CS663, Assignment 2 **Yash Sanghvi Pradyot Prakash**
Instructor: Prof. Suyash Awate 13D070042, 1300500XX
Question 3: Patch Based Bilateral Filtering

Problem Statement

(50 points) Edge-preserving Smoothing using Patch-Based Filtering.
Input image: 3/data/barbara.mat.
Redo the previous problem using patch-based filtering. If you think your code takes too long to run, (i) resize the image by subsampling by a factor of 2 along each dimension, after applying a Gaussian blur of standard deviation around 0.66 pixel width and (ii) apply the filter to the resized image.
Use 99 patches. Use a Gaussian, or clipped Gaussian, weight function on the patches to make the patch more isotropic (as compared to a square patch). Note: this will imply neighbor-location weighted distances between patches. For filtering pixel p, use patches centered at pixels q that lie within a window of size approximately 25 25 around p.

1. Write a function myPatchBasedFiltering.m to implement this.
2. Show the original, corrupted, and filtered versions side by side, using the same (gray) colourmap.
3. Show the mask used to make patches isotropic, as an image.
4. Report the optimal parameter value found, say σ along with the optimal RMSD.
5. Report RMSD values for filtered images obtained with (i) 0.9σ and (ii) 1.1σ , with all other parameter values unchanged.

Code

(i) Code for Patch Based Filter

```
function [ outputImage ] = myPatchBasedFiltering( inputImage, W, P,
    Sigma_I)
%UNTITLED3 Summary of this function goes here
[SizeX, SizeY] = size(inputImage);
outputImage = 255*ones(SizeX,SizeY);
h = waitbar(0, 'Patch Based Filtering in Progress...');

for i = 1:SizeX
    for j = 1:SizeY
        WindowX_L = max(1,i-W);
        WindowX_U = min(SizeX,i+W);
        WindowY_L = max(1,j-W);
        WindowY_U = min(SizeY,j+W);

        Window = inputImage(WindowX_L:WindowX_U,WindowY_L:WindowY_U)
            ;
        [WinSizeX, WinSizeY] = size(Window);
        Weight_Window = ones(WinSizeX,WinSizeY);
```

```
17
18          SelfPatchX_L = max(1,i-P);
19          SelfPatchX_U = min(SizeX,i+P);
20          SelfPatchY_L = max(1,j-P);
21          SelfPatchY_U = min(SizeY,j+P);
22
23          SelfPatch = inputImage(SelfPatchX_L:SelfPatchX_U,
                SelfPatchY_L:SelfPatchY_U);
24          [SelfPatchX, SelfPatchY] = size(SelfPatch);
25          for k = 1:WinSizeX
26              for l = 1:WinSizeY
27                  PatchX_L = max(1,k-P);
28                  PatchX_U = min(WinSizeX,k+P);
29                  PatchY_L = max(1,l-P);
30                  PatchY_U = min(WinSizeY,l+P);
31                  Patch = Window(PatchX_L:PatchX_U,PatchY_L:PatchY_U);
32                  [PatchSizeX, PatchSizeY] = size(Patch);
33                  CroppedPatch = Patch(1:min(PatchSizeX,SelfPatchX),1:
                        min(PatchSizeY,SelfPatchY));
34                  SelfCroppedPatch = SelfPatch(1:min(PatchSizeX,
                        SelfPatchX),1:min(PatchSizeY,SelfPatchY));
35                  Diff = (SelfCroppedPatch-CroppedPatch).*(
                        myGenerateGaussian(min(PatchSizeX,SelfPatchX),min
                        (PatchSizeY,SelfPatchY),P/3));
36                  Weight_Window(k,l) = sum(sum(Diff.^2))/(min(
                        PatchSizeX,SelfPatchX)*min(PatchSizeY,SelfPatchY)
                        );
37
38              end
39
40          end
41          Weight_Window = exp(-(Weight_Window)/(2*(Sigma_I^2)));%(
                myGenerateGaussian(WinSizeX,WinSizeY,5));
42  %       Weight_Window = myGenerateGaussian(WinSizeX,WinSizeY,Sigma_X);
43          Weight_Window = Weight_Window/sum(sum(Weight_Window));
44          outputImage(i,j) = sum(sum(Window.*Weight_Window));
45
46      end
47      waitbar(double(i*j)/double(SizeX*SizeY));
48  end
49  close(h);
50  end
```

(ii) Main Script

```
1  %% MyMainScript
2
3  tic;
4  %% Your code here
5  im = load('..\data\barbara');
```

```
src = 2*im.imageOrig;
src_shrink = myShrinkImageByFactorD(src,2);
% Calculate the Standard Deviation of the image
Sigma = 0.05*(max(max(src_shrink))-min(min(src_shrink)));
[sizeX, sizeY] = size(src);
noise = Sigma*randn([sizeX,sizeY]);
Corrupted = src + noise;
Corrupted_shrink = myShrinkImageByFactorD(Corrupted,2);

% myShowImage(uint8(src));
% title('Original Image');

% myShowImage(uint8(Corrupted));
% title('Corrupted Image');
% Intial RMSD between original and corrupted image
% PRevious 7, 3

G = fspecial('gaussian',2,0.66);
Corrupted_shrink = imfilter(Corrupted_shrink,G,'replicate');
Inital_RMSD = myRMSD(src_shrink,Corrupted_shrink)
BilateralFilter_Shrink = myPatchBasedFiltering(Corrupted_shrink, 12,
    4, 1.08);
myShowImage(uint8(src));
title('Source Image');

Final_RMS = myRMSD(src_shrink,BilateralFilter_Shrink)
BilateralFilter = myBilinearInterpolation(BilateralFilter_Shrink,
   sizeX,sizeY);
myShowImage(uint8(BilateralFilter));
title('Corrected Image');

toc;
```

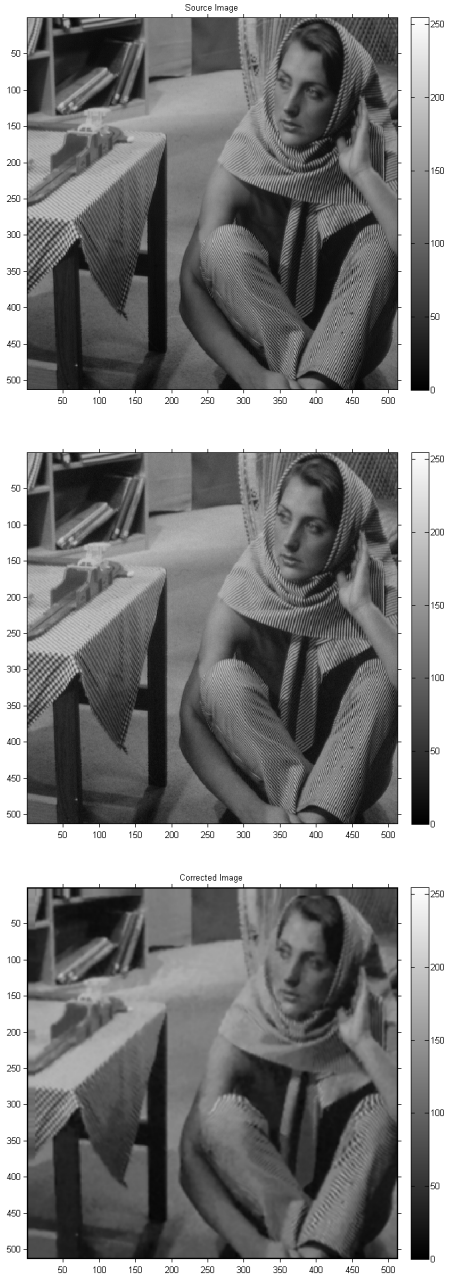
Explanation

We go through each pixel 'p' in the image as a part of two nested 'for' loops in i and j. For each pixel 'p', we choose a window of length $(2W+1)$ X $(2W+1)$. For the windows lying outside of the image, i.e close to the edges and corners, we simply take a chopped version of the window. For each pixel 'q' in the window determined for pixel 'p', We take a patch of size $(2P+1)$ X $(2P+1)$ and calculate the difference matrix with a similar $(2P+1)$ X $(2P+1)$ patch around pixel 'p'.Both the patches are made isotropic by multiplying them by a Gaussian window of variance size, one third the window length (decided by trial and error). In boundary conditions i.e. where the patches or section of patch lying outside the image, we take the chopped version of the both the patches (to be keep both of their dimensions same).
The mean sum of squares for the matrix entries is taken and the Gaussian i.e. $e^{-||D||^2/2\sigma^2}$ is allotted as the weight for the pixel 'q'. Here $||D||$ represents the mean of sum of squared differences and σ is a tuning parameter. Thus the weight s for the entire window of pixel 'p' are normalized so that sum of weights is 1. Using these weights we find the filtered value for

the pixel 'p'.

Other Implementation Details The corrupted image was obtained by taking source image and adding a random white gaussian noise to it of variance equal to 5% of the range of the image. To speed up the process, the corrupted image was scaled down by a factor of 2 and bilateral filtering was applied to it. Later, the output image's dimensions were doubled to the original size of source.

Result Images

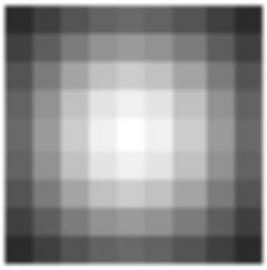


(Top) Original Image **(Centre)** Corrupted Image **(Bottom)** Image after applying Patch

based Bilateral Filter of optimum σ to minimum RMSD between the source image and the image obtained after bilateral filtering.

Optimal Parameters

Note the optimal parameters are $\sigma^* = \mathbf{1.125}, RMSD = \mathbf{15.14}$
(i) RMSD at $0.9\sigma^*$ (i.e. 1.08) = **15.223**
(ii) RMSD at $1.1\sigma^*$ (i.e. 1.35) = **15.209**



Gaussian Kernel, Dimension 9X9, $\sigma = 3$ used for making the patches isotropic