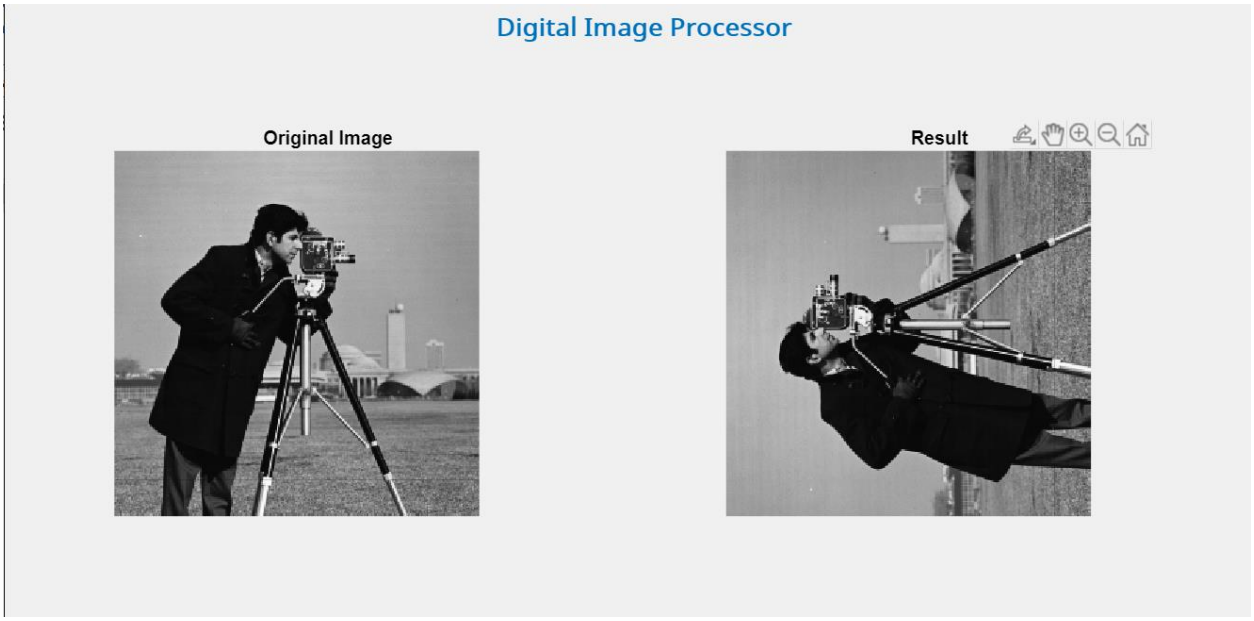
Rotation:

The user can also perform rotation under the interpolation tab. The user has the option of specifying the method of interpolation as well as the degrees of rotation.

Output:



With 90° degrees:



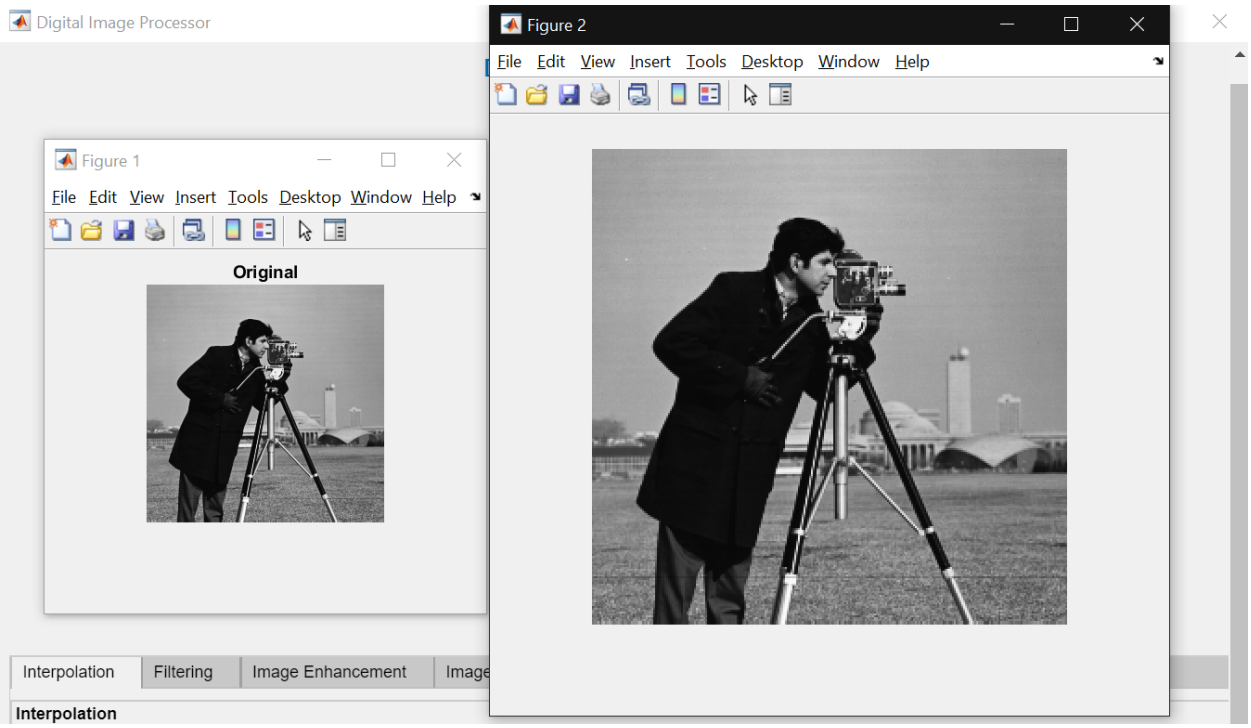
Code:

```
function RotateButtonPushed(app, event)
    global a;
    img = a;
    val = app.DropDown_2.Value;
    deg = app.DegreesEditField.Value;

    if strcmp(val, 'Nearest')
        img = imrotate(img, deg,'nearest');
    elseif strcmp(val, 'Bilinear')
        img = imrotate(img, deg,'bilinear');
    else
        img = imrotate(img, deg,'bicubic');
    end

    imshow(img, 'parent', app.UIAxes_2);
end
```

Zooming:

The user can zoom the image to have a better look at the image.

Output:



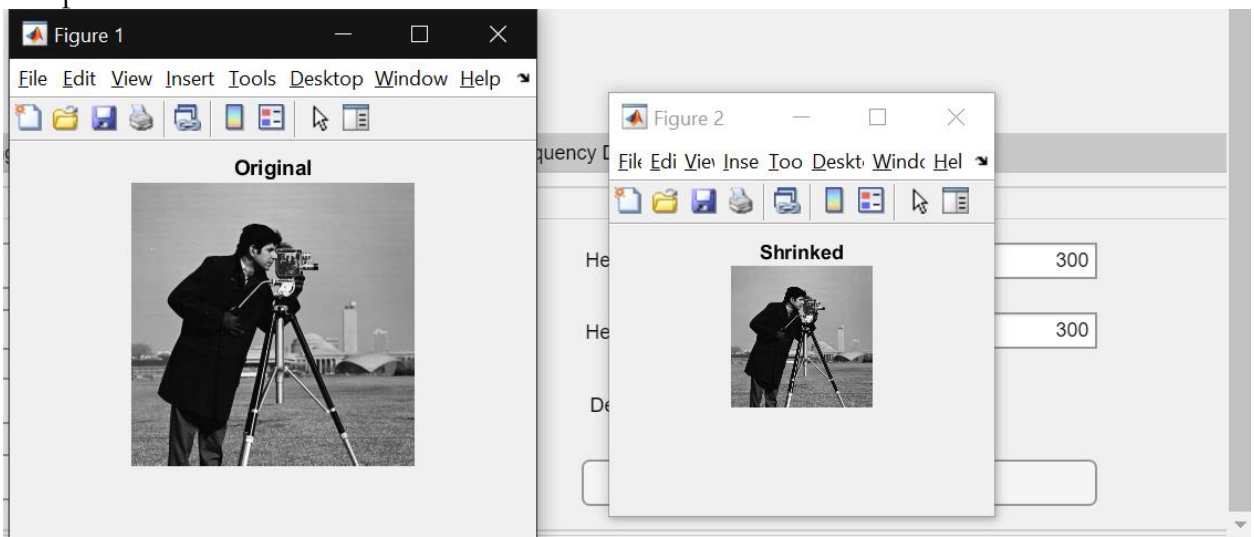
Code:

```
function ZoomByFactorof2ButtonPushed(app, event)
    global a;
    img = a;
    [m, n] = size(img);
    for i = 1:2*m
        for j = 1:2*n
            p = i - floor(i / 2);
            q = j - floor(j / 2);

            b(i, j) = a(p, q);
        end
    end
    figure;imshow(a)
    title('Original')

    figure;imshow(b);
end
```

Shrinking:

The user has an option to shrink the image.

Output:

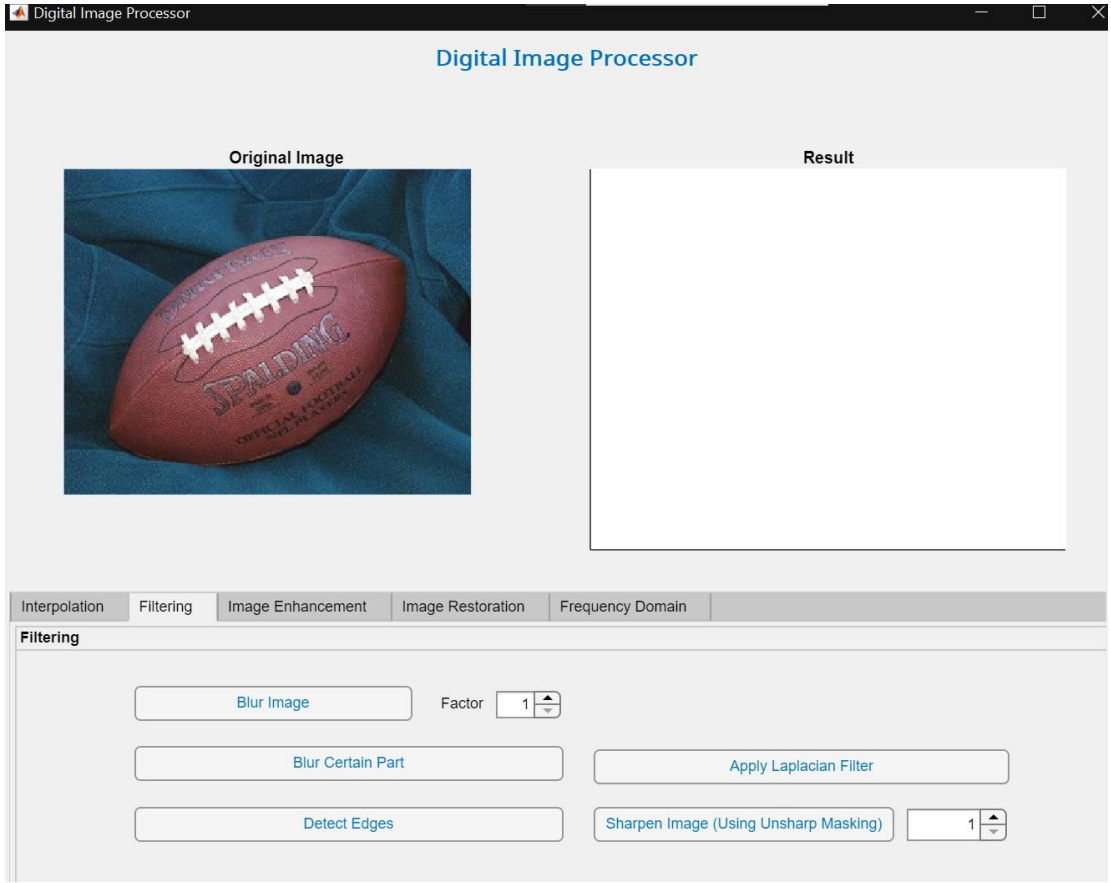


Code:

```
function ShrinkByFactorof2ButtonPushed(app, event)
    global a;
    img = a;
    factor = 2;
    s = size(img);
    s1 = s/factor;
    k=1;
    l=1;
    for i=1:s1
        for j=1:s1
            d(i,j)=img(k,l);
            l=l+factor;
        end
        l=1;
        k=k+factor;
    end
    figure
    imshow(img);
    title('Original')
    figure(2)
    imshow(d);
    title('Shrinked')
end
```

Task 2: Filtering

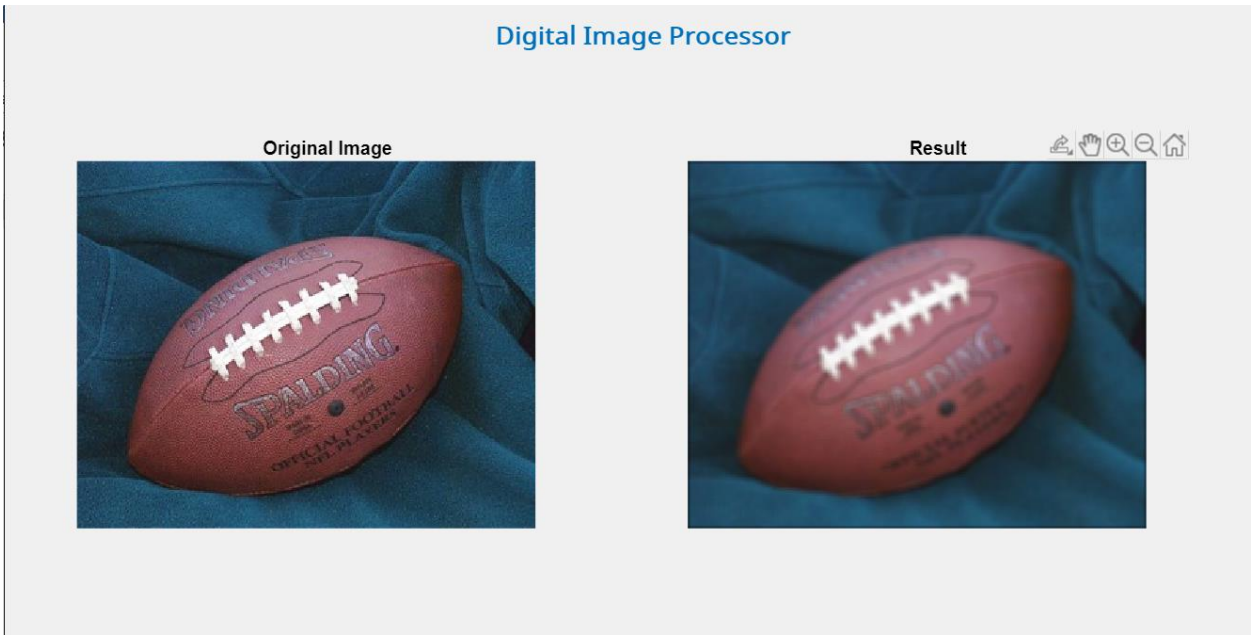
Following is the initial interface of the filtering tab. There are mainly 5 operations available to the user in this tab. Let us go through each of them and demonstrate their outputs.



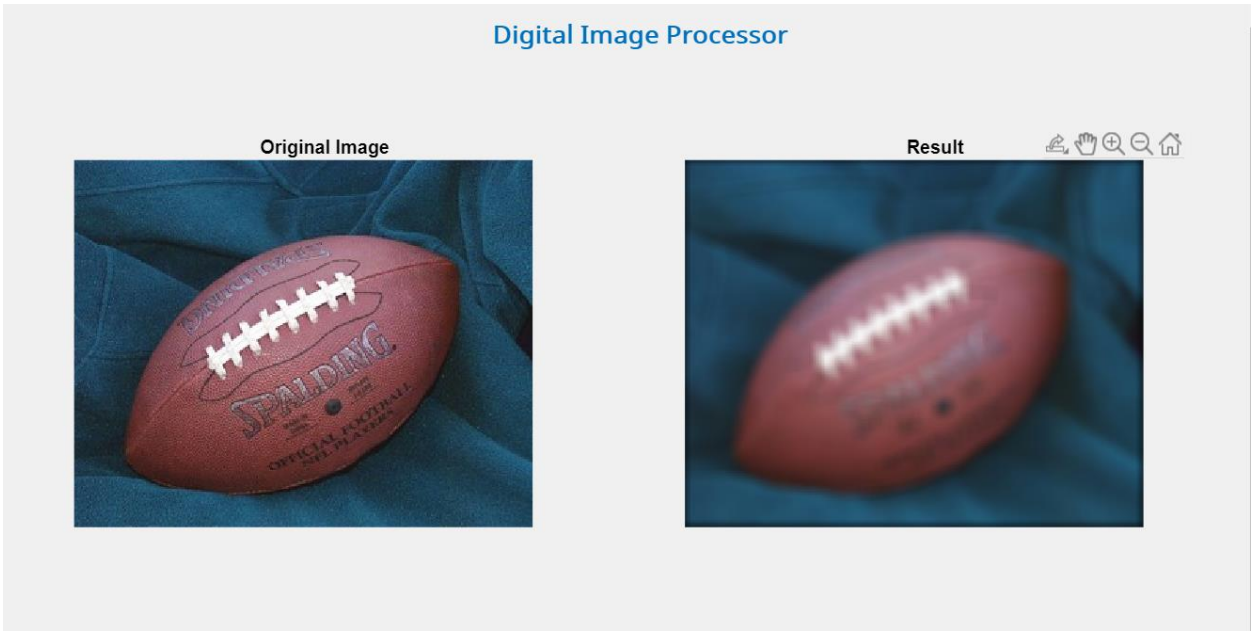
Blurring:

The user has the option to blur an image by providing a factor. The higher the factor, the higher the blurring effect in the image will be produced.

Output:



Output with factor = 10:



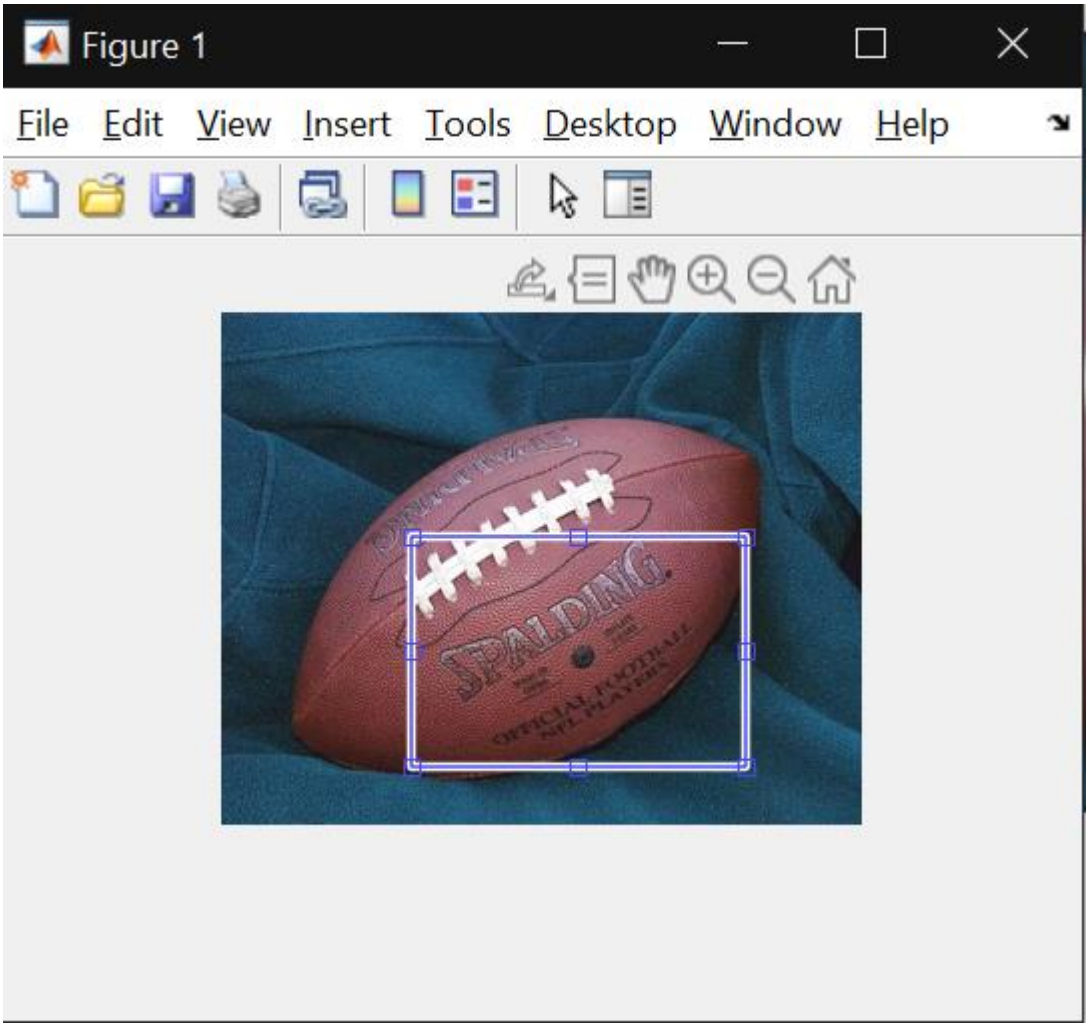
Code:

```
function BlurImageButtonPushed(app, event)
    global filt_img;
    m = filt_img;
    val = app.FactorSpinner_3.Value;
    ag = fspecial('average', [val, val]);
    g = imfilter(m, ag);
    imshow(g, 'parent', app.UIAxes_2);
end
```

Blurring A Specific Part of The Image:

Aside from blurring the whole image, the user has the option to blur the image except for the specific portion of the image that the user selects.

Output:



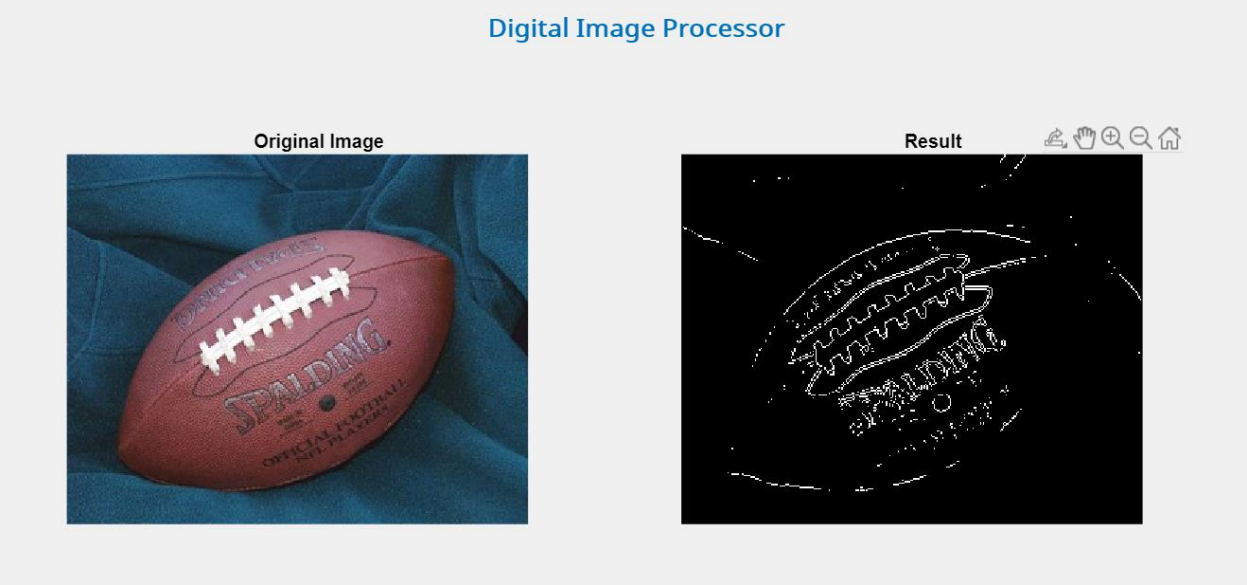
Code:

```
function BlurCertainPartButtonPushed(app, event)
    global filt_img;
    warning off;
    img = filt_img;
    imshow(img);
    [j,rect] = imcrop(img);
    ag = fspecial('average', [20,20]);
    g = imfilter(img, ag);
    g( rect(2):rect(2)+rect(4),rect(1):rect(1)+rect(3), :)=j;
    imshow(g);
end
```

Detect Edges:

The user has the option to detect edges in an image, this is achieved using the 'Sobel' operator.

Output:



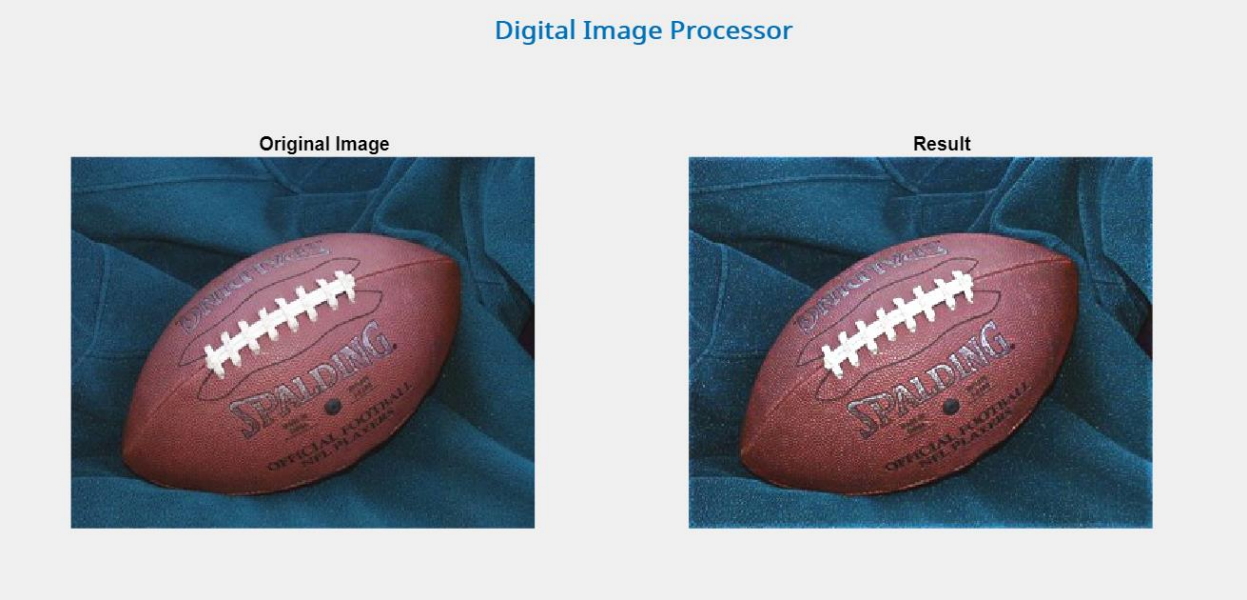
Code:

```
function DetectEdgesButtonPushed(app, event)
    global filt_img;
    e = im2gray(filt_img);
    w=edge(e, 'sobel');
    imshow(w, 'parent', app.UIAxes_2);
end
```

Sharpening:

The user can sharpen an image by specifying the factor just as with the blur option.

Output:

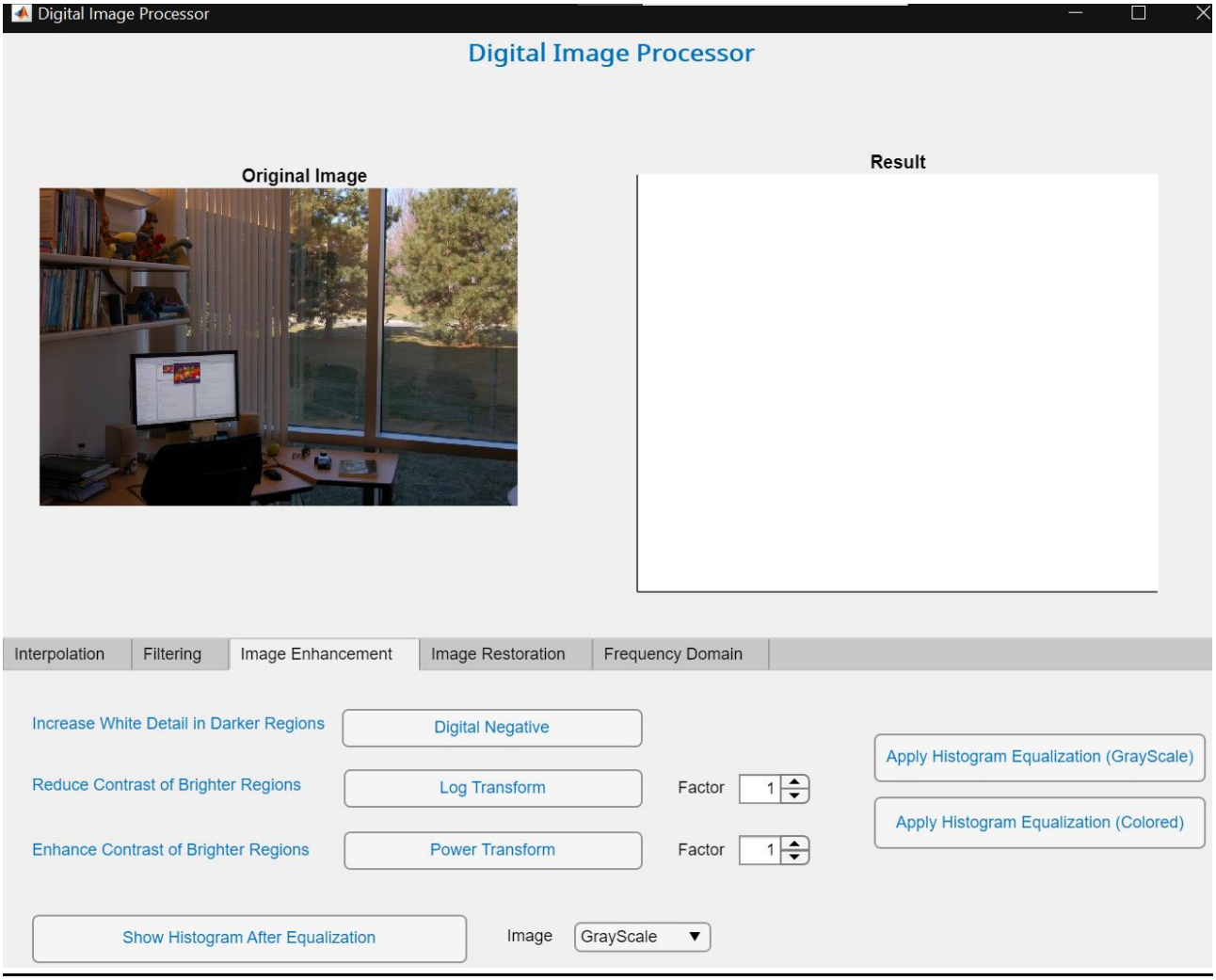


Code:

```
function SharpenImageUsingUnsharpMaskingButtonPushed(app, event)
    %highpass filter
    %those pixels having highest intensities will pass, and low
    %intensity ones will be attenuated
    global filt_img;
    img = filt_img;
    val = app.Spinner_3.Value;
    ag = fspecial('average', [val, val]);
    g = imfilter(img, ag);
    g = imsubtract(img, g);
    x = imadd(img, g);
    imshow(x, 'parent', app.UIAxes_2);
end
```

Task 3: Image Enhancement

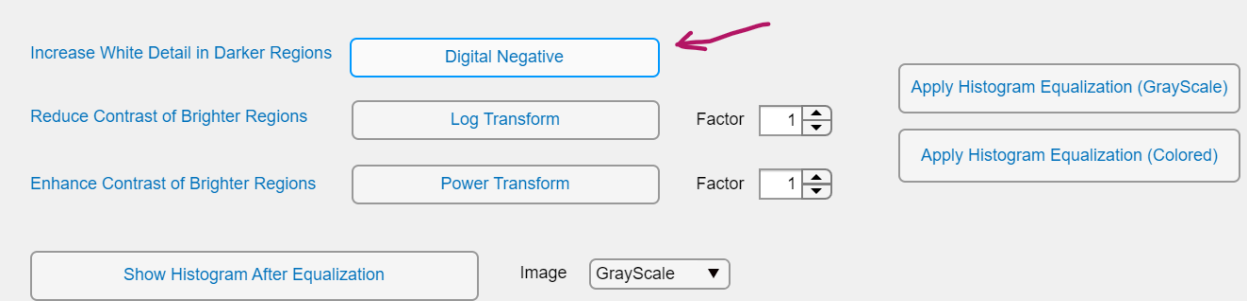
Following is the initial interface of the image enhancement tab.



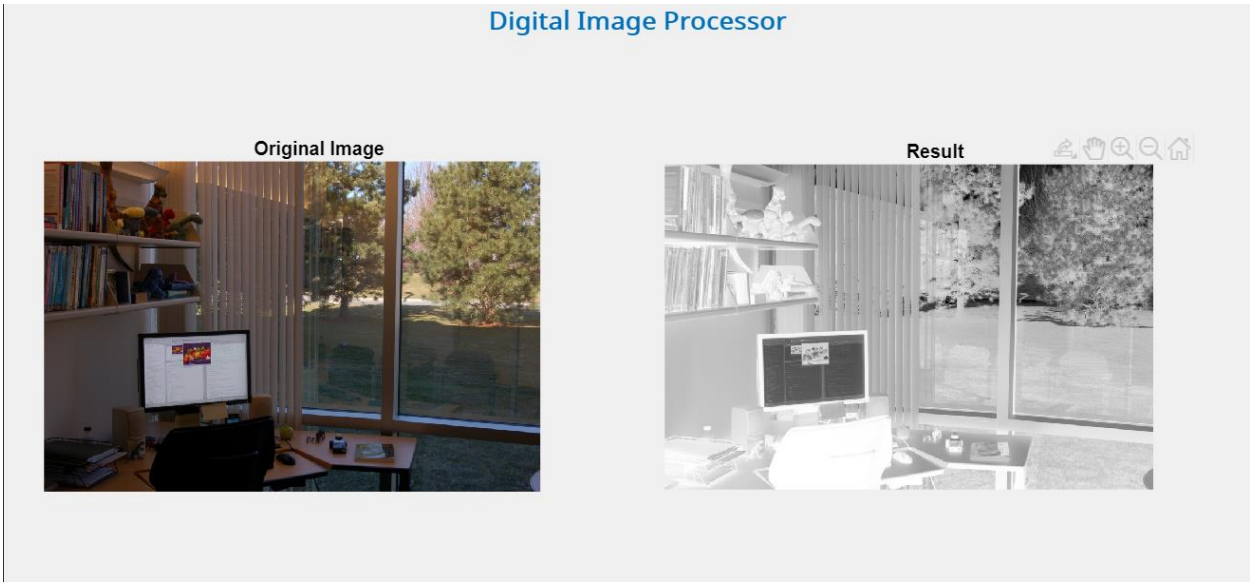
There are 3 linear contrast functions that the user can apply on an image. In addition to the linear contrast functions, the user can enhance the image using histogram equalization. Let us demonstrate the output of each operation.

Digital Negative:

The user can increase white detail in darker regions using the digital negative operation.



Output:

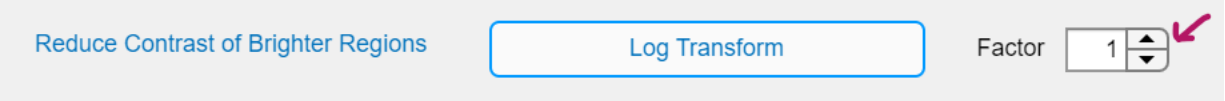


Code:

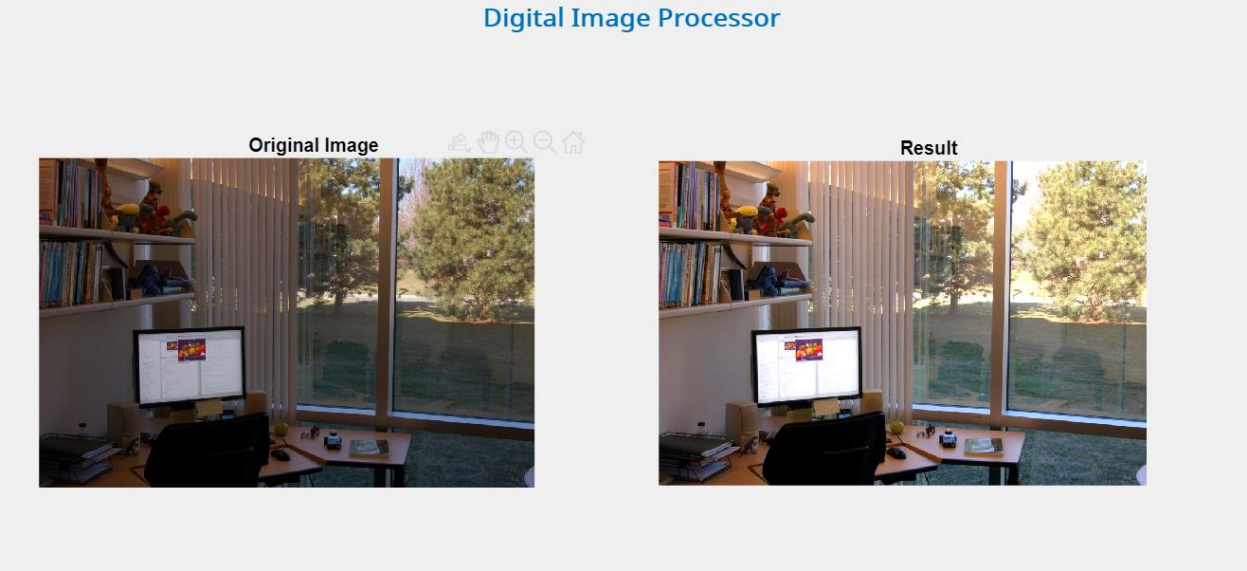
```
    function DigitalNegativeButtonPushed(app, event)
        global ie_img;
        img = ie_img;
        img = rgb2gray(img);
        img1 = 255-img;
        imshow(img1, 'parent', app.UIAxes_2);
    end
```

Log Transform:

The user can reduce the contrast of brighter regions while enhancing the regions with lower contrast. In addition, the user can specify the factor of the log transformation as well.



Output:



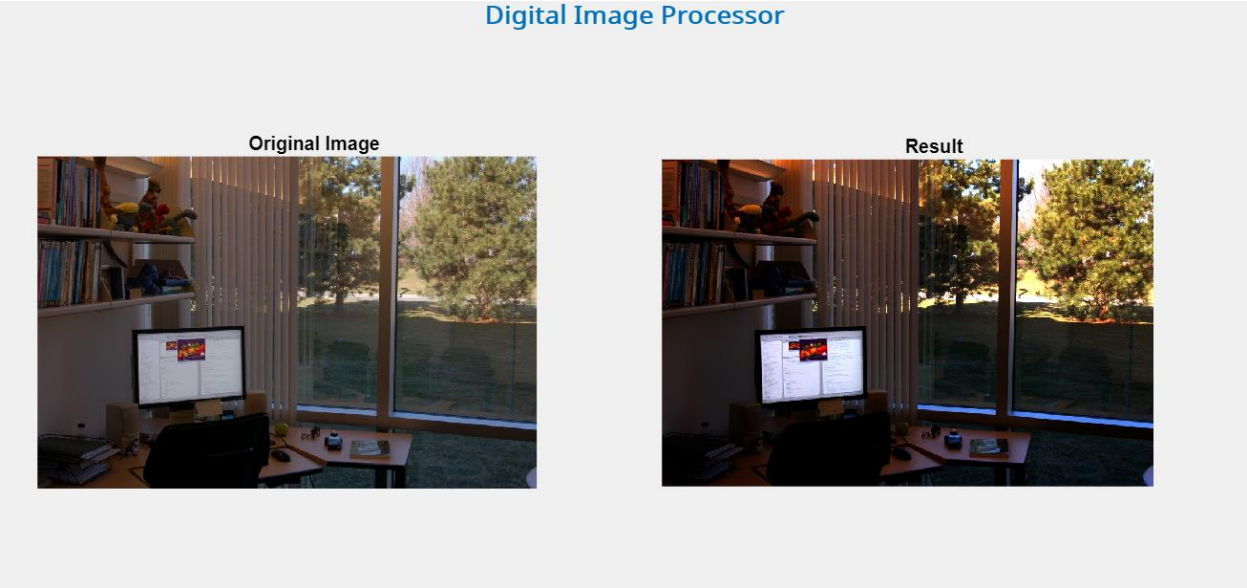
Code:

```
function LogTransformButtonPushed(app, event)
    global ie_img;
    img = ie_img;
    img = im2double(img);
    img_double = img;
    [r, c] = size(img);
    factor = app.FactorSpinner.Value;
    for i = 1:r
        for j = 1:c
            img_double(i, j) = factor * log(1 + img(i, j));
        end
    end
    imshow(img_double, 'parent', app.UIAxes_2);
end
```

Power Transform:

The user can enhance the contrast of brighter regions while compressing the darker regions.

Output:



Code:

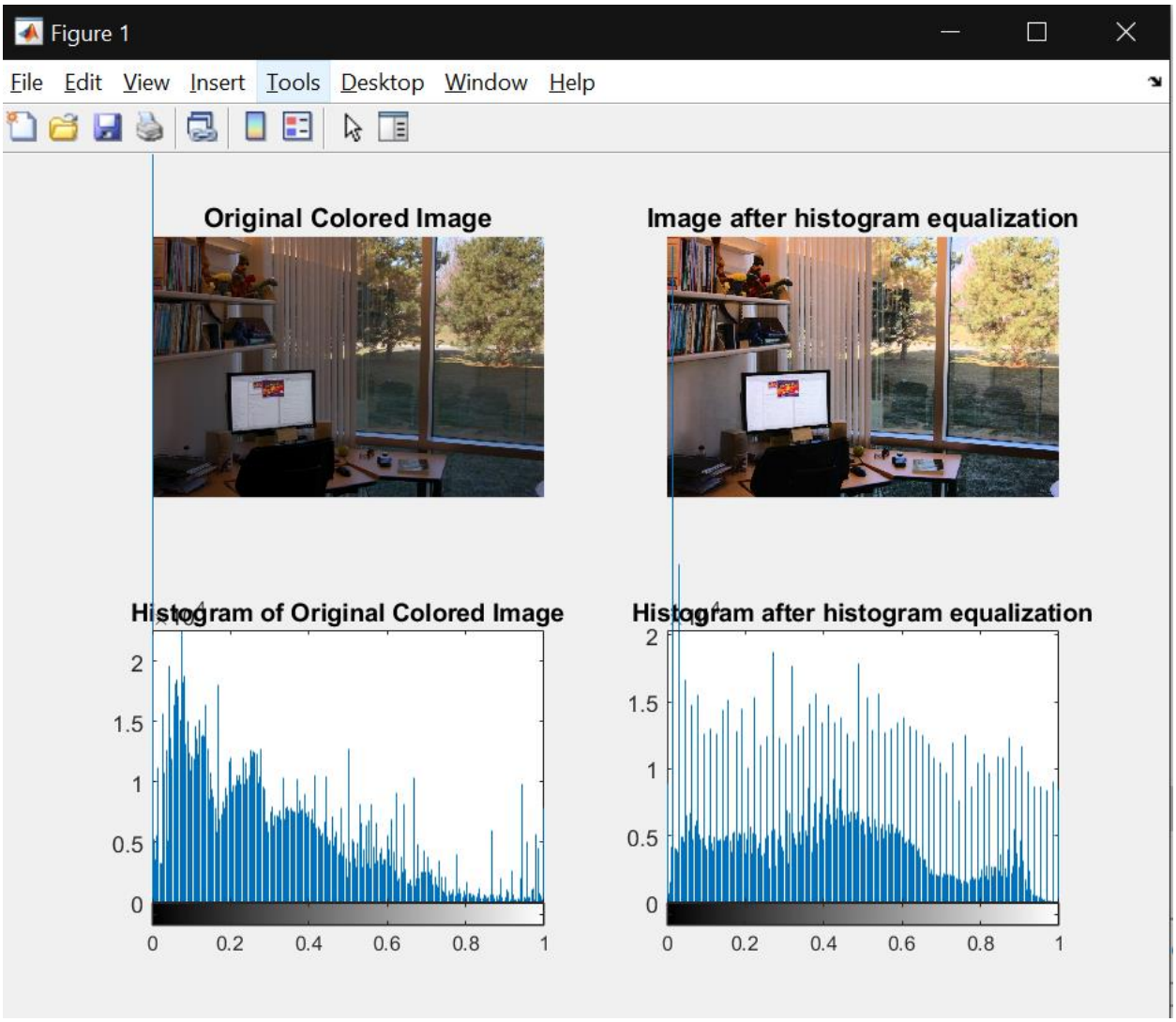
```
function PowerTransformButtonPushed(app, event)
    global ie_img;
    img = ie_img;
    img = im2double(img);
    img_double = img;
    [r, c] = size(img);
    factor = app.FactorSpinner_2.Value;
    for i = 1:r
        for j = 1:c
            img_double(i, j) = factor * img(i, j) ^ 2;
        end
    end
    imshow(img_double, 'parent', app.UIAxes_2);
end
```

Histogram Equalization:

The user can adjust the contrast of an image using the histogram equalization technique. It spreads out the most frequent intensities.

Output:

Digital Image Processor

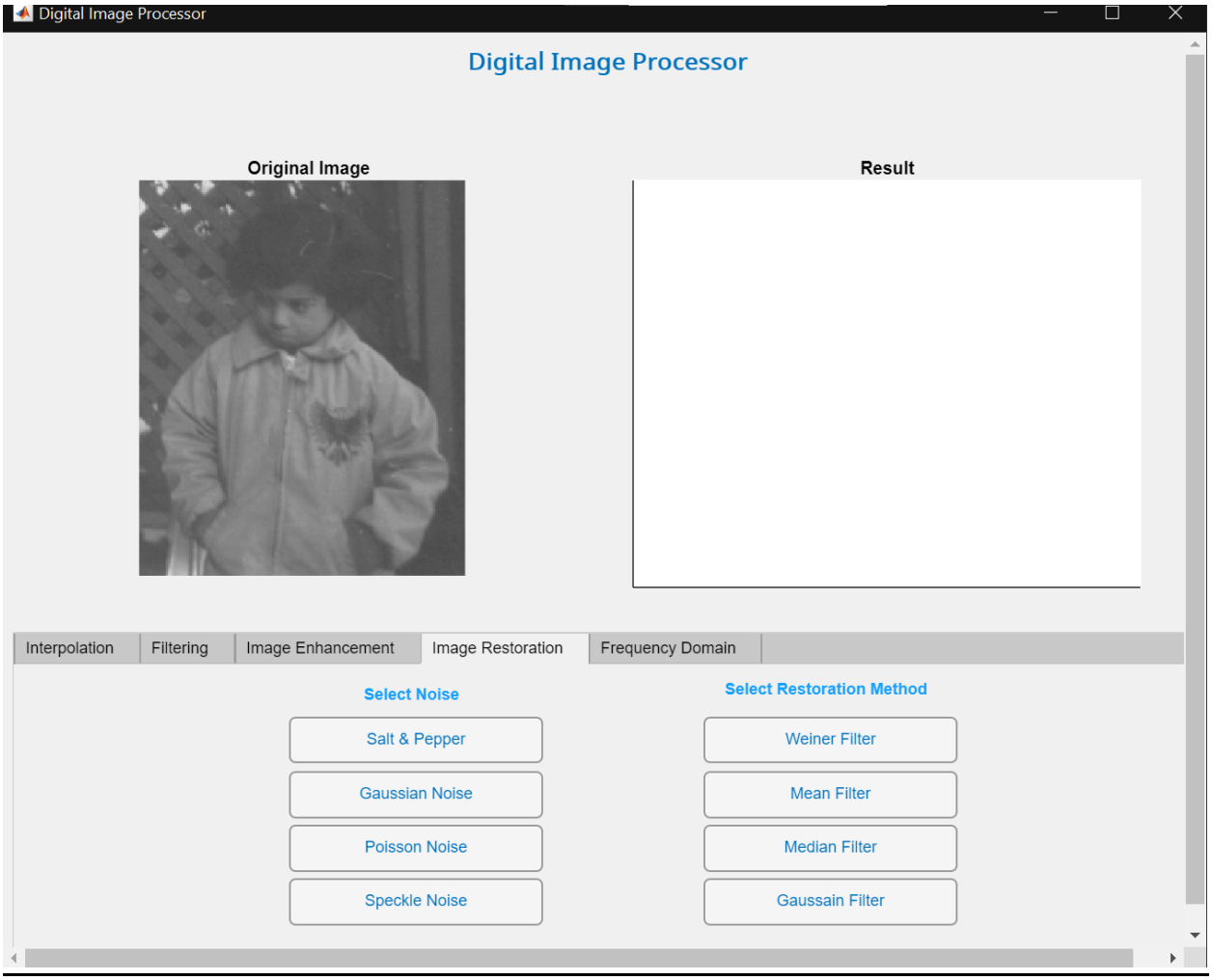


Code:

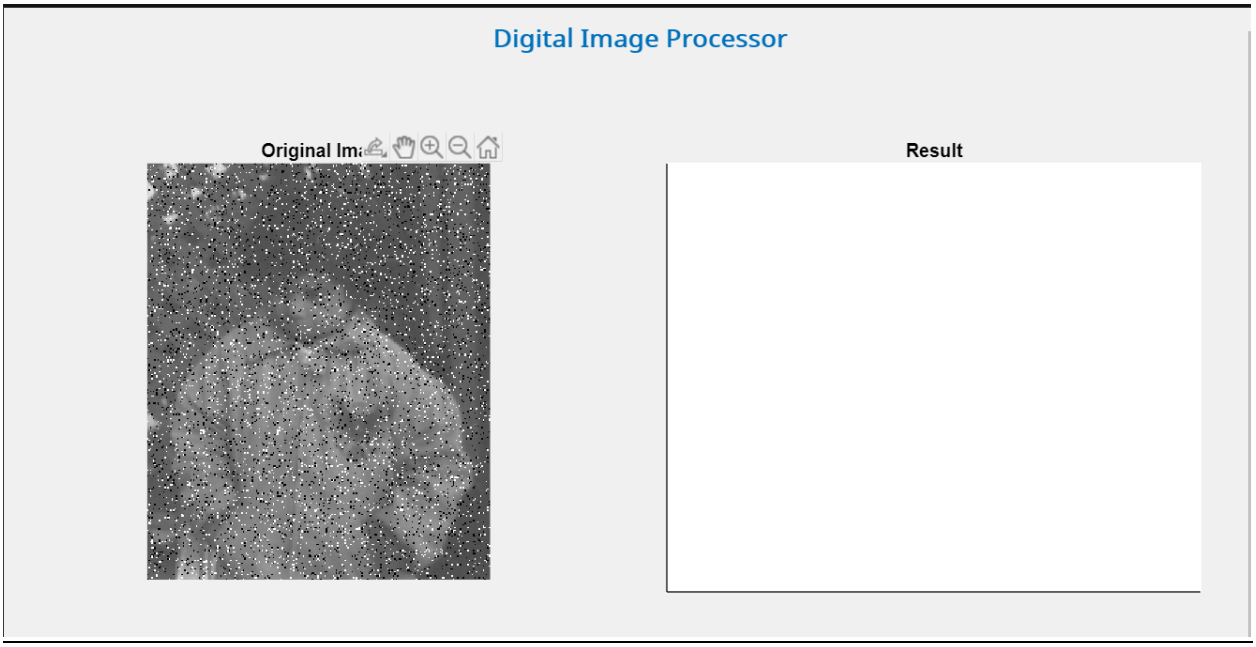
```
function ApplyHistogramEqualizationColoredButtonPushed(app, event)
    global ie_img;
    img = ie_img;
    img = rgb2hsv(img);
    img(:,:,3) = histeq(img(:,:,3));
    rs = hsv2rgb(img);

    imshow(rs, 'parent', app.UIAxes_2);
end
```

Task 4: Image Restoration

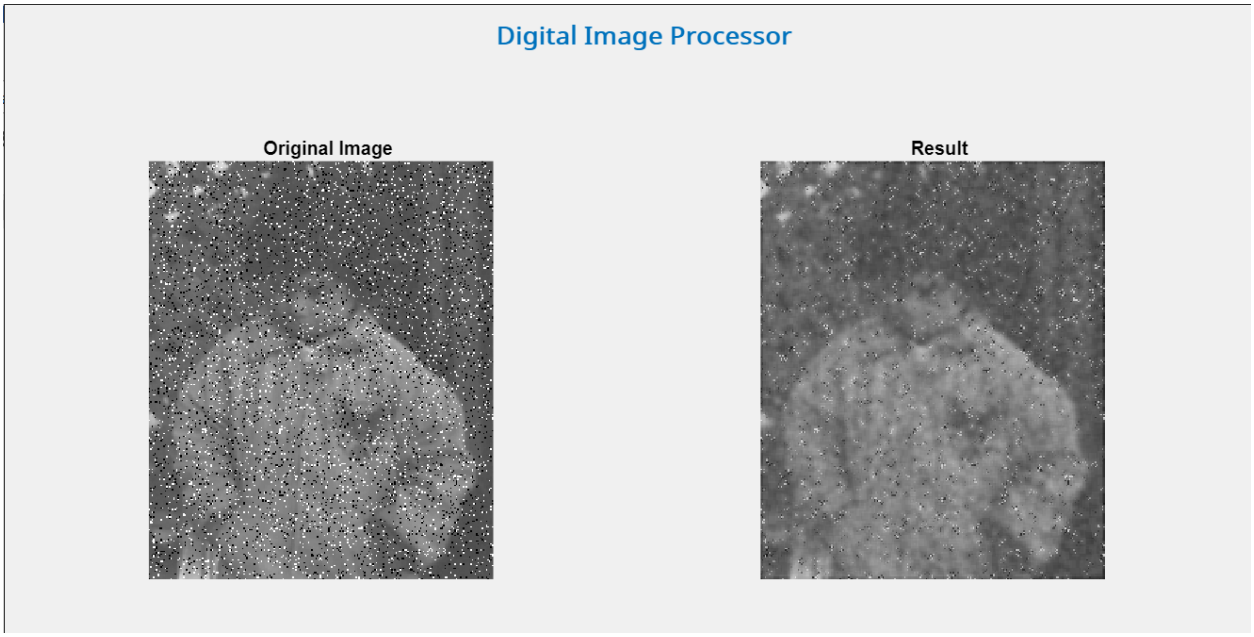
Following is the initial interface of the image restoration tab. The user can add the specific noise and try to restore the image by applying different filters provided on the right-hand side.



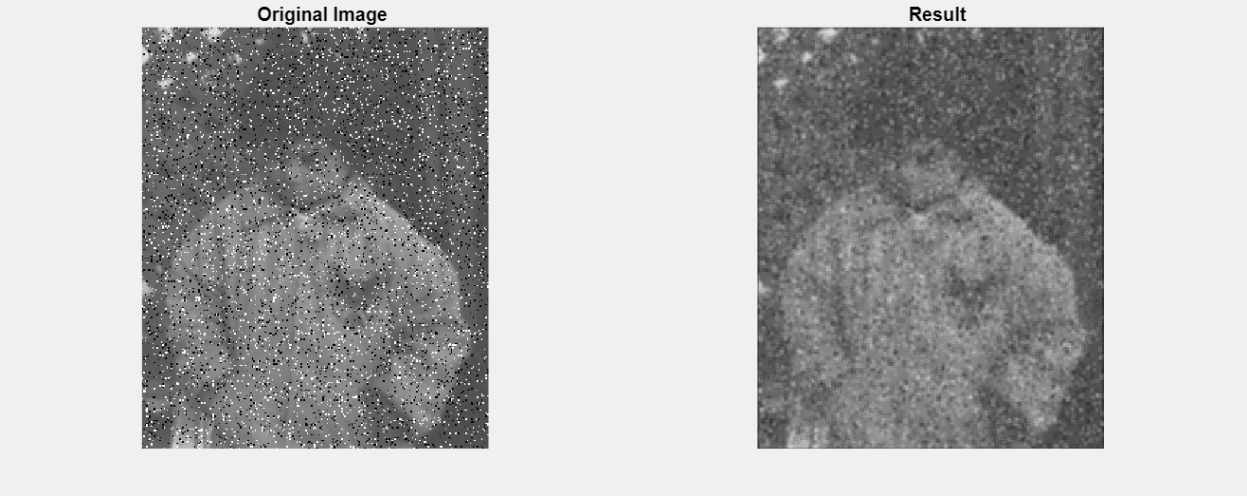
Salt and Pepper Noise:



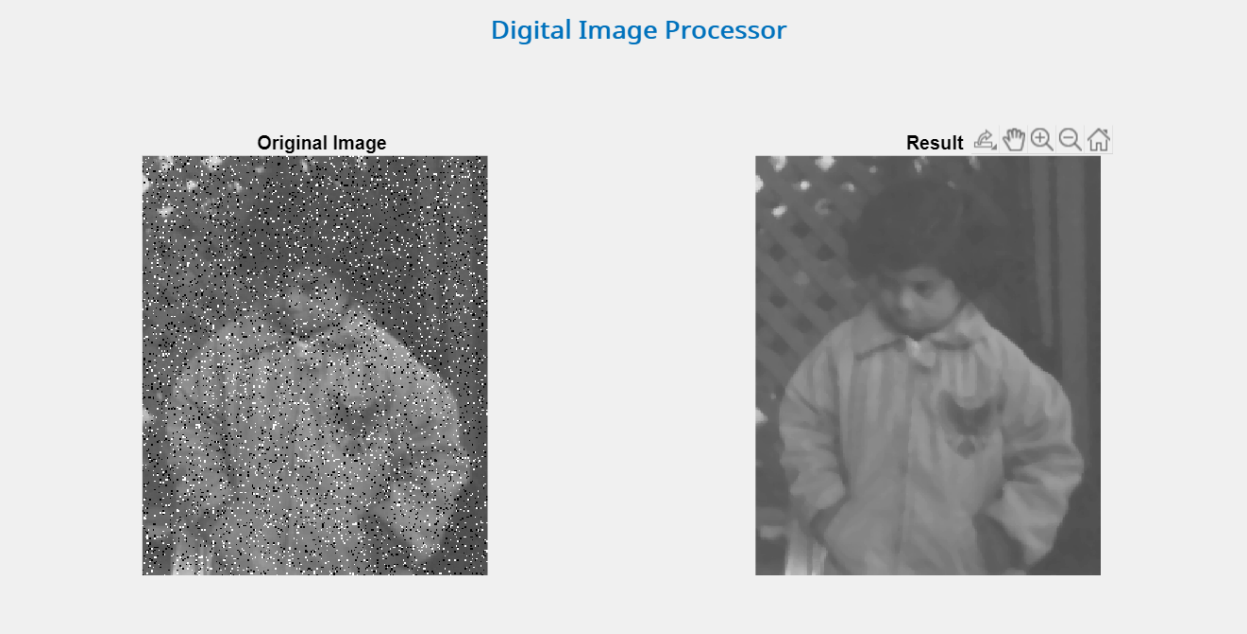
Removal of the noise using **Weiner filter**:



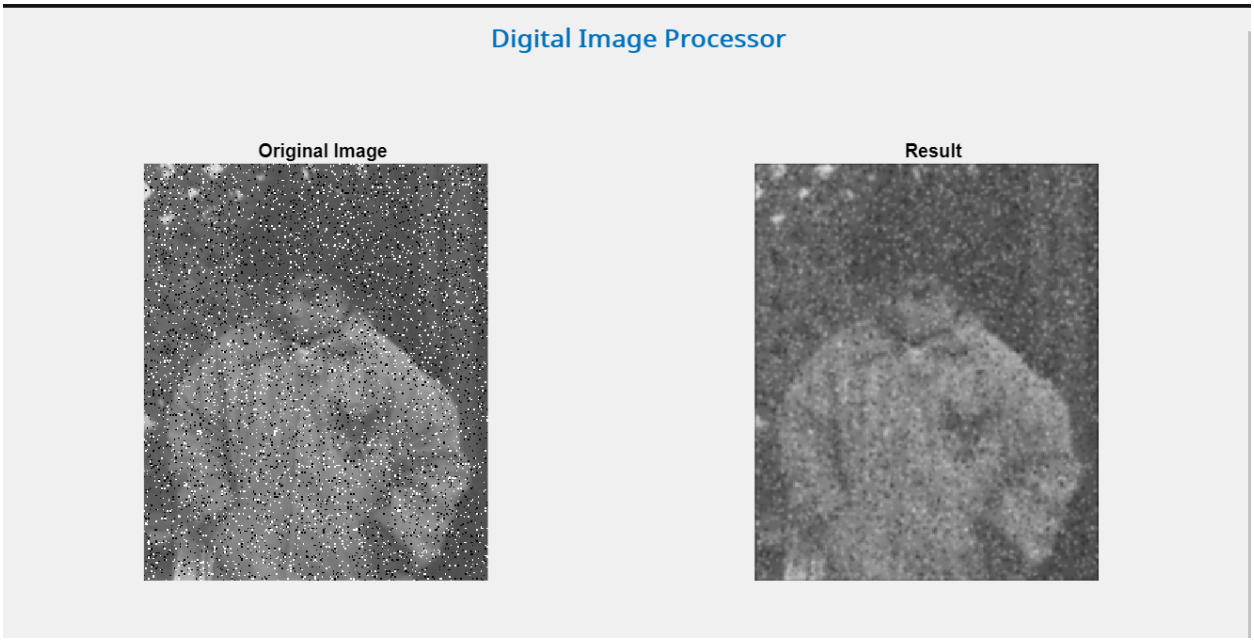
Removal of the salt and pepper noise using **Mean filter**:



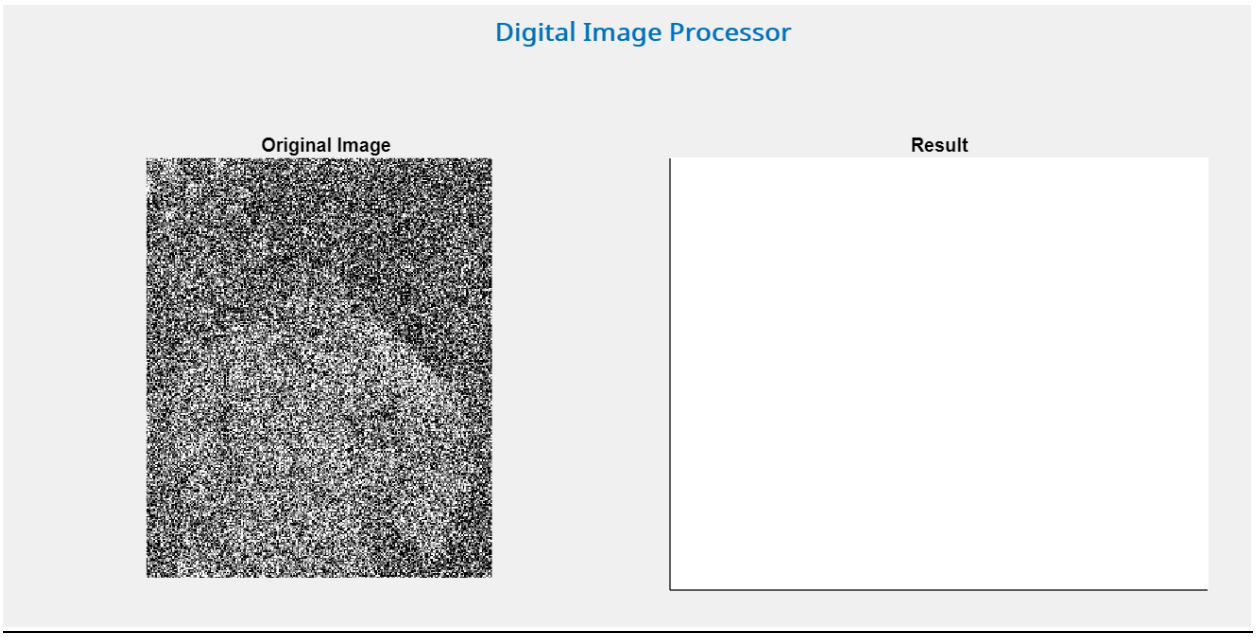
Removal of salt and pepper using **Median filter**:



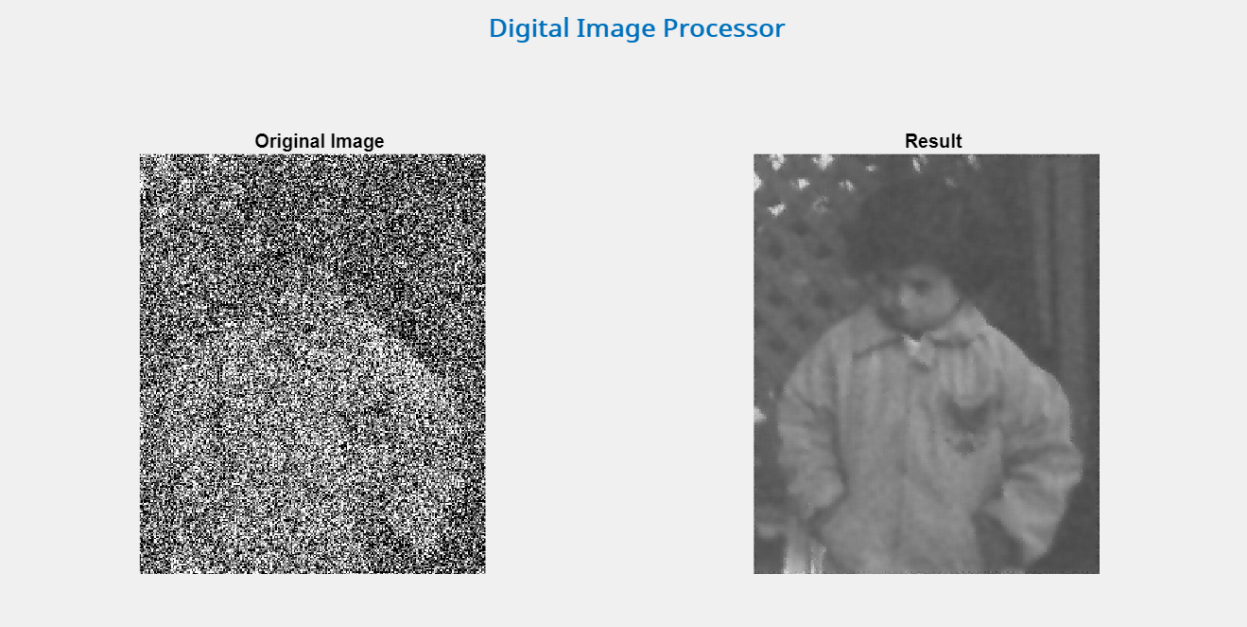
Removal of the salt and pepper using **Gaussian filter**:



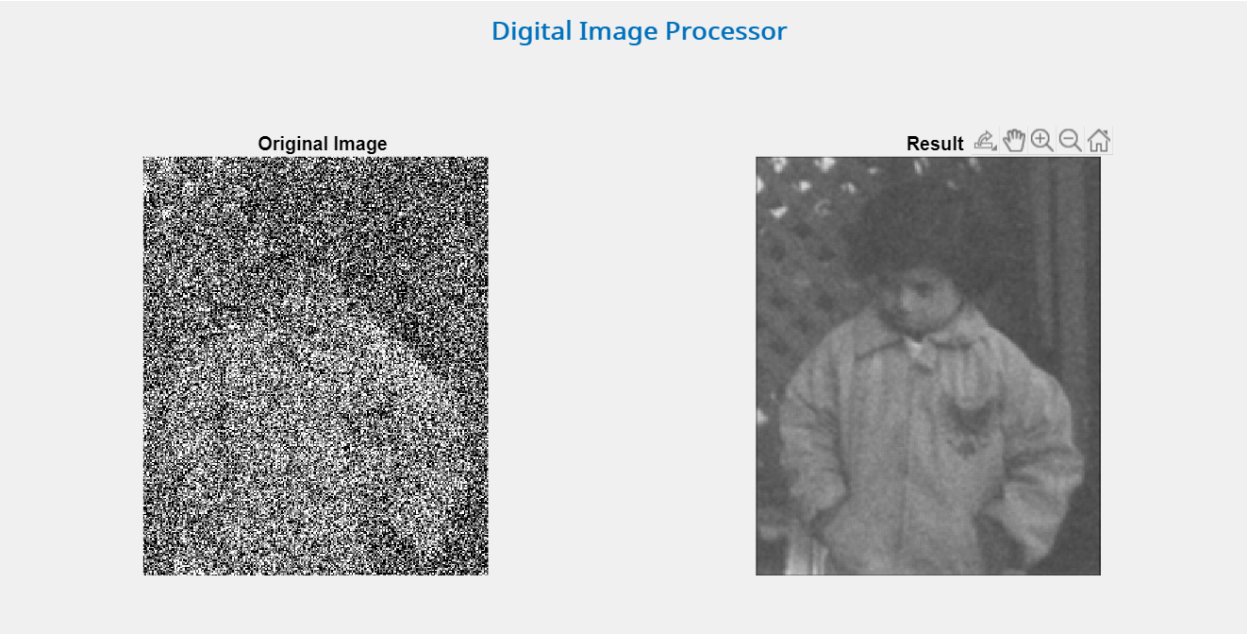
Gaussian Noise:



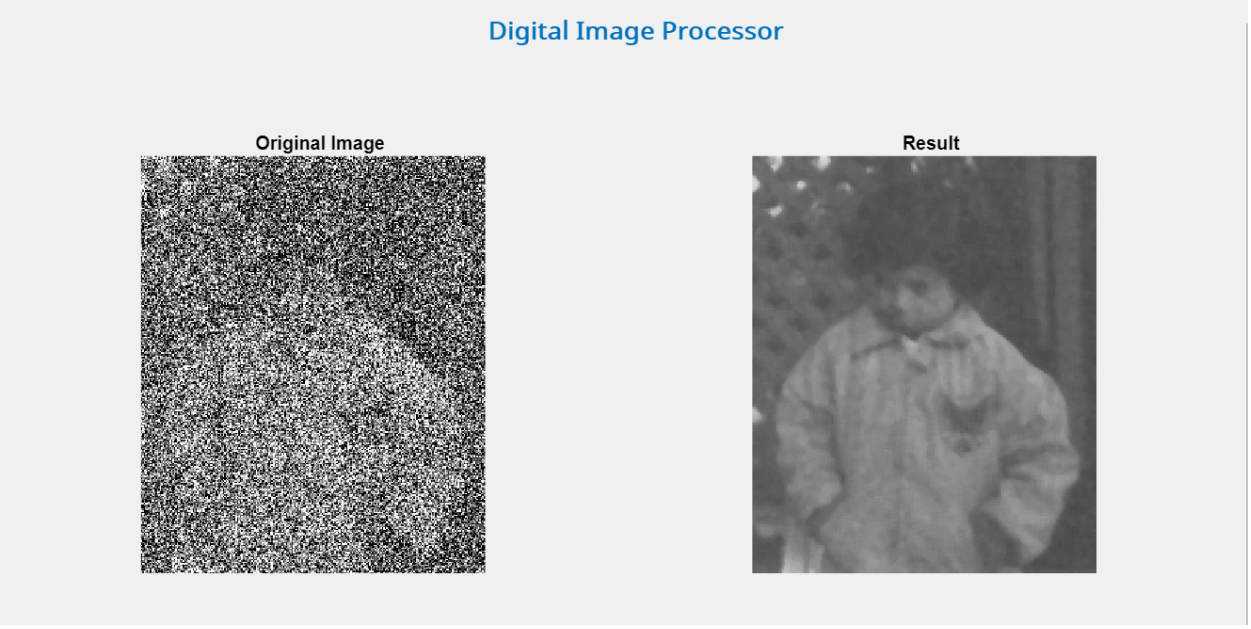
Removal of the gaussian noise using **Weiner filter:**



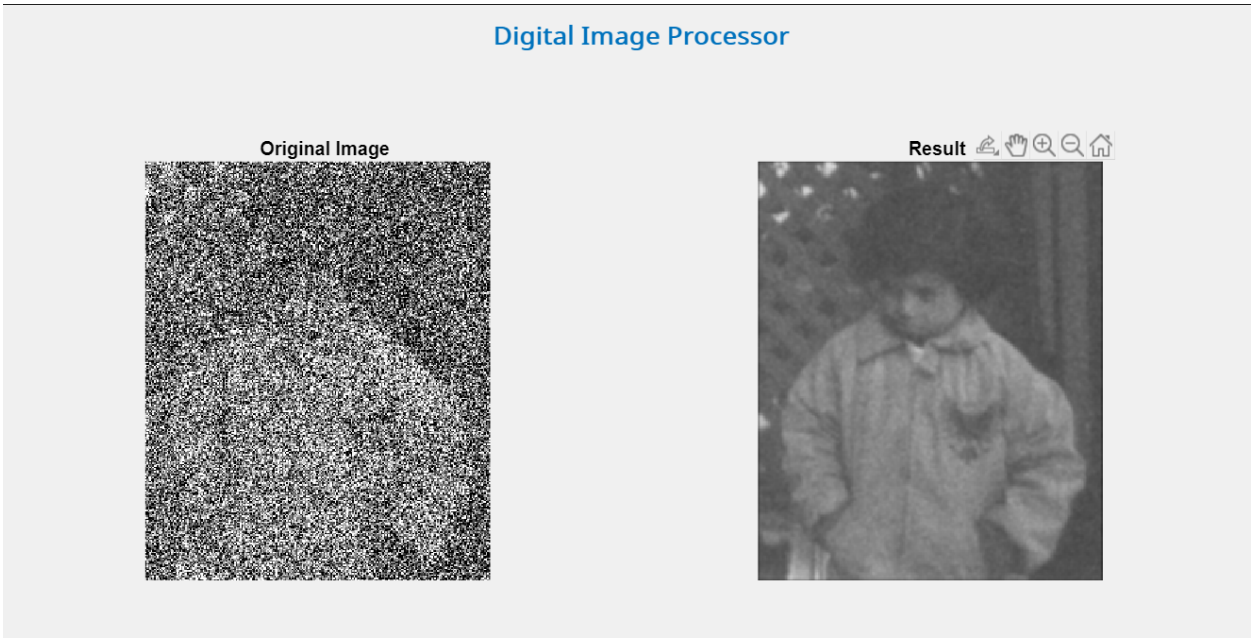
Removal of the gaussian noise using **Mean filter**:



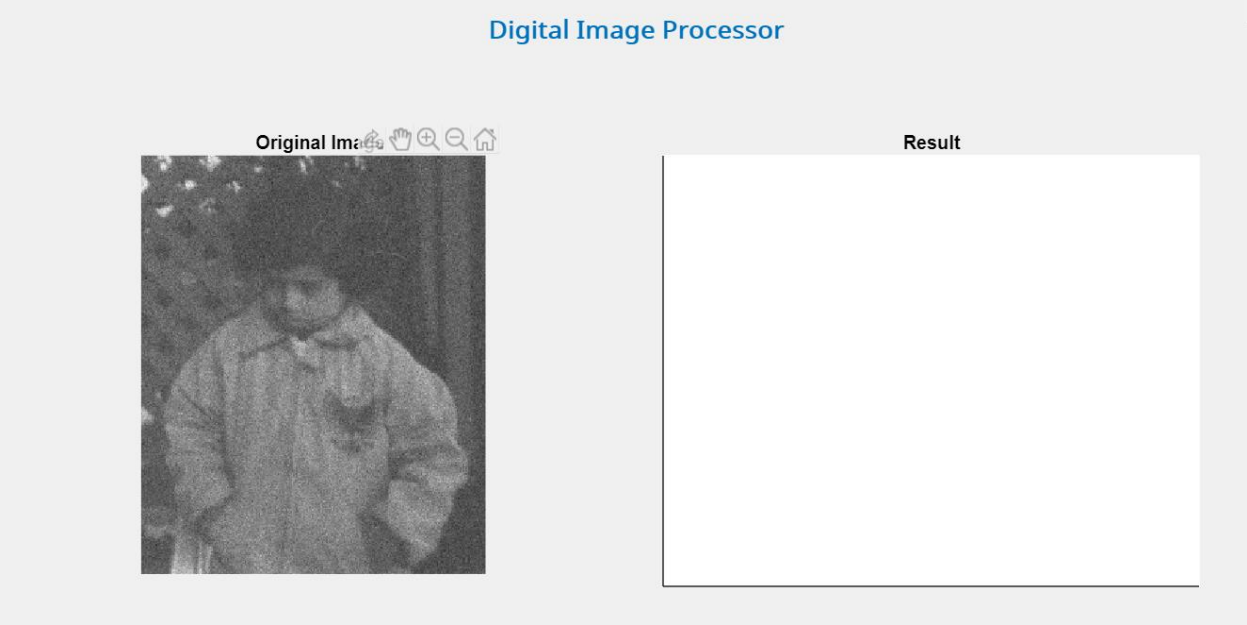
Removal of the gaussian noise using **Median filter**:



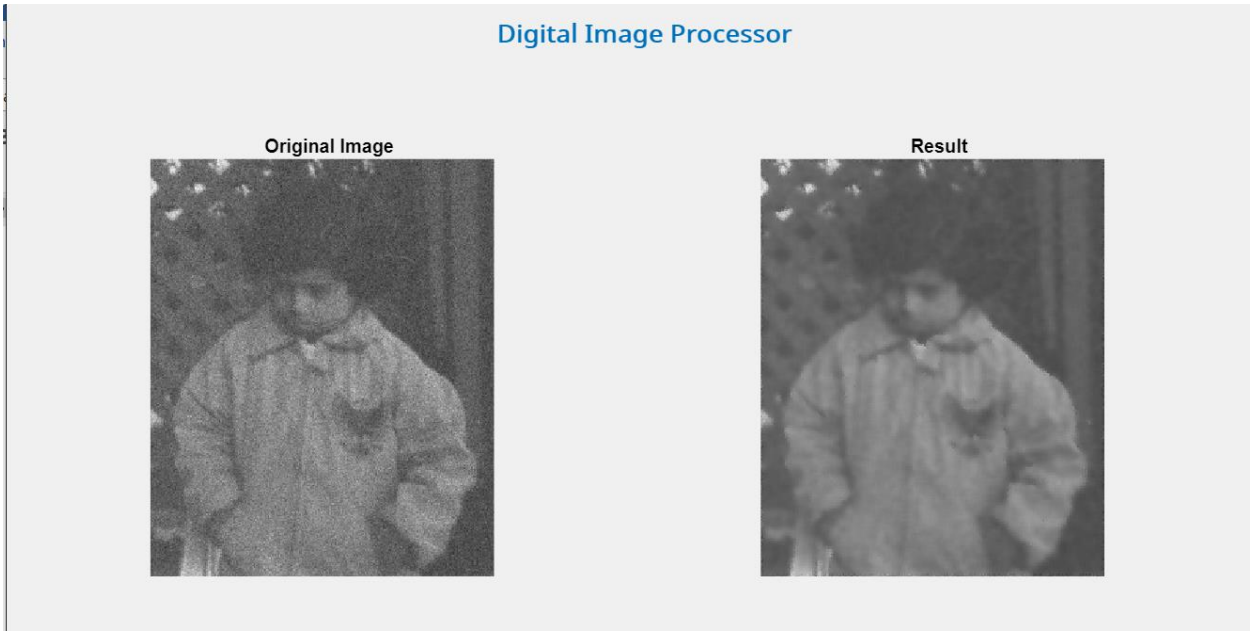
Removal of the gaussian noise using **Gaussian filter**:



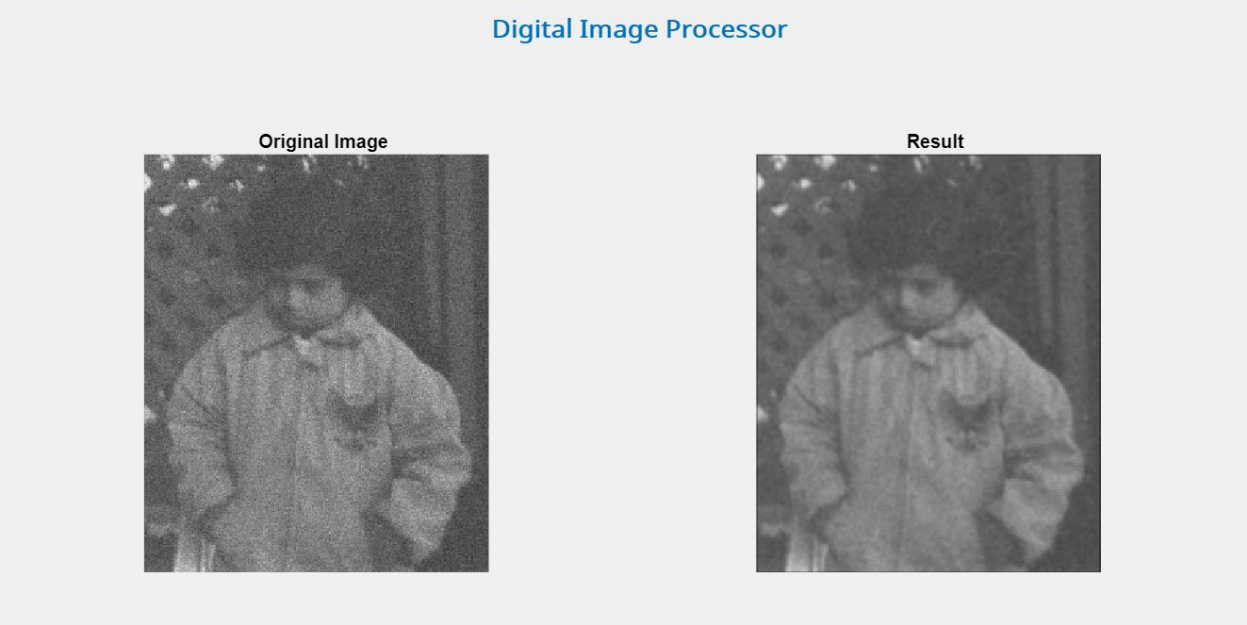
Poisson Noise:



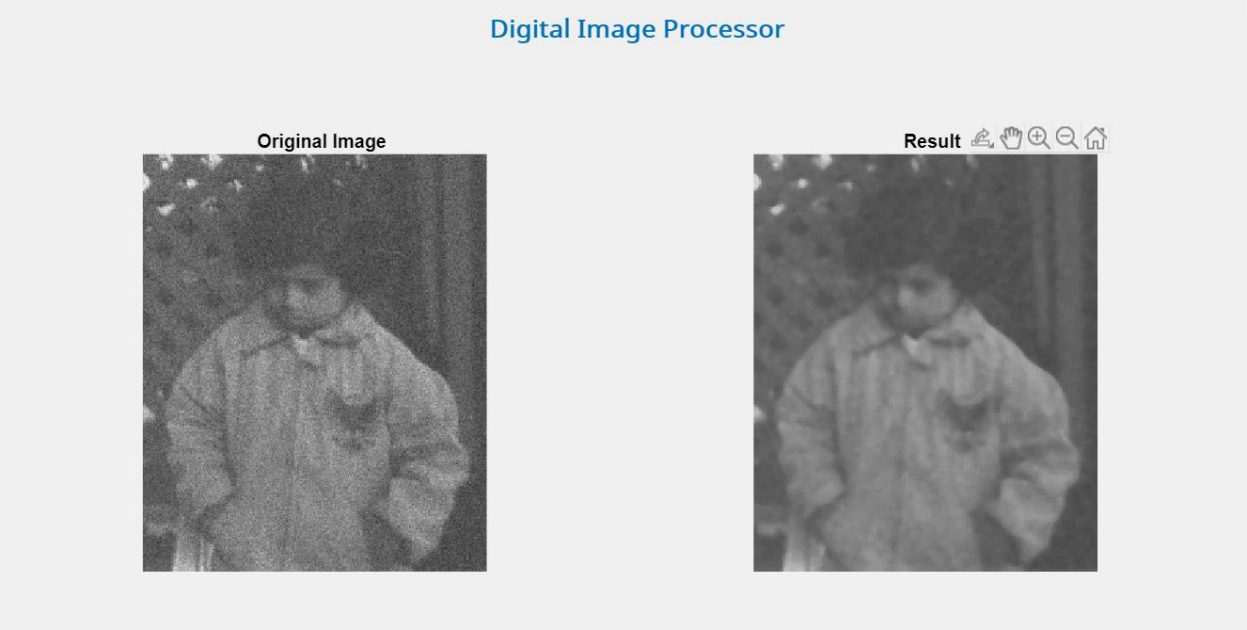
Removal of the Poisson noise using the **Weiner filter**:



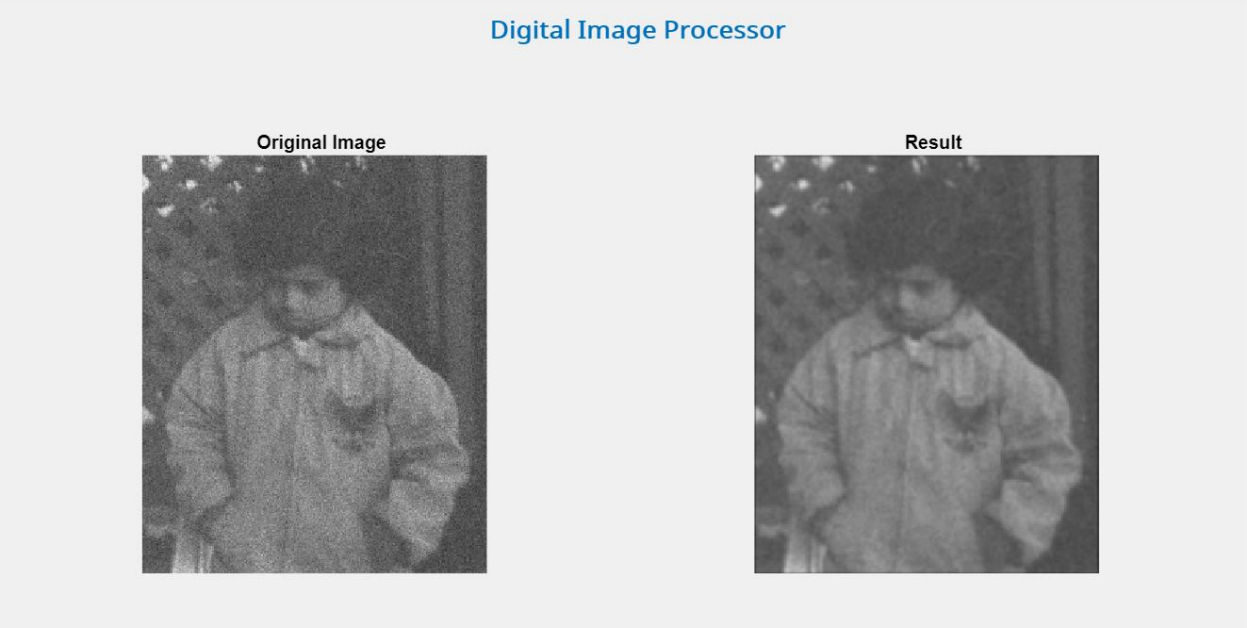
Removal of the Poisson noise using the **Mean filter**:



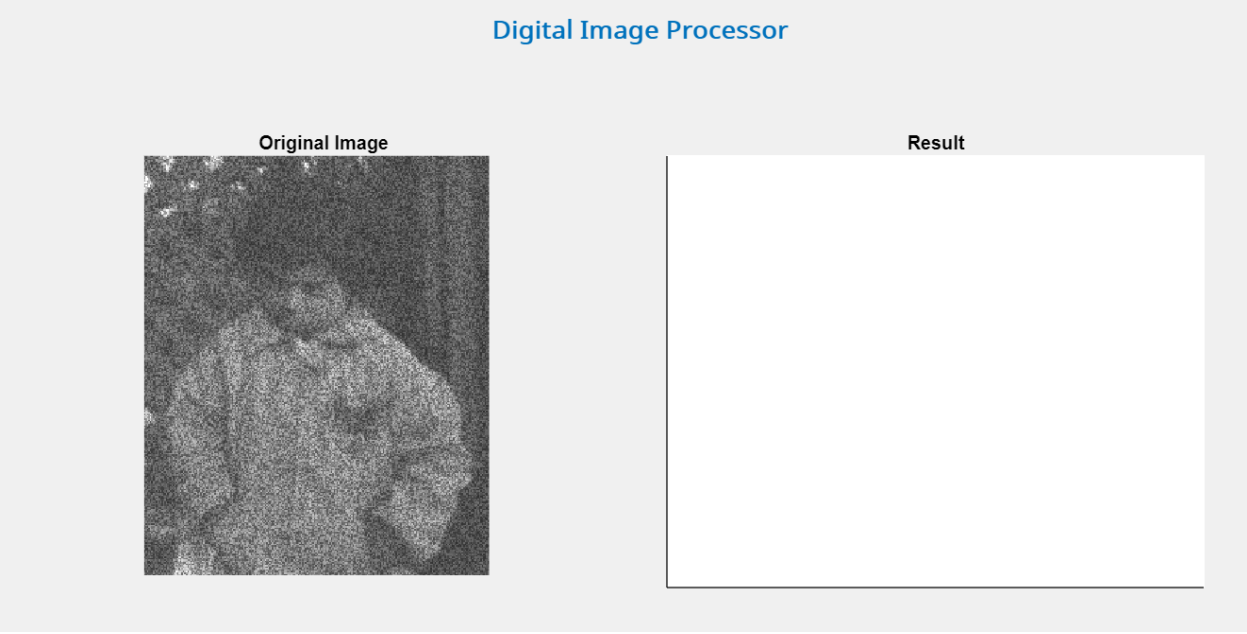
Removal of the Poisson noise using the **Median filter**:



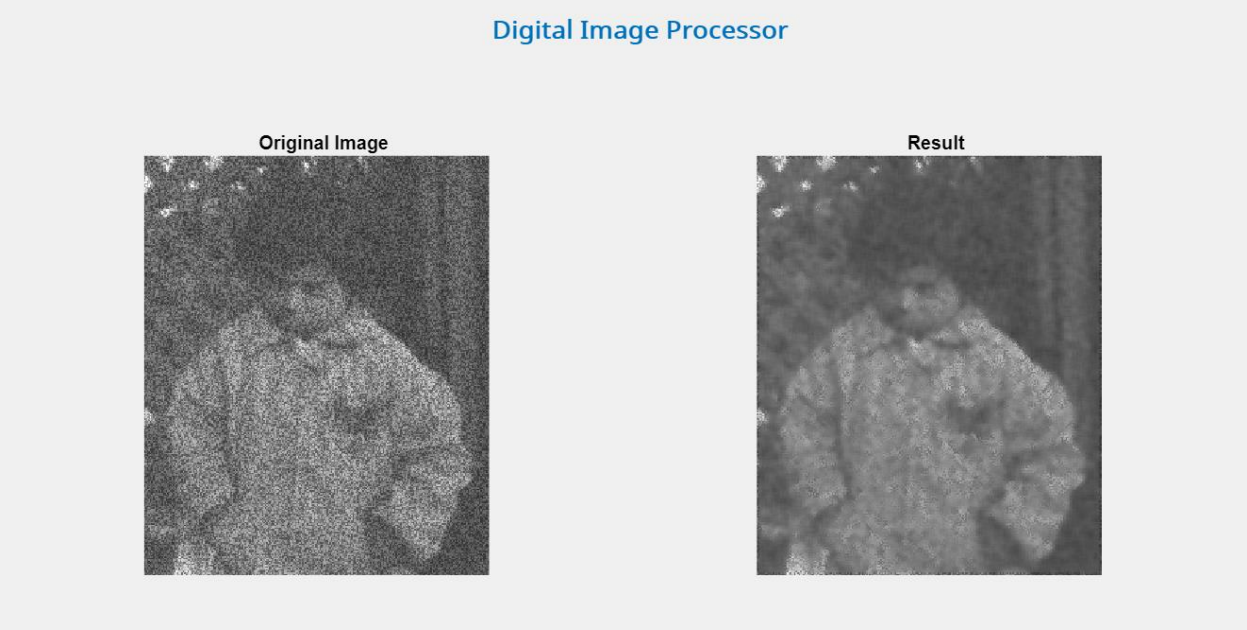
Removal of the Poisson noise using the **Gaussian filter**:



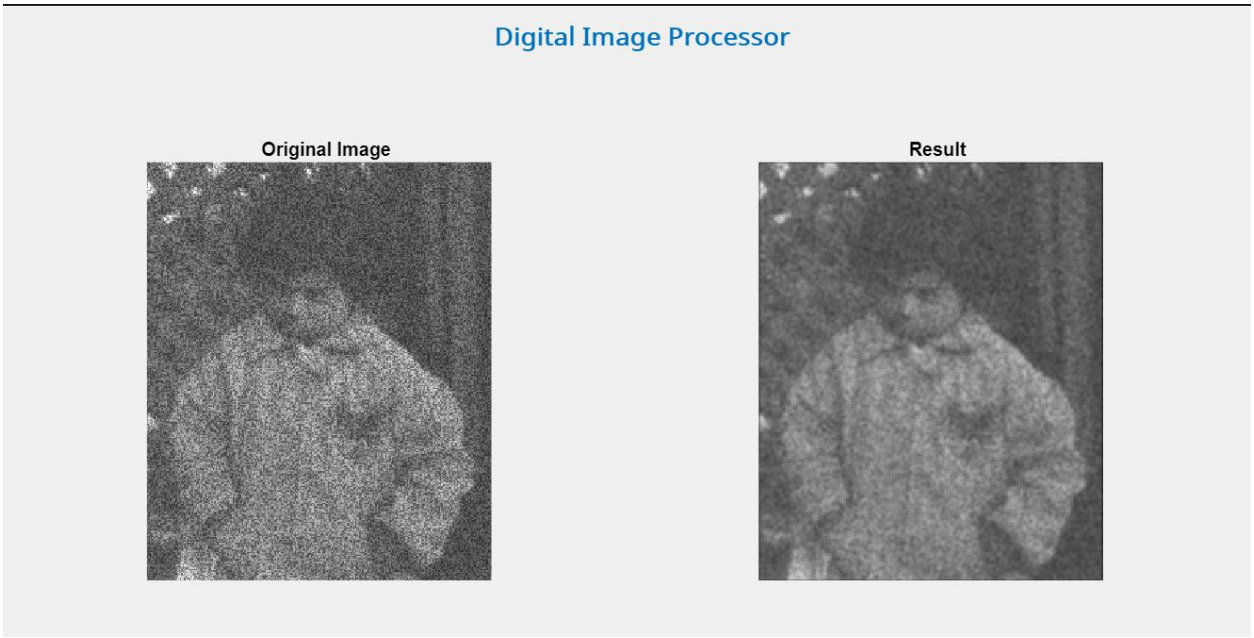
Speckle Noise:



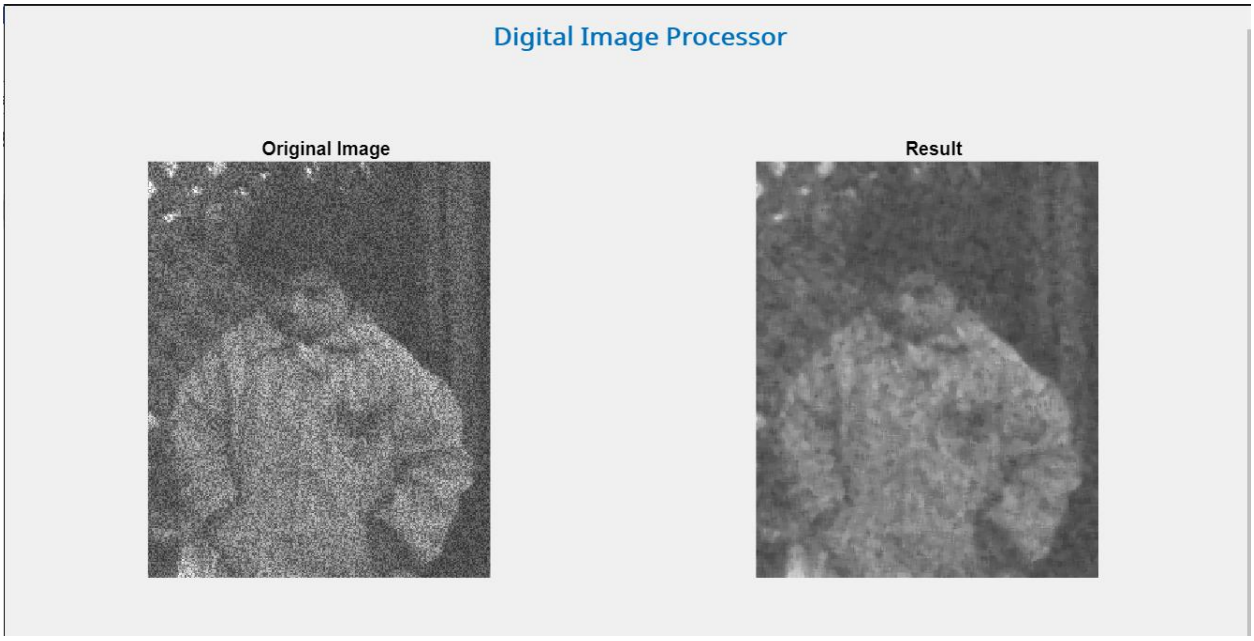
Removal of the Speckle noise using the **Weiner filter**:



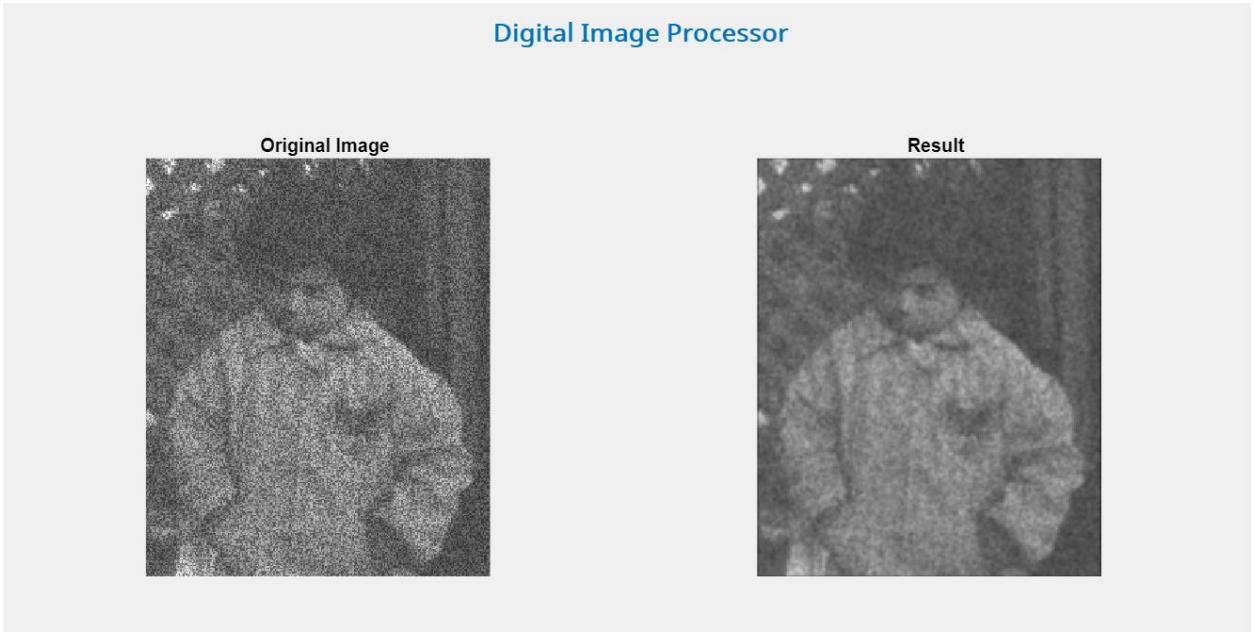
Removal of the Speckle noise using the **Mean filter**:



Removal of the Speckle noise using the **Median filter**:



Removal of the Speckle noise using the **Gaussian filter**:



Following are the codes.

Code for the Weiner Filter:

```
function WeinerFilterButtonPushed(app, event)
    global noise_img;
    w = wiener2(noise_img, [5, 5]);
    imshow(w, 'parent', app.UIAxes_2);
end
```

Code for the Mean Filter:

```
function MeanFilterButtonPushed(app, event)
    global noise_img;
    img = noise_img;
    mf = ones(3, 3) / 9;
    noise_free = imfilter(img, mf);
    imshow(noise_free, 'parent', app.UIAxes_2);
end
```

Code for the Median Filter:

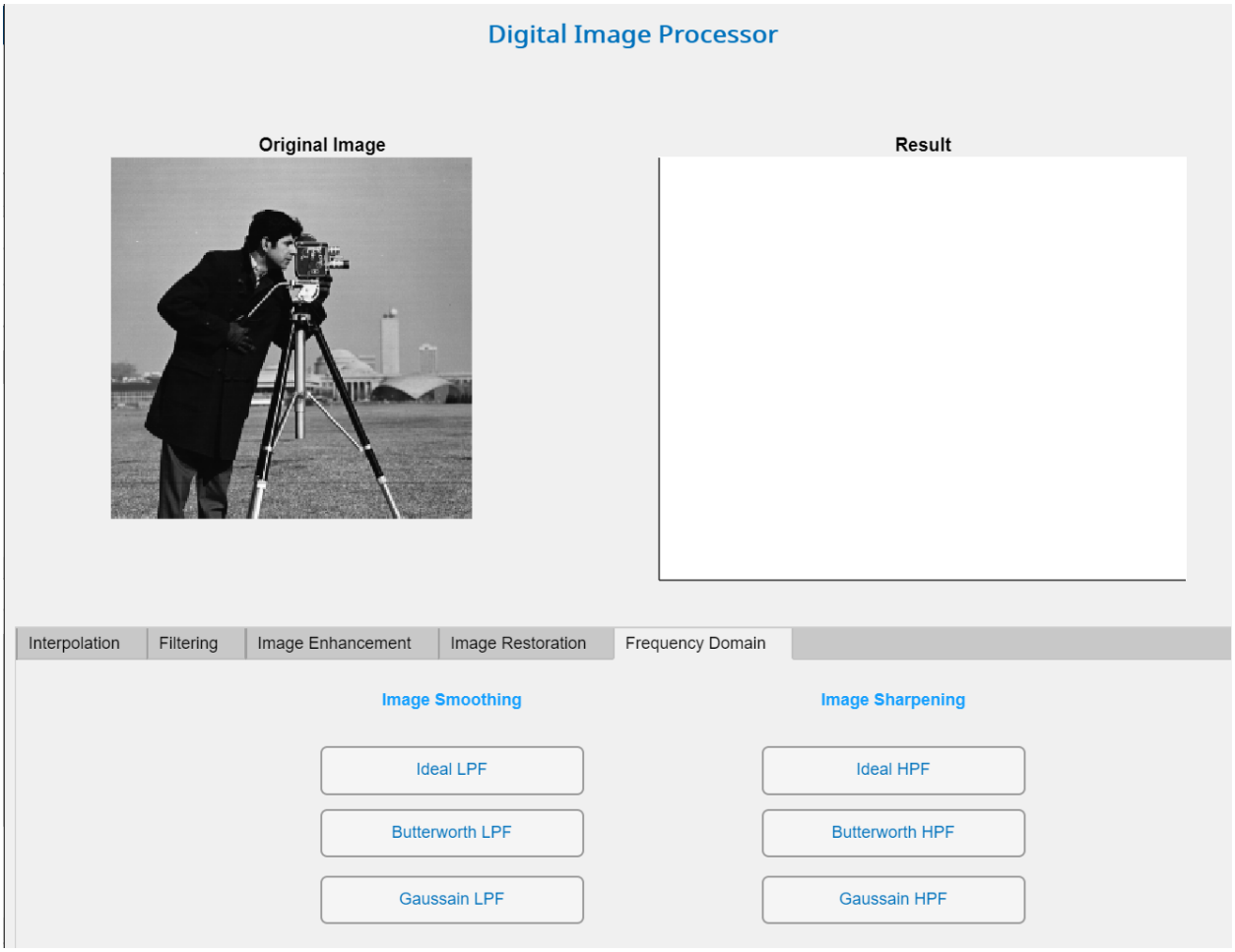
```
function MedianFilterButtonPushed(app, event)
    global noise_img;
    J = noise_img;
    p = medfilt3(J, [5,5,3]);
    imshow(p, 'parent', app.UIAxes_2);
end
```

Code for the Gaussian Filter:

```
    global noise_img;
    sigma = 1;
    W = 0;
    kernel = zeros(5, 5);
    for i = 1:5
        for j=1:5
            sq_dist = (i-3)^2 + (j-3)^2;
            kernel(i, j) = exp(-1*(sq_dist)/(2*sigma*sigma));
            W = W + kernel(i, j);
        end
    end
    kernel = kernel / W;
    [m, n] = size(noise_img);
    output = zeros(m, n);
    Im = padarray(noise_img, [2 ,2]);

    for i = 1:m
        for j = 1:n
            temp = Im(i:i+4, j:j+4);
            temp = double(temp);
            conv = temp.*kernel;
            output(i, j) = sum(conv(:));
        end
    end

    output = uint8(output);
    imshow(output, 'parent', app.UIAxes_2);
```

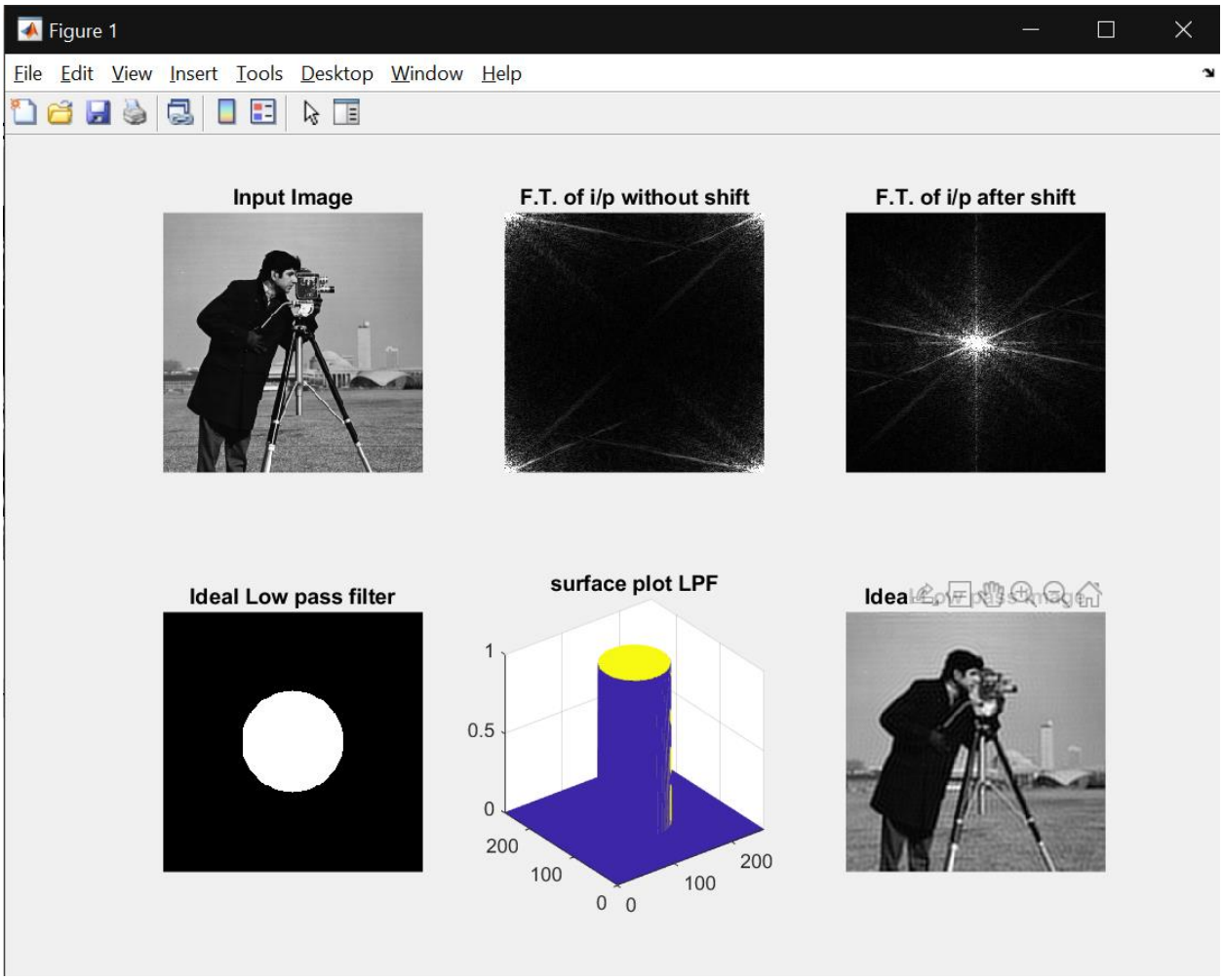
Task 5: Frequency Domain Filtering

Following is the initial interface of the frequency domain tab. The user can perform smoothing as well as sharpening using the provided filters in the tab.



Ideal Low Pass Filter:

Following shows the Fourier transform and the surface plot of the Ideal LPF.



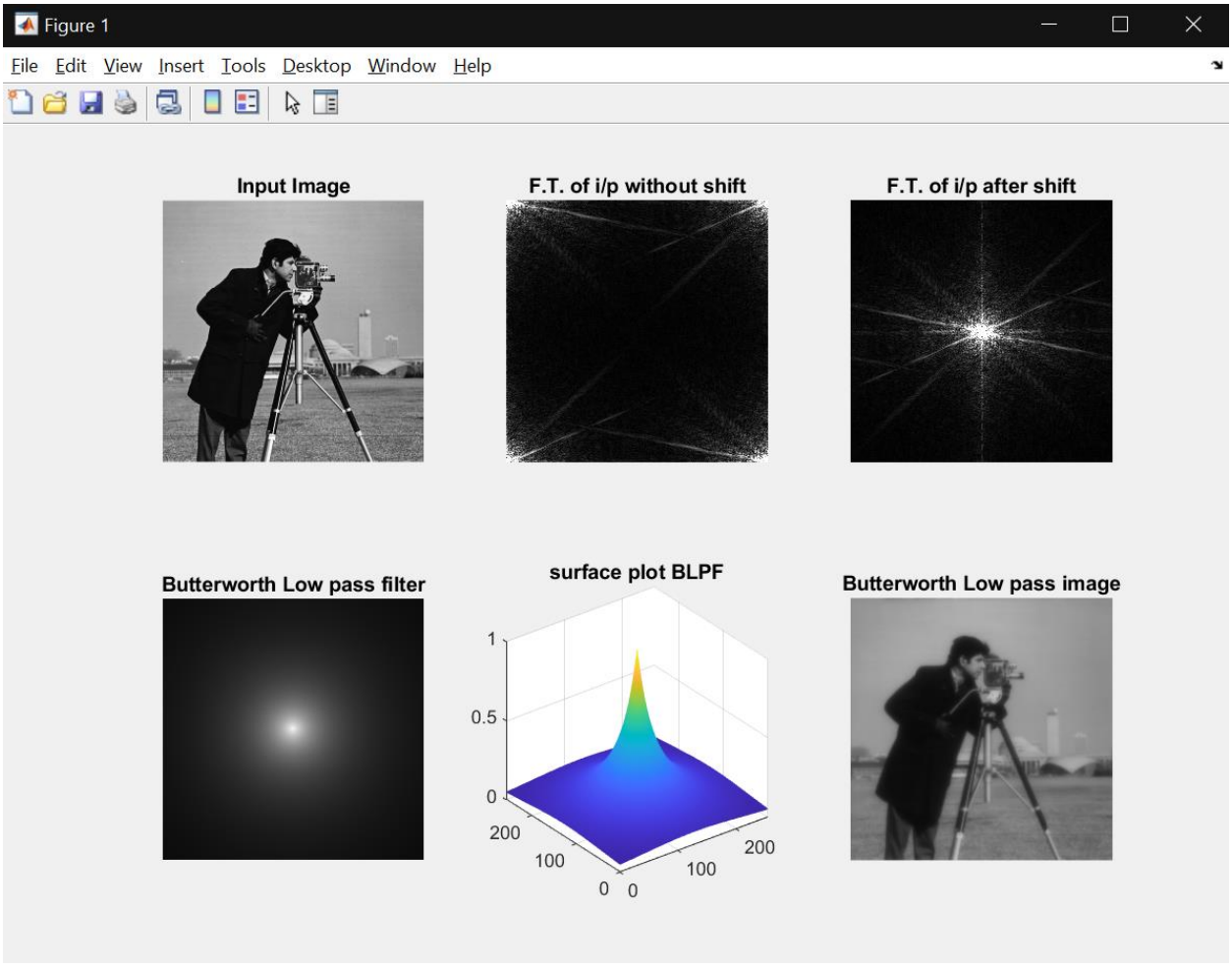
Code:

```
global freq_img;
a = im2double(freq_img);
subplot(2,3,1);
imshow(a);
title('Input Image');

[m,n] = size(a);  % size of input image
D0 = 50; % Assigning Cut-off Frequency
A = fft2(a);   %fourier transform of input image
subplot(2,3,2);
imshow(uint8(abs(A)));
title('F.T. of i/p without shift');

A_shift = fftshift(A);  %shifting origin
A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

subplot(2,3,3);
imshow(uint8(A_real));
title('F.T. of i/p after shift');
```

```
A_low = zeros(m,n);
d = zeros(m,n);
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        if d(u,v) <= D0
            A_low(u,v)=A_shift(u,v);
            filt(u,v) = 1;
        else
            A_low(u,v)=0;
            filt(u,v) = 0;
        end
    end
end

subplot(2,3,4);
imshow(filt)
title('Ideal Low pass filter')

subplot(2,3,5);
mesh(filt)
title('surface plot LPF')
B = fftshift(A_low);  %Reshifting the origin of filtered image
B_inverse = ifft2(B); %Taking inverse fourier transform
B_real = abs(B_inverse);%Taking real part(Low pass output image)

subplot(2,3,6);
imshow(B_real);
title('Ideal Low pass image');
imshow(B_real, 'parent', app.UIAxes_2);
```

Butterworth Low Pass Filter:

Following shows the Fourier transform and the surface plot of the Butterworth LPF.

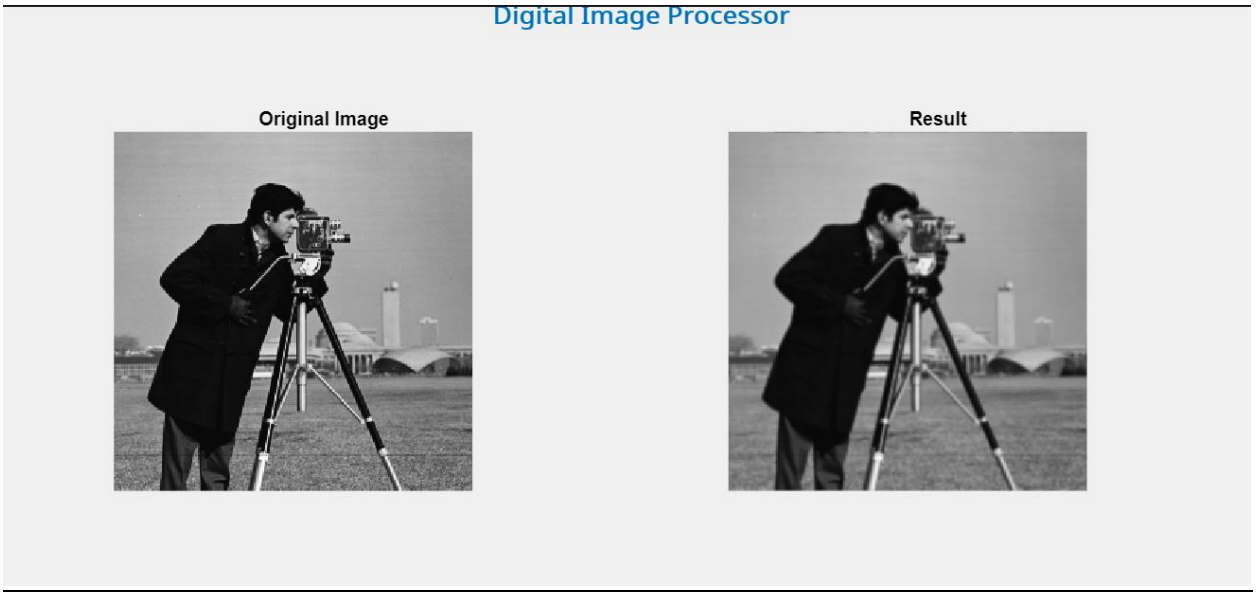


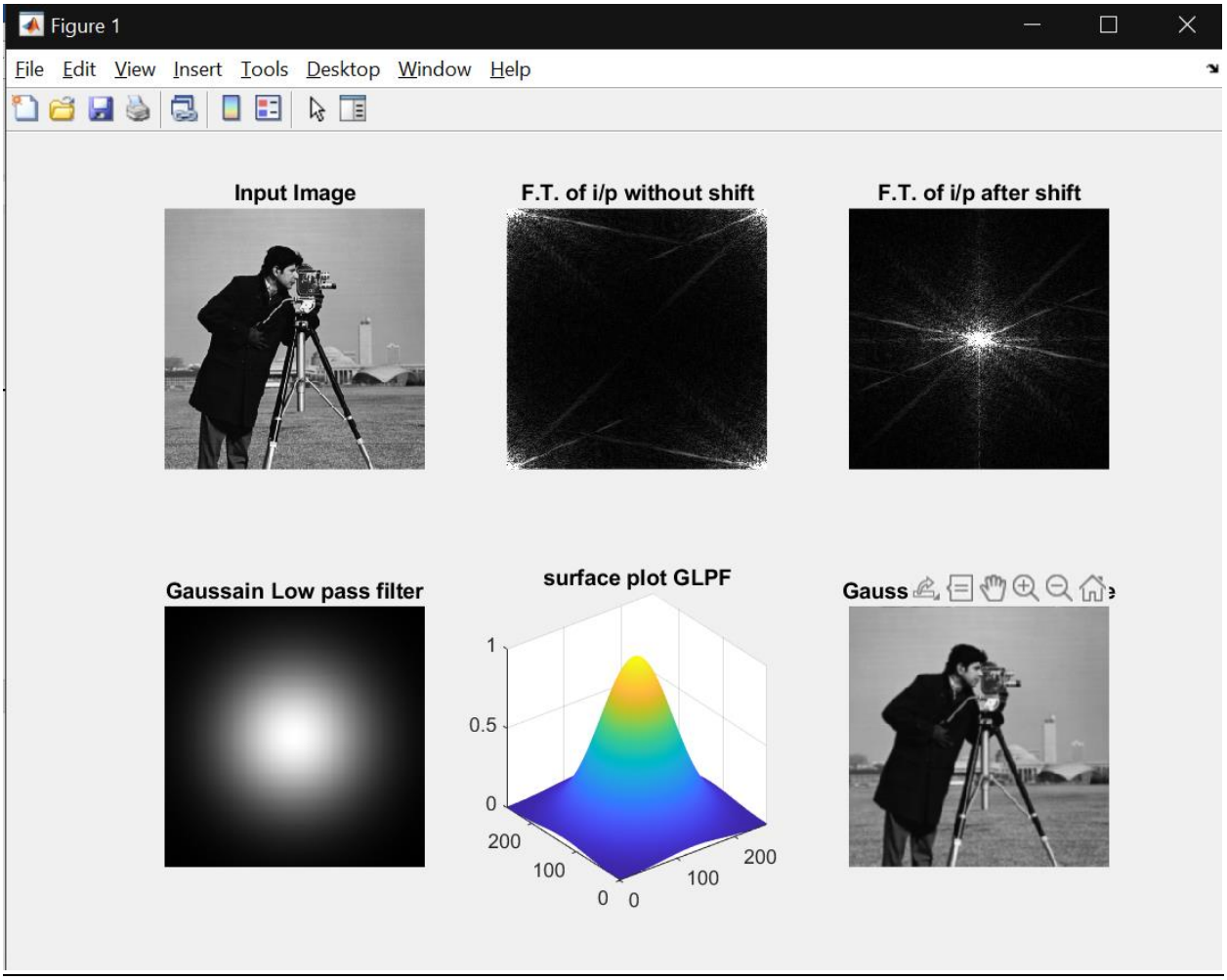
Code:

```
global freq_img;
a = im2double(freq_img);
subplot(2,3,1);
imshow(a);
title('Input Image');

[m,n] = size(a);  % size of input image
D0 = 50; % Assigning Cut-off Frequency
A = fft2(a);   %fourier transform of input image
subplot(2,3,2);
imshow(uint8(abs(A)));
title('F.T. of i/p without shift');

A_shift = fftshift(A);  %shifting origin
A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

subplot(2,3,3);
imshow(uint8(A_real));
title('F.T. of i/p after shift');

A_low = zeros(m,n);
d = zeros(m,n);
order = 1;
```

```
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        H(u,v) = 1/((1+d(u,v)/D0)^(2*order));
    end
end

subplot(2,3,4);
imshow(H)
title('Butterworth Low pass filter')

subplot(2,3,5);
mesh(H)
title('surface plot BLPF')

butter_LPF = A_shift.*H;
butter_LPF_inverse = ifft2(butter_LPF);
butter_LPF_real = abs(butter_LPF_inverse);

subplot(2,3,6);
imshow(butter_LPF_real);
title('Butterworth Low pass image');
imshow(butter_LPF_real, 'parent', app.UIAxes_2);
```

Gaussian Low Pass Filter:



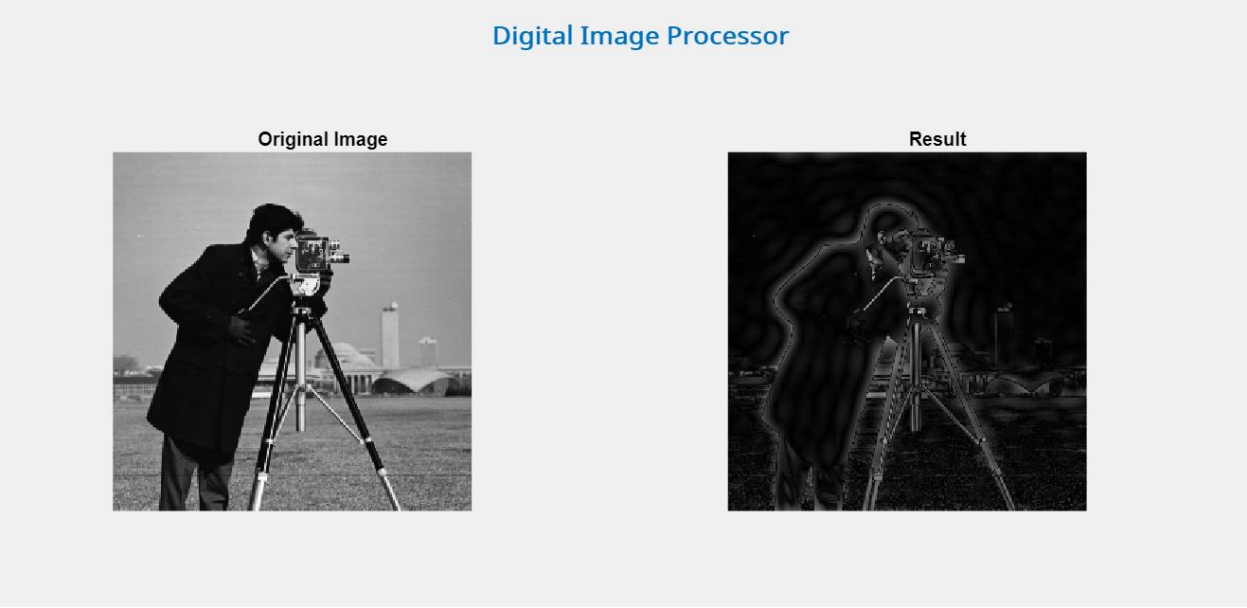


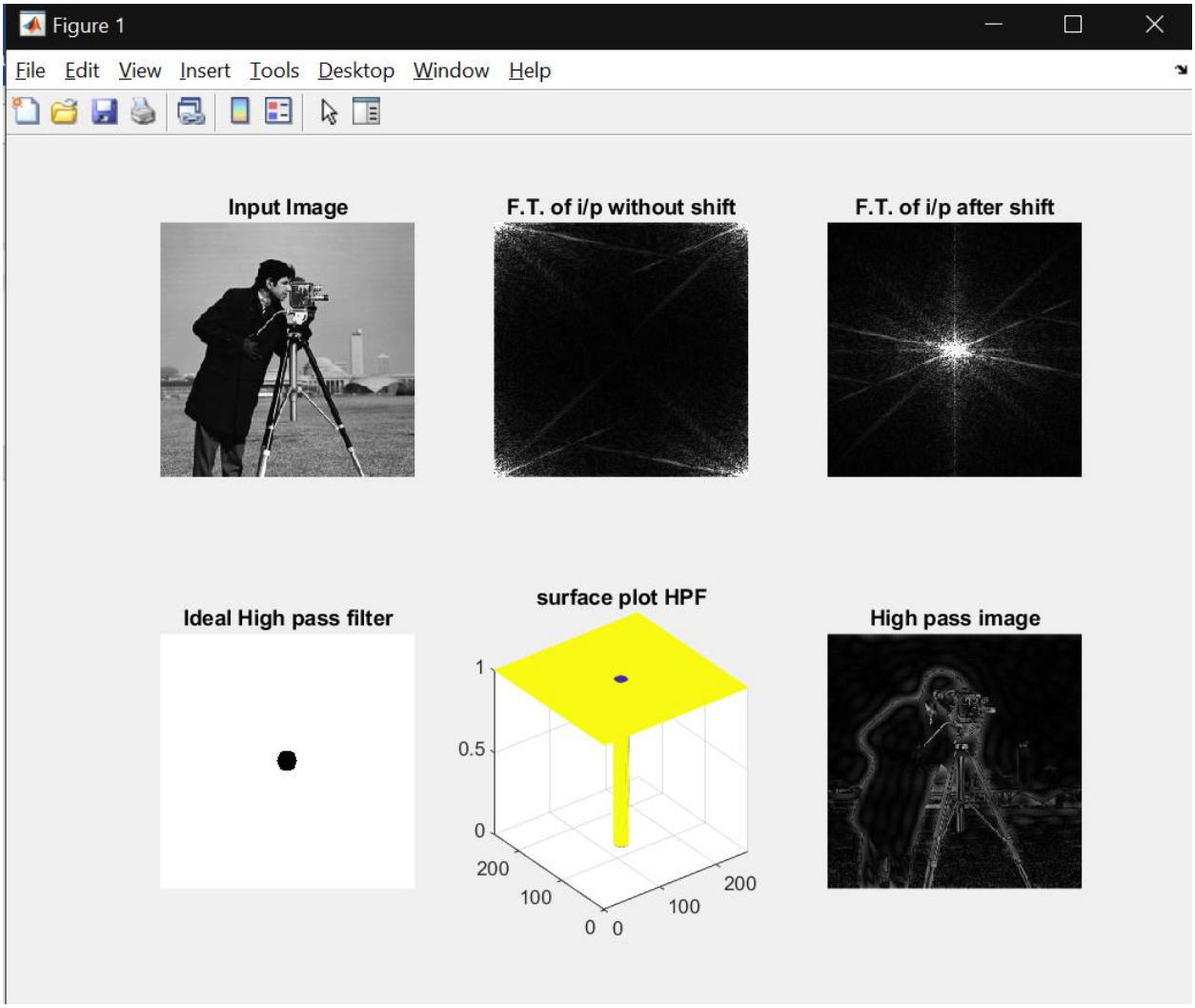
Code:

```
function GaussainLPFButtonPushed(app, event)
    global freq_img;
    a = im2double(freq_img);
    subplot(2,3,1);
    imshow(a);
    title('Input Image');

    [m,n] = size(a);  % size of input image
    D0 = 50; % Assigning Cut-off Frequency
    A = fft2(a);   %fourier transform of input image
    subplot(2,3,2);
    imshow(uint8(abs(A)));
    title('F.T. of i/p without shift');

    A_shift = fftshift(A);  %shifting origin
    A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

    subplot(2,3,3);
    imshow(uint8(A_real));
    title('F.T. of i/p after shift');

    A_low = zeros(m,n);
    d = zeros(m,n);
```

```
for u=1:m
    for v=1:n
        d=sqrt((u-m/2).^2+(v-n/2).^2);
        H(u,v) = exp(-(d^2)/(2*D0.^2));
    end
end

H_low = A_shift.*H;
H_low_real = H.*A_real;
H_low_shift = fftshift(H_low);
H_low_image = ifft2(H_low_shift);

subplot(2,3,4);
imshow(H)
title('Gaussain Low pass filter')

subplot(2,3,5);
mesh(H)
title('surface plot GLPF')

subplot(2,3,6);
imshow(abs(H_low_image));
title('Gaussian Low pass image');
imshow(abs(H_low_image), 'parent', app.UIAxes_2);
```

Ideal High Pass Filter:





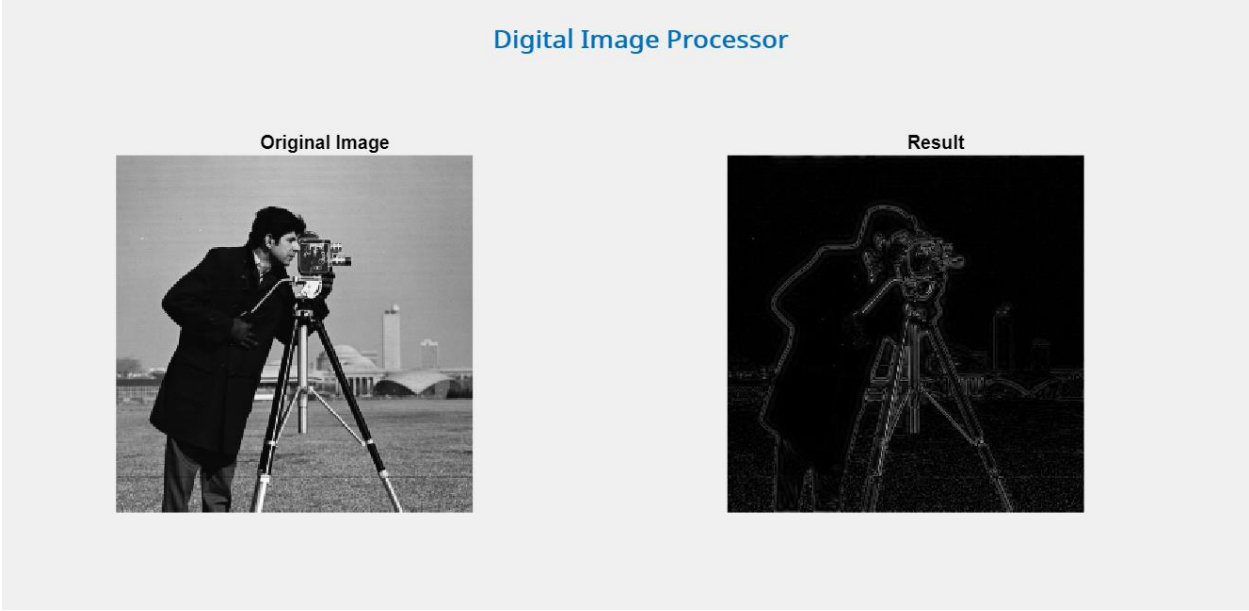
Code:

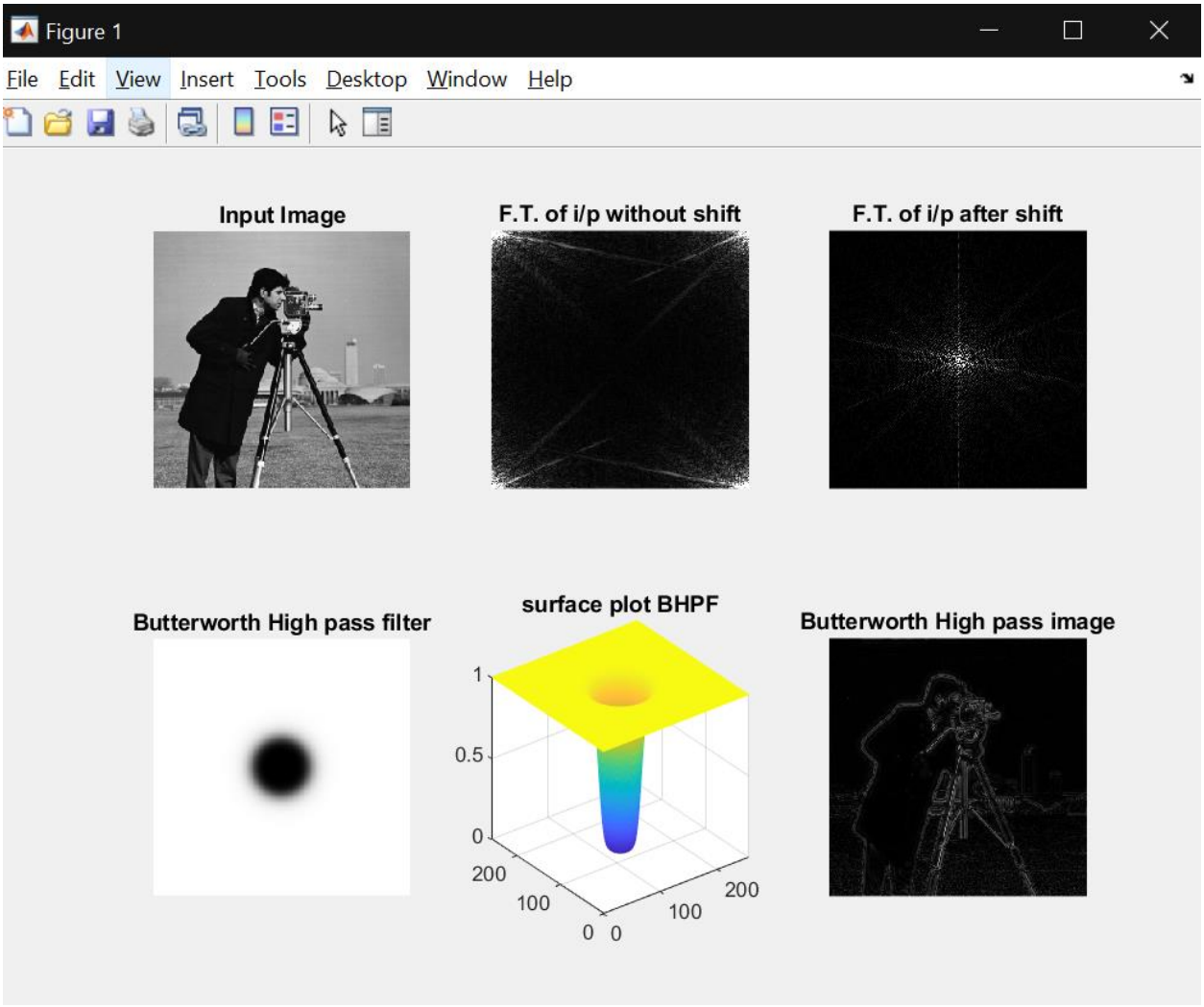
```
global freq_img;
a = im2double(freq_img);
subplot(2,3,1);
imshow(a);
title('Input Image');

[m,n] = size(a);  % size of input image
D0 = 10; % Assigning Cut-off Frequency
A = fft2(a);   %fourier transform of input image
subplot(2,3,2);
imshow(uint8(abs(A)));
title('F.T. of i/p without shift');

A_shift = fftshift(A);  %shifting origin
A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

subplot(2,3,3);
imshow(uint8(A_real));
title('F.T. of i/p after shift');

A_high = zeros(m,n);
D = zeros(m,n);
```

```
for u=1:m
    for v=1:n
        D(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        if D(u,v) <= D0
            A_high(u,v) = 0;
            filt(u,v) = 0;
        else
            A_high(u,v) = A_shift(u,v);
            filt(u,v) = 1;
        end
    end
end

subplot(2,3,4);
imshow(filt)
title('Ideal High pass filter')

subplot(2,3,5);
mesh(filt)
title('surface plot HPF')

B = fftshift(A_high);  %Reshifting the origin of filtered image
B_inverse = ifft2(B); %Taking inverse fourier transform
B_real = abs(B_inverse);%Taking real part(Low pass output image)
```

```
subplot(2,3,6);
imshow(B_real);
title('High pass image');
imshow(B_real, 'parent', app.UIAxes_2);
```

Butterworth High Pass Filter:



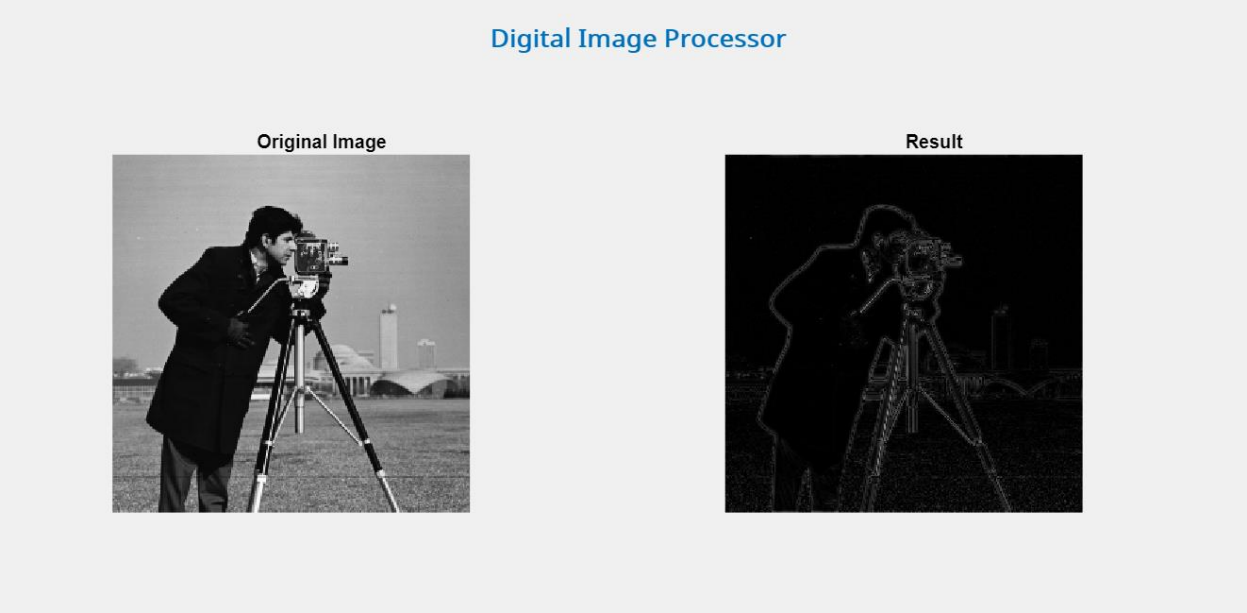


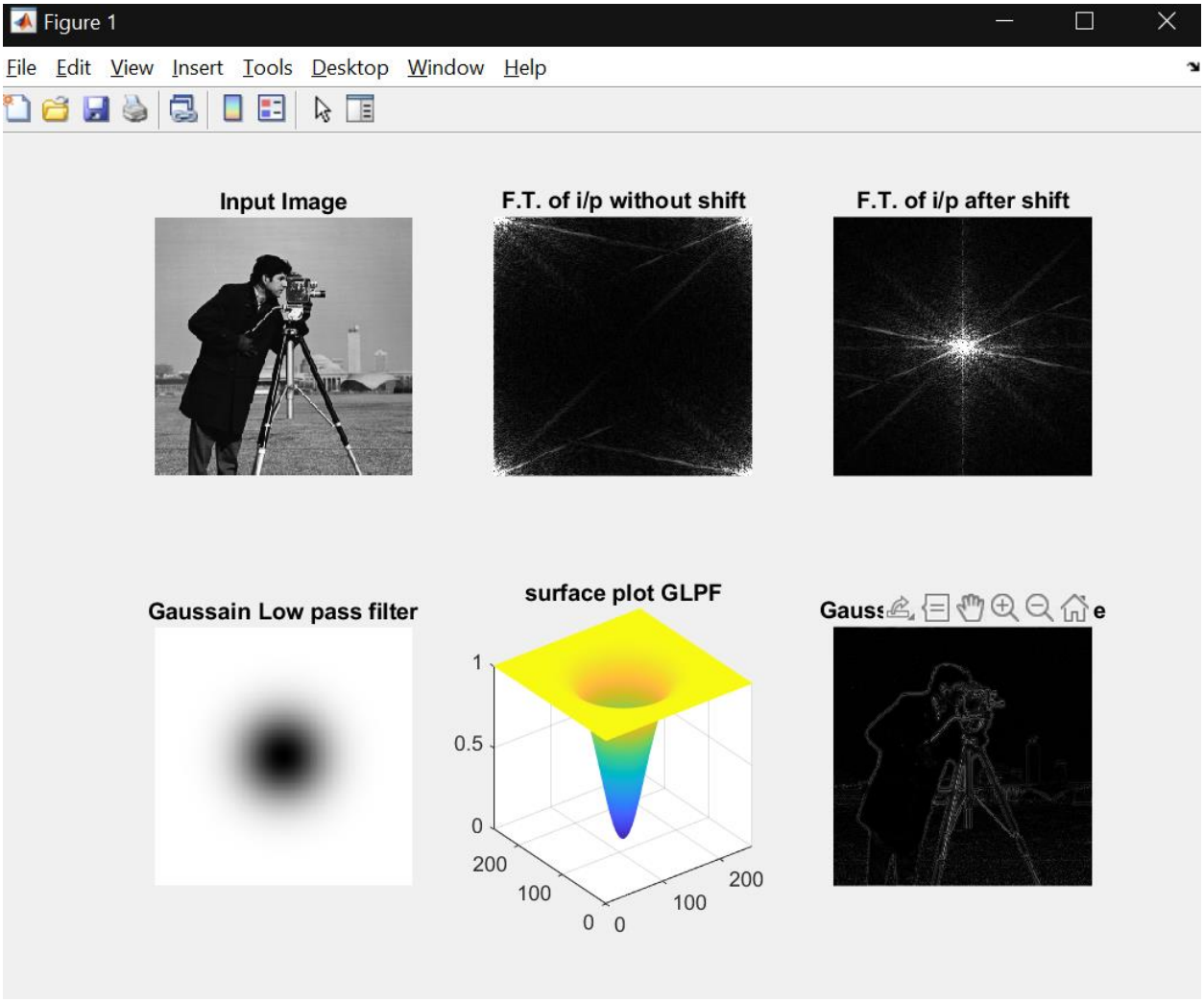
Code:

```
global freq_img;
a = im2double(freq_img);
subplot(2,3,1);
imshow(a);
title('Input Image');

[m,n] = size(a);  % size of input image
D0 = 30;
A = fft2(a);   %fourier transform of input image
subplot(2,3,2);
imshow(uint8(abs(A)));
title('F.T. of i/p without shift');

A_shift = fftshift(A);  %shifting origin
A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

subplot(2,3,3);
imshow(uint8(A_shift));
title('F.T. of i/p after shift');

d = zeros(m,n);
order = 4;
```

```
for u=1:m
    for v=1:n
        d(u,v)=sqrt((u-(m/2))^2+(v-(n/2))^2);
        H(u,v) = 1/((1+(D0/d(u,v))^(2*order)));
    end
end

butter_HPF = A_shift.*H;
butter_HPF_inverse = ifft2(butter_HPF);
butter_HPF_real = abs(butter_HPF_inverse);

subplot(2,3,4);
imshow(H)
title('Butterworth High pass filter')

subplot(2,3,5);
mesh(H)
title('surface plot BHPF')

subplot(2,3,6);
imshow(butter_HPF_real);
title('Butterworth High pass image');
imshow(butter_HPF_real, 'parent', app.UIAxes_2);
```

Gaussian High Pass Filter:





Code:

```
global freq_img;
a = im2double(freq_img);
subplot(2,3,1);
imshow(a);
title('Input Image');

[m,n] = size(a);  % size of input image
D0 = 30; % Assigning Cut-off Frequency
A = fft2(a);   %fourier transform of input image
subplot(2,3,2);
imshow(uint8(abs(A)));
title('F.T. of i/p without shift');

A_shift = fftshift(A);  %shifting origin
A_real = abs(A_shift);  %Real part of A_shift (Freq domain repres of image)

subplot(2,3,3);
imshow(uint8(A_real));
title('F.T. of i/p after shift');

d = zeros(m,n);
```

```
for u=1:m
    for v=1:n
        d=sqrt((u-m/2).^2+(v-n/2).^2);
        H(u,v) = 1 - exp(-(d^2)/(2*D0.^2));
    end
end

G_high = A_shift.*H;
G_high_real = H.*A_real;
G_high_shift = fftshift(G_high);
G_high_image = ifft2(G_high_shift);

subplot(2,3,4);
imshow(H)
title('Gaussain Low pass filter')

subplot(2,3,5);
mesh(H)
title('surface plot GLPF')

subplot(2,3,6);
imshow(abs(G_high_image));
title('Gaussian Low pass image');
imshow(abs(G_high_image), 'parent', app.UIAxes_2);
```

END