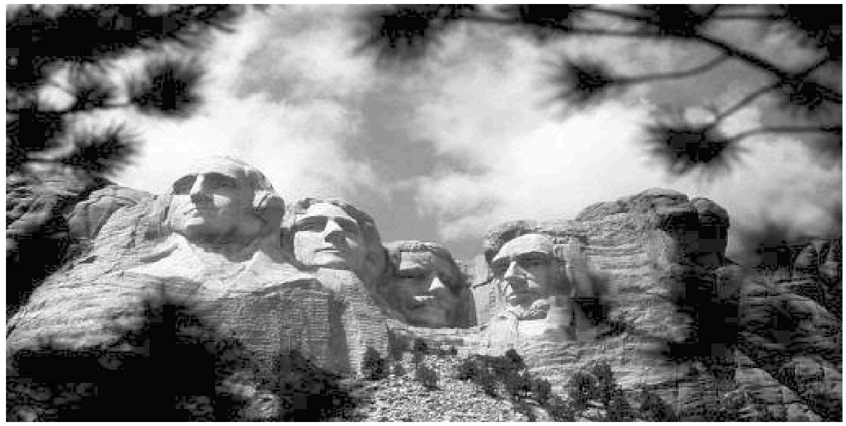
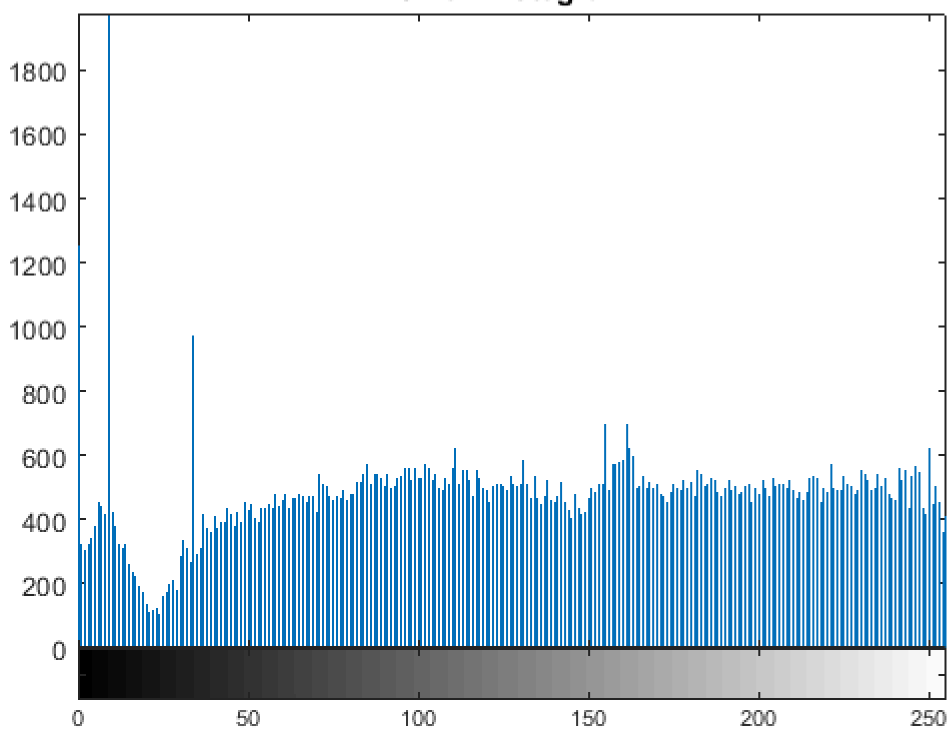
```
%Histogram Specification

clc;
clear all;
close all;
%Reading the input image
img1 = imread("givenhist.jpg");

[r1 c1] = size(img1); % Rows and columns in image

hist1_inp = zeros(256,1);

%Calculating input image histogram values
for i = 1:r1
    for j = 1:c1
        k = img1(i,j);
        hist1_inp(k+1) = hist1_inp(k+1) + 1;
    end
end

pdf1 = hist1_inp/(r1*c1); %pdf of input image histogram

%Calculating input image CDF
sk = zeros(256,1);
sk(1) = pdf1(1);
for i = 2:256
    sk(i) = sk(i-1)+ pdf1(i);
end

sk = round(sk*255);

%Reading the input image to be matched
img2 = imread("sphist.jpg");
hist2_inp = zeros(256,1);

%Calculating histogram values of the image to be matched
for i = 1:r1
    for j = 1:c1
        k = img2(i,j);
        hist2_inp(k+1) = hist2_inp(k+1) + 1;
    end
end

pdf2 = hist2_inp/(r1*c1); %pdf calculation
sz = zeros(256,1);
%cdf calculation
sz(1) = pdf2(1);
for i = 2:256
    sz(i) = sz(i-1)+ pdf2(i);
end

sz = round(sz*255);
```

```
%Transformation function calculation
T = (255)*ones(256,1);
for i = 1:256
    for j = 1:256
       if sk(i) < sz(j)
           if sz(j)-sk(i) > sk(i) - sz(j-1)
               T(i) = j-1;
               break;
           end
           if sz(j)-sk(i) < sk(i) - sz(j-1);
               T(i) = j;
               break;
          end
       end
     end
end

%Histogram specification
img_out = zeros(r1,c1);
for i = 1:r1
    for j = 1:c1
        img_out(i,j) = T(img1(i,j)+1)-1;
    end
end
img_out = uint8(img_out);

figure(1);
imshow(img1)
title("Given image");
figure(2);
imhist(img1);
title("Given Histogram");

figure(3);
imshow(img2)
title("Specifed image");
figure(4);
imhist(img2)
title("Specified Histogram");


figure(5);
imshow(img_out)
title("Transformed image");
figure(6);
imhist(img_out)
title("Transformed Histogram");
```

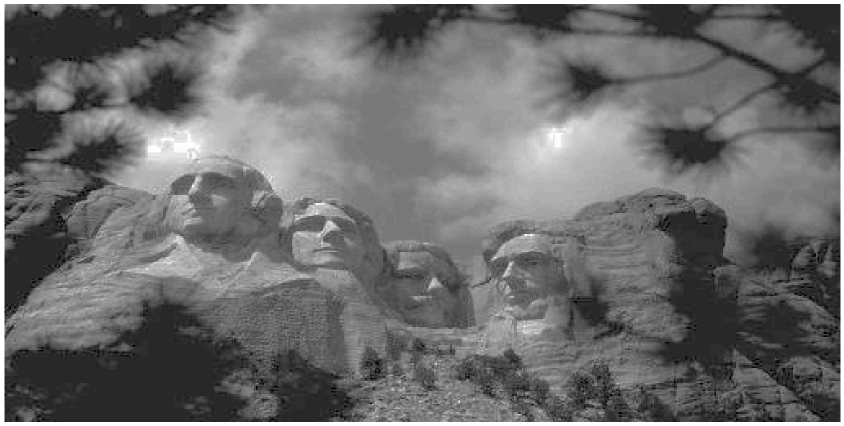
Given image



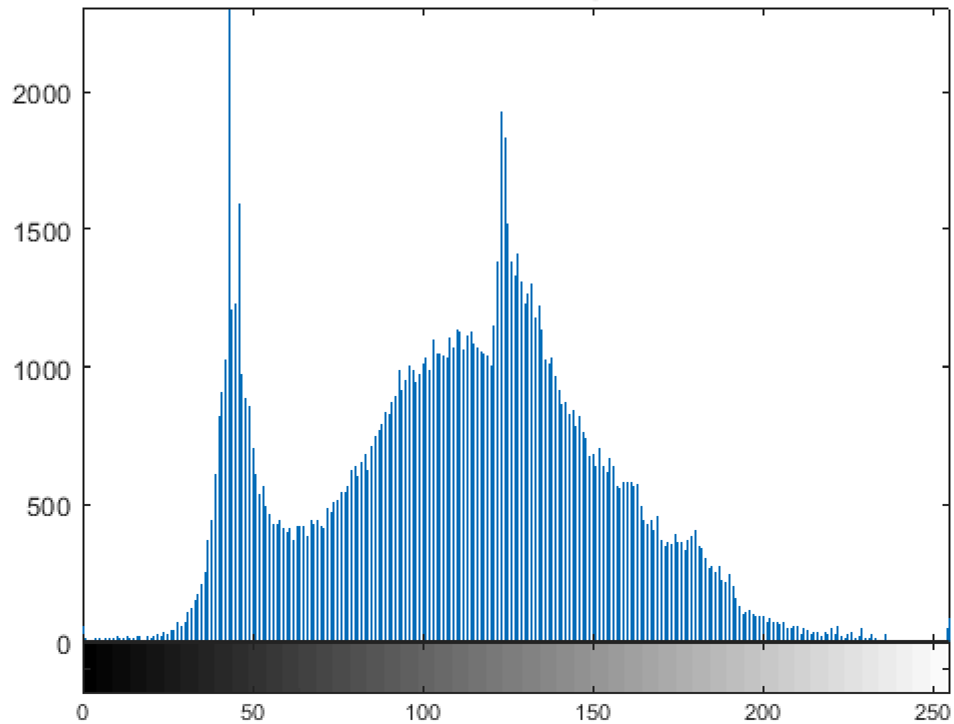
Given Histogram

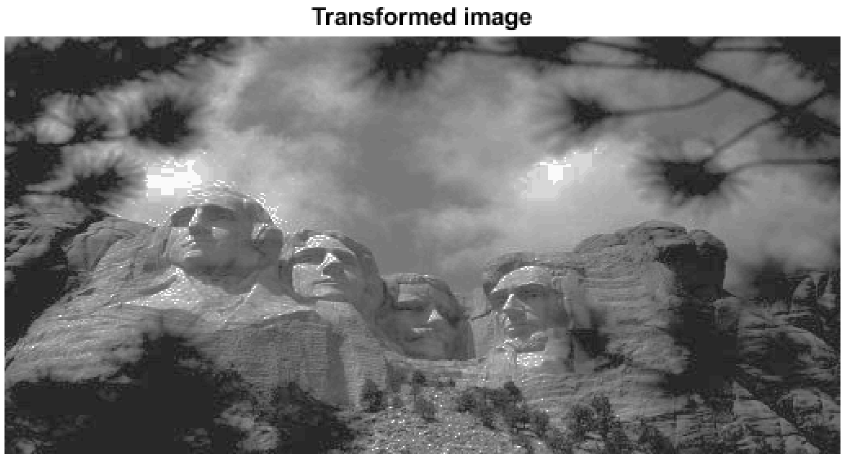


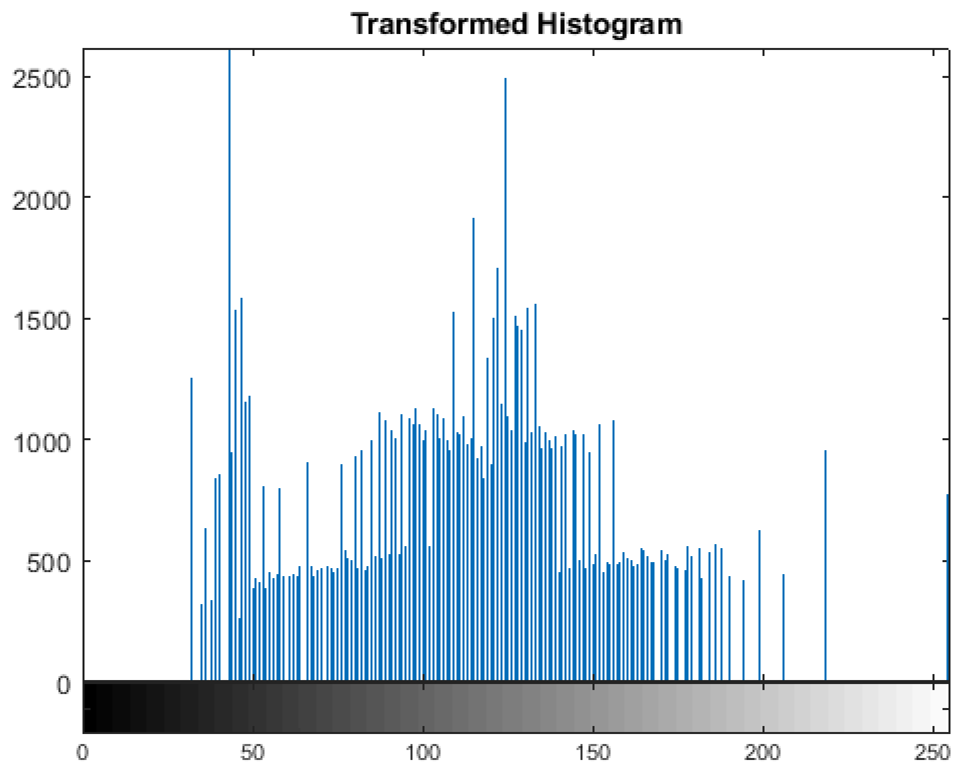
Specifed image



Specified Histogram







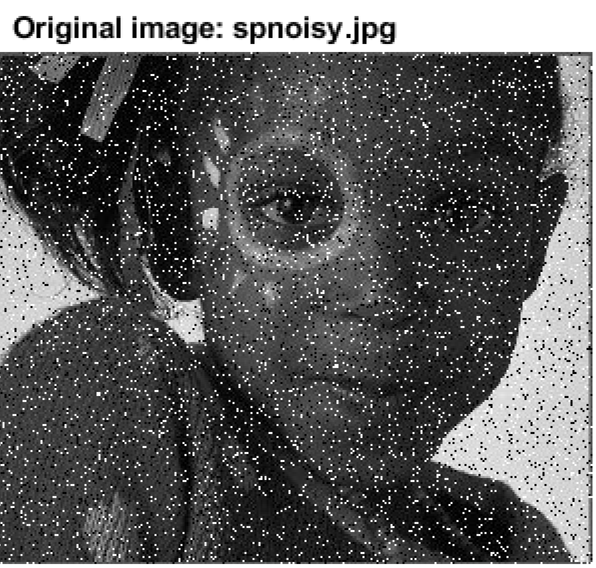
Published with MATLAB® R2020a

```
% Bilateral Filtering

% A. spnoisy.jpg

clc;
clear all;
close all;

%Reading the input image
im = imread("spnoisy.jpg");
im = rgb2gray(im); % Changing into grey scale
[R C] = size(im); % Rows and columns in image
im = double(im); % To carry out big number calculations on pixel
 intensities

vard =25; % Variance of domain component, sigma = 5
varr = 40000;  % Variance of range component, sigma = 200
norm_factor = 0;
sum = 0;
d = ones(R,C);% array of m*n containing all ones to store the enhanced
 image
count = 0;

% Starting the iteration at 3,3 pixel since the top left corner of its
 5*5
% patch will be at 1,1 so this way we will not be going out of the
 array,
% similarly ending the iteration at R-2,C-2, so that the bottom right
% corner of the 5*5 patch centred at R-2,C-2 will be at R,C hence we
 will
% still be staying inside the array.

for i = 3:R-2
     for j = 3:C-2
        for r = -2:2 % Iterating over 5*5 Neibourhood
            for c = -2:2
                ip = im(i+r,j+c);
                iu = im(i,j);
                df = exp(-1*(r^2 +c^2)/(2*vard)); % Eucledian distance
 between p and u for domain component
                rf = exp(-1*(ip-iu)^2/(2*varr)); % Squared difference
 between the intensity values ip and iu for range component
                sum = sum + rf*df*ip; % Summing pixel intensities
 weighted by domain and range components weights
                norm_factor = norm_factor + rf*df; % Sum of weights of
 domain and range components
            end
        end
        d(i,j) = sum/norm_factor; % Modified intensity value of
 im(i,j)
        sum = 0;
        norm_factor = 0;
```

```
    end
end

d1 = uint8(d); % type casting to unsigned int8 format
figure(1);
imshow(uint8(im));
title("Original image: spnoisy.jpg ");
figure(2);
imshow(d1);
title("Enhanced image: spnoisy.jpg");

% B. spunifnoisy.jpg

%Reading the input image
im = imread("spunifnoisy.jpg");
im = rgb2gray(im); % Changing into grey scale
[R C] = size(im); % Rows and columns in image
im = double(im); % To carry out big number calculations on pixel
 intensities

vard =100; % Variance of domain component
varr = 40000;  % Variance of range component
norm_factor = 0;
sum = 0;
d = ones(R,C);% array of m*n containing all ones to store the enhanced
 image
count = 0;

% Starting the iteration at 3,3 pixel since the top left corner of its
 5*5
% patch will be at 1,1 so this way we will not be going out of the
 array,
% similarly ending the iteration at R-2,C-2, so that the bottom right
% corner of the 5*5 patch centred at R-2,C-2 will be at R,C hence we
 will
% still be staying inside the array.

for i = 3:R-2
    for j = 3:C-2
       for r = -2:2 % Iterating over 5*5 Neibourhood
           for c = -2:2
               ip = im(i+r,j+c);
               iu = im(i,j);
               df = exp(-1*(r^2 +c^2)/(2*vard)); % Eucledian distance
 between p and u for domain component
               rf = exp(-1*(ip-iu)^2/(2*varr)); % Squared difference
 between the intensity values ip and iu for range component
               sum = sum + rf*df*ip; % Summing pixel intensities
 weighted by domain and range components weights
               norm_factor = norm_factor + rf*df; % Sum of weights of
 domain and range components
           end
       end
```

```
        d(i,j) = sum/norm_factor; % Modified intensity value of
 im(i,j)
        sum = 0;
        norm_factor = 0;
     end
end

d1 = uint8(d); % type casting to unsigned int8 format
figure(3);
imshow(uint8(im));
title("Original image: spunifnoisy.jpg ");
figure(4);
imshow(d1);
title("Enhanced image: spunifnoisy.jpg ");

% C. unifnoisy.jpg

%Reading the input image
im = imread("unifnoisy.jpg");
im = rgb2gray(im); % Changing into grey scale
[R C] = size(im); % Rows and columns in image
im = double(im); % To carry out big number calculations on pixel
 intensities

vard =4; % Variance of domain component
varr = 400;  % Variance of range component
norm_factor = 0;
sum = 0;
d = ones(R,C);% array of m*n containing all ones to store the enhanced
 image
count = 0;

% Starting the iteration at 3,3 pixel since the top left corner of its
 5*5
% patch will be at 1,1 so this way we will not be going out of the
 array,
% similarly ending the iteration at R-2,C-2, so that the bottom right
% corner of the 5*5 patch centred at R-2,C-2 will be at R,C hence we
 will
% still be staying inside the array.

for i = 3:R-2
     for j = 3:C-2
        for r = -2:2 % Iterating over 5*5 Neibourhood
            for c = -2:2
                ip = im(i+r,j+c);
                iu = im(i,j);
                df = exp(-1*(r^2 +c^2)/(2*vard)); % Eucledian distance
 between p and u for domain component
                rf = exp(-1*(ip-iu)^2/(2*varr)); % Squared difference
 between the intensity values ip and iu for range component
                sum = sum + rf*df*ip; % Summing pixel intensities
 weighted by domain and range components weights
```

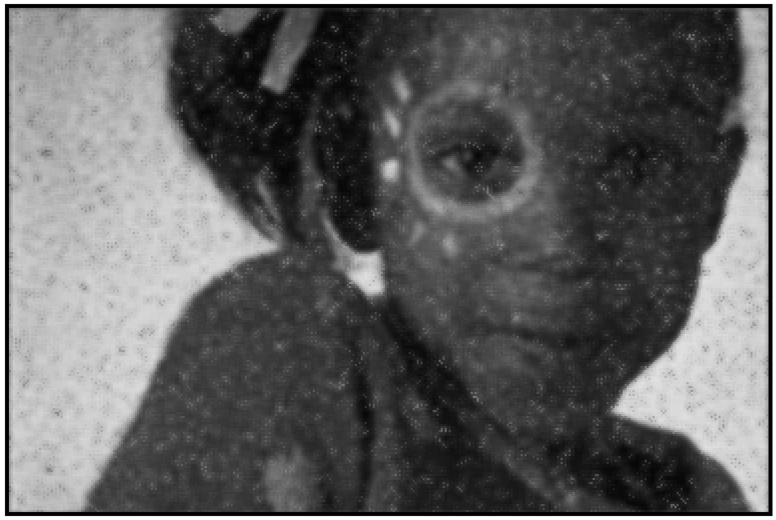
```
                norm_factor = norm_factor + rf*df; % Sum of weights of
 domain and range components
            end
        end
        d(i,j) = sum/norm_factor; % Modified intensity value of
 im(i,j)
        sum = 0;
        norm_factor = 0;
     end
end

d1 = uint8(d); % type casting to unsigned int8 format
figure(5);
imshow(uint8(im));
title("Original image: unifnoisy.jpg ");
figure(6);
imshow(d1);
title("Enhanced image: unifnoisy.jpg ");
```



Enhanced image: spnoisy.jpg

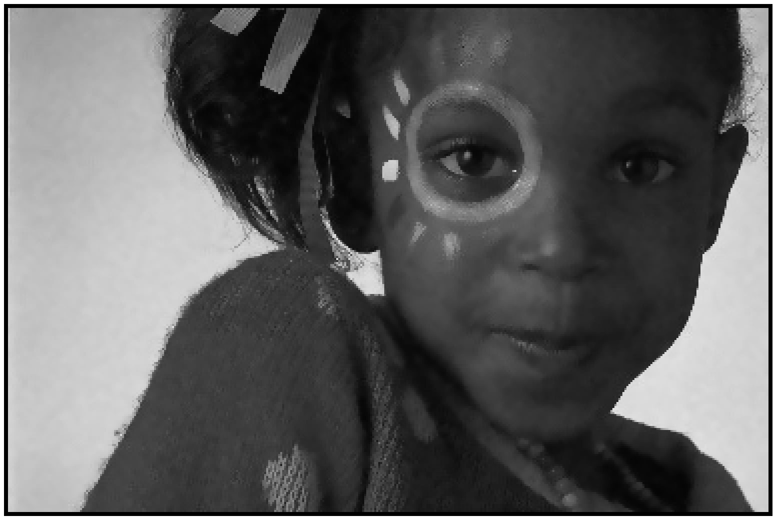
Original image: spunifnoisy.jpg

Enhanced image: spunifnoisy.jpg



Original image: unifnoisy.jpg

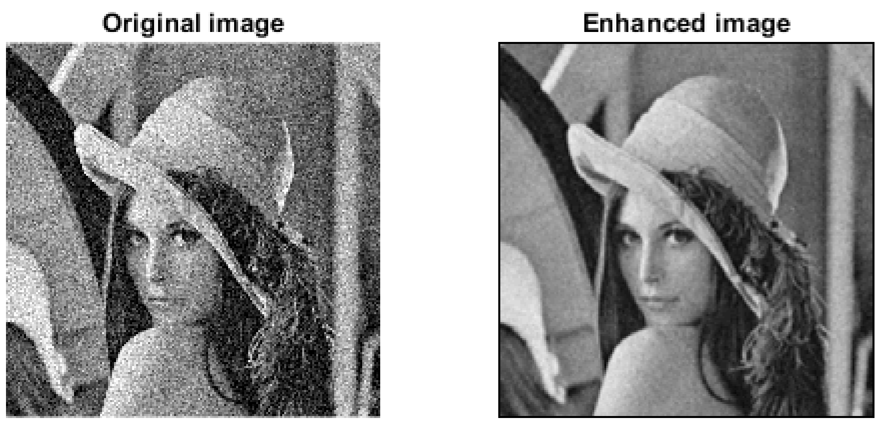
Enhanced image: unifnoisy.jpg



Published with MATLAB® R2020a

```
% non-local means filter

close all;
clc;
clear all;

%Reading the input image
im = imread('lenna.noise.jpg');

subplot(1,2,1);% Original image
imshow(im)
title("Original image");

im = double(im); % To carry out big number calculations on pixel
 intensities
[m,n] = size(im); % Number of rows and columns in image
d = ones(m,n); % array of m*n containing all ones

%Starting the iteration at 6,6 because in its 5*5 neibourhood at the
%top-left corner pixel will be at 4,4 and considering 7*7 patch around
 this
%pixel its top left pixel will be at 1,1, so we will not be going out
 of
%array using 6,6 as stating point. Similary ending at m-6,n-6.

for i = 6:m-6
    for j = 6:n-6
        v = im(i-3:i+3,j-3:j+3); % 7*7 patch around im(i,j)
        sum = 0; % It will be storing the sum of weights
        patch = zeros(7,7); % It will be storing sum of weighted
 patches in the 5*5 neibourhood of v
        for r = -2:2 %Iterating over 5*5 neibourhood
            for c = -2:2
                x = i+r;
                y = j+c;
                w = im(x-3:x+3,y-3:y+3); %ones of the patches in the
 5*5 neibourhood of v
                diff = (v-w)^2; % Squared differences between the
 intensity values of the two patches

                %Finding out the sum of all the sq differences in the
 diff
                %array which is 7*7
                norm = 0; % will be storing the sum of all the sq
 differences
                for p = 1:7
                    for q=1:7
                        norm = norm + diff(p,q);
                    end
                end
                s = exp(-1*(norm)/10^10);% Calculating the weight on
 the patch
```

```
                w = w.*s;
                patch = patch + w; %Sum of the intensity values in all
 weighted patches in the 5*5 neibourhood
                sum = sum + s; % Sum of all weights in the 5*5
 neibourhood
            end
        end
        d(i-3:i+3,j-3:j+3) = round(patch/sum); % Revised intensity
 values of im(i,j) patch
    end
end

d1 = uint8(d); % type casting to unsigned int8 format
subplot(1,2,2);
imshow(d1); % Non-local means filtered image
title("Enhanced image");
```

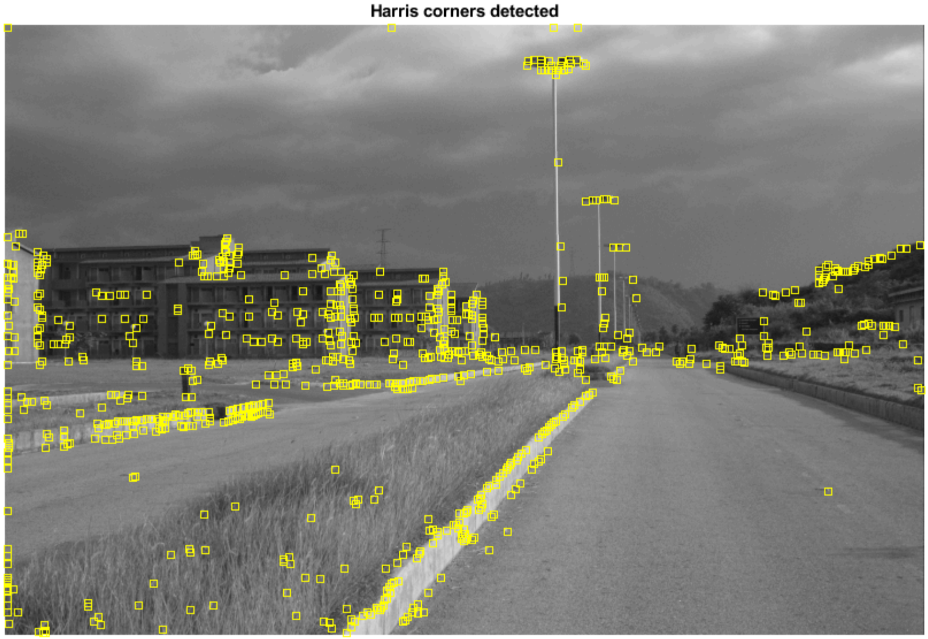


Published with MATLAB® R2020a

```
% Corner Detection

clc;
clear all;
close all;
%Reading the input image
im = imread("IITG.jpg");
im = rgb2gray(im);% Changing into grey scale

% Considering a 3*3 sobel mask
dx = [-1 -2 -1;0 0 0;1 2 1];
dy = [-1 0 1;-2 0 2;-1 0 1];

% Constructing gaussian kernel
sigma = 1;
mask = zeros(3,3);
sum = 0;
for i=1:3
    for j=1:3
        e = (i-2)^2+(j-2)^2;
        mask(i,j) = exp((-1*e)/(2*sigma^2));
        sum = sum + mask(i,j);
    end
end
mask = mask/sum; %Normalising

Ix = xcorr2(im,dx); %Gradient along X
Iy = xcorr2(im,dy); %Gradient along Y

Ix2 = xcorr2(Ix.^2,mask); %Convolving with gaussian kernel as the
 weighting factor
Iy2 = xcorr2(Iy.^2,mask);
Ixy = xcorr2(Ix.*Iy,mask);
Ix =  xcorr2(Ix,mask);
Iy =  xcorr2(Iy,mask);

R = ((Ix2.*Iy2)-(Ixy).^2 -0.05*(Ix+Iy).^2); %Corner score

max_sup = ordfilt2(R,9,ones(3,3)); %Maximal Supression
harris_points = (R==max_sup) & (R > 5*10^7); %Another step of
 thresholding
[r,c] = find(harris_points); %(x,y) co-ordinates of corner points

figure(1)
imshow(im);
title("Original image");
figure(2)
imshow(im);
hold on;
plot(c,r,'ys');
title("Harris corners detected")
```

Original image

Harris corners detected



Published with MATLAB® R2020a