# Report for Question 2 – Bilateral Filtering

Barbara Image 1

Grass Image 4

Honey Comb Image 5

Spatial Gaussian Mask 7

Code Explanation 8

First, we read the images.

myNumOfColors = 200;  
myColorScale = [ [0:1/(myNumOfColors-1):1]' , [0:1/(myNumOfColors-1):1]' , [0:1/(myNumOfColors-1):1]' ];  
% First, read the images  
tic;  
imGrass = imread("../data/grass.png");  
imHoneyComb = imread("../data/honeyCombReal.png");  
imBarbara = load("../data/barbara.mat");  
imBarbara = cast(imBarbara.imageOrig, 'uint8');

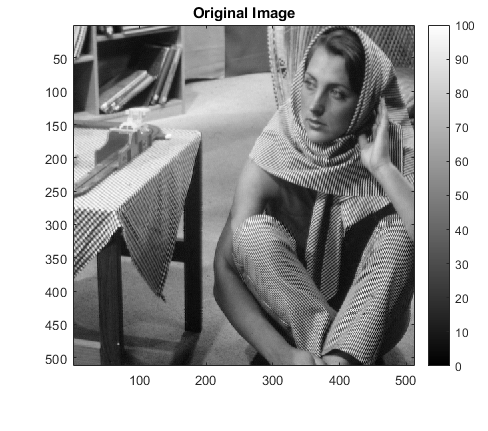
We have two parameters and , the standard deviation of the spatial gaussian kernel and the intensity gaussian kernel. Run the bilateral filtering function on the images, at the “optimal” values of the parameters and (refer to table in the RMSD Results section)

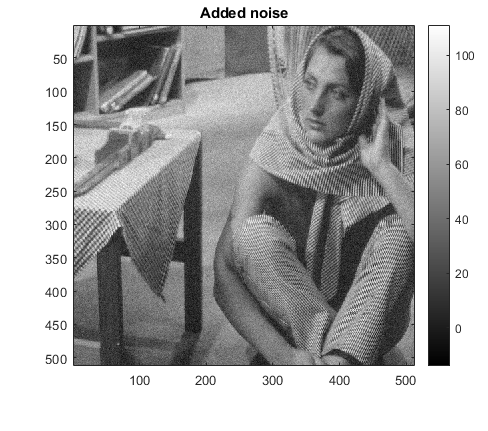
The measure of optimality is the Root Mean Square distance, of the original image, say A, and the filtered image, say B, defined as where ‘p’ varies over the pixels of the image. Note that the RMSD may be different in different iterations for the same values of the parameters, due to the randomness in the noise added. Thus, we average the RMSD over 20 iterations, and report the result up to 2 decimal places.

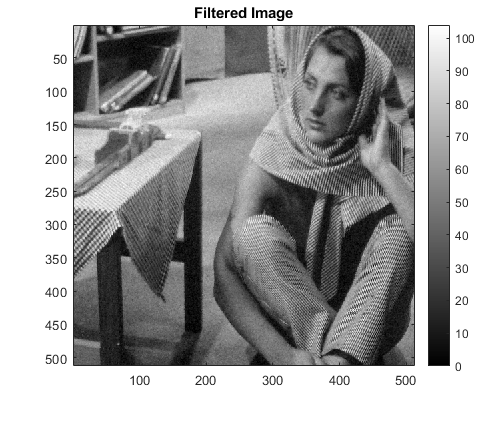
## Barbara Image

sigma\_space = 0.6;  
sigma\_intensity = 18;  
r1 = 0;  
for i = 1 : 20  
 [rmsd,im\_noisy,newim] = myBilateralFiltering(imBarbara, sigma\_space, sigma\_intensity);  
 r1 = r1 + rmsd;  
end  
disp("The optimal avg RMSD is:");  
disp(r1/20);  
% f1 = figure;  
% figure(f1);  
figure(), imagesc(imBarbara), title("Original Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(im\_noisy), title("Added noise"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(newim), title("Filtered Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;

The optimal avg RMSD is:  
 2.4932







### RMSD Results

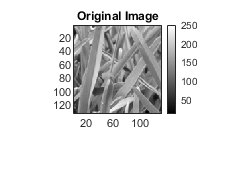
We run the bilateral filtering in a similar fashion, at the specified values. The first row has the RMSD at the optimal parameter values for reference.

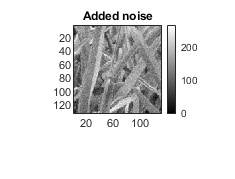
|  |  |  |
| --- | --- | --- |
|  |  | RMSD |
|  |  | 2.49 |
|  |  | 2.60 |
|  |  | 2.44 |
|  |  | 2.50 |
|  |  | 2.50 |

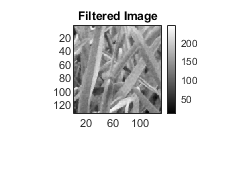
## Grass Image

sigma\_space = 1;  
sigma\_intensity = 46;  
r2 = 0;  
for i = 1 : 20  
 [rmsd,im\_noisy,newim] = myBilateralFiltering(imGrass, sigma\_space, sigma\_intensity);  
 r2 = r2 + rmsd;  
end  
disp("The optimal avg RMSD is:");  
disp(r2/20)  
figure(), imagesc(imGrass), title("Original Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(im\_noisy), title("Added noise"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(newim), title("Filtered Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;

The optimal avg RMSD is:  
 5.0338







### RMSD Results

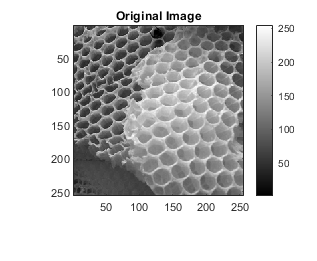
We run the bilateral filtering in a similar fashion, at the specified values. The first row has the RMSD at the optimal parameter values for reference.

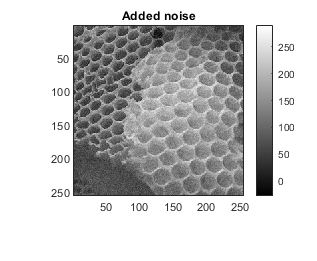
|  |  |  |
| --- | --- | --- |
|  |  | RMSD |
|  |  | 5.03 |
|  |  | 5.06 |
|  |  | 5.06 |
|  |  | 5.06 |
|  |  | 5.08 |

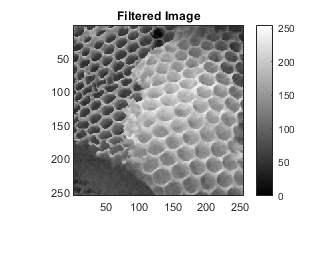
## Honey Comb Image

sigma\_space = 1.2;  
sigma\_intensity = 34;  
r3 = 0;  
for i = 1 : 20  
 [rmsd,im\_noisy,newim] = myBilateralFiltering(imHoneyComb, sigma\_space, sigma\_intensity);  
 r3 = r3 + rmsd;  
end  
disp("The optimal avg RMSD is:");  
disp(r3/20);  
figure(), imagesc(imHoneyComb), title("Original Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(im\_noisy), title("Added noise"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;  
figure(), imagesc(newim), title("Filtered Image"), colormap(myColorScale), daspect([1 1 1]), colorbar, truesize;

The optimal avg RMSD is:  
 4.9341







### RMSD Results

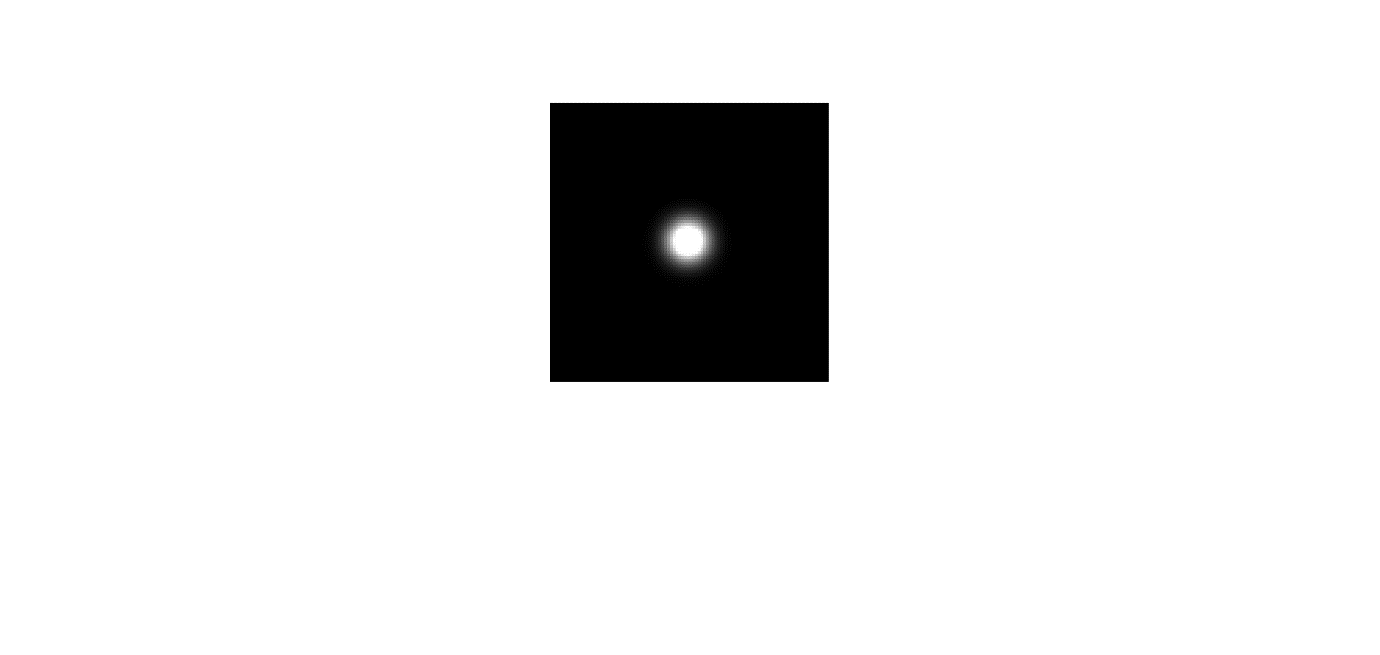
We run the bilateral filtering in a similar fashion, at the specified values. The first row has the RMSD at the optimal parameter values for reference.

|  |  |  |
| --- | --- | --- |
|  |  | RMSD |
|  |  | 4.93 |
|  |  | 4.95 |
|  |  | 4.97 |
|  |  | 4.96 |
|  |  | 4.95 |

Note that the optimality of the intensity bandwidth is more visible than that of the spatial bandwidth. Of course, one possible reason is the randomness in the noise. Now note that controls the window size for bilateral filtering, and the factors of 0.9 and 1.1 may not change the window size that much. On the other hand, the effect of the intensity bandwidth is directly eliminating the pixel intensity values from the noise. This is especially visible in the grass image, since that image has a lot of intensities at similar values, and very low contrast. So, noise has a lot more effect.

## Spatial Gaussian Mask

The spatial gaussian mask for a 100x100 window, with centre at 50, 50 and standard deviation 5 is shown below. This comes from the function ‘spacial\_gaussian\_filter’ used in the code. Clearly, the weights (high intensity means high weight) fall off very quickly and in a manner that is rotationally invariant, as expected.



## Code Explanation

function [rmsd,im\_noisy, newim] = myBilateralFiltering(im, sigma\_space, sigma\_intensity)

[M,N] = size(im);  
 im\_min = double(min(im,[],'all'));  
 im\_max = double(max(im,[],'all'));  
 % corrupt the image with IID gaussian noise  
 im\_noisy = double(im) + 0.05\*im\_max\*randn(M,N);  
 window\_len = round(3\*sigma\_space);  
 im\_max = 2\*im\_max;

store\_intensity\_gaussians is nothing but a matrix storing the output values for all combinations of intensity differences given to the gaussian intensity kernel. This can be computed in time O(MN), so we can simply read off these values when needed.

store\_intensity\_gaussians = zeros(im\_max+1, im\_max+1);  
 for i = 1 : im\_max+1  
 for j = 1 : im\_max+1  
 store\_intensity\_gaussians(i,j) = exp(-((i-j)^2)/(2\*sigma\_intensity\*sigma\_intensity));  
 end  
 end  
 % output image  
 newim = zeros(M,N);  
 for i = 1:M  
 for j = 1:N  
 [iLow, iHigh, jLow, jHigh] = WindowCorners(im\_noisy, window\_len, i, j);  
 window = im\_noisy(iLow:iHigh,jLow:jHigh);  
 spatial\_mask = spacial\_gaussian\_filter(window, sigma\_space, i-iLow+1, j-jLow+1);  
 intensity\_mask = intensity\_gaussian\_filter(reshape(window,[numel(window) 1]), im\_noisy(i,j), store\_intensity\_gaussians);  
 intensity\_mask = reshape(intensity\_mask, size(window));  
 mask = spatial\_mask.\*intensity\_mask;  
 mask = mask./sum(mask,'all');  
 newim(i,j) = sum(mask.\*window, 'all');  
 %newim(i,j) = sum(spatial\_mask.\*window, 'all');  
 end  
 end  
 newim = cast(newim, 'uint8');  
  
 % RMSD CALCULATION  
 rmsd\_matrix = (im - newim).\*(im-newim);  
 rmsd = sqrt(sum(rmsd\_matrix,'all')/numel(rmsd\_matrix));

end

The two for loops run over the image. In each iteration, (i,j) is the “centre” of the window to be considered around it. Of course, if (i,j) is near an edge of the image, the window will be cropped accordingly. After that, the two masks, spatial gaussian kernel and intensity gaussian kernel are computed. The latter utilises the store\_intensity\_gaussians, while the former is a simple LTI gaussian filter. Then we multiply both these pointwise with the actual window (after normalization) to get the intensity at that pixel in the filtered image (stored in newim). Finally, using the definition, we compute the RMSD between the input image and the newly generated image.

windowCorners is a function to obtain the corners of the window, given the coordinates of the pixel around which the window is to be made.

function [iLow, iHigh, jLow, jHigh] = WindowCorners(img, windowSize, centerI, centerJ)  
 [maxI, maxJ] = size(img);  
 split = floor(windowSize/2);  
 iLow = max(centerI - split, 1); iHigh = min(maxI, centerI + split);  
 jLow = max(centerJ - split, 1); jHigh = min(maxJ, centerJ + split);  
end

Spatial gaussian mask – LTI gaussian filter with the centre of the gaussian at a given point (center\_i, center\_j). Simply compute the weights by definition, then normalize before returning.

function spatial\_mask = spacial\_gaussian\_filter(window, sigma\_space, center\_i, center\_j)  
 [m,n] = size(window);  
 %use separability  
 u = zeros(1,m);  
 for i = 1:m  
 u(i) = exp(-((i-center\_i)^2)/(2\*sigma\_space\*sigma\_space));  
 end  
 v = zeros(1,n);  
 for j = 1:n  
 v(j) = exp(-((j-center\_j)^2)/(2\*sigma\_space\*sigma\_space));  
 end  
 spatial\_mask = transpose(u)\*v;  
 spatial\_mask = spatial\_mask./sum(spatial\_mask,'all');  
end

Read off the intensity kernel outputs from the matrix store\_intensity\_gaussian, given the intensity at the current pixel of the window and the centre pixel of the window. Note that this has been vectorized in the bilateral filtering loop.

function intensity\_mask = intensity\_gaussian\_filter(pixel\_intensity, center\_intensity, store\_intensity\_gaussians)  
 x = max(1, round(pixel\_intensity)+1);  
 y = max(1, round(center\_intensity)+1);  
 intensity\_mask = store\_intensity\_gaussians(x,y);  
end

[*Published with MATLAB® R2020a*](https://www.mathworks.com/products/matlab)