16D170005_16D100012_16D070001_filtering

by Bhishma Dedhia

Submission date: 21-Aug-2018 02:29AM (UTC+0800)

Submission ID: 991588042

File name: turnitin.docx (25.04K)

Word count: 1719

Character count: 15748

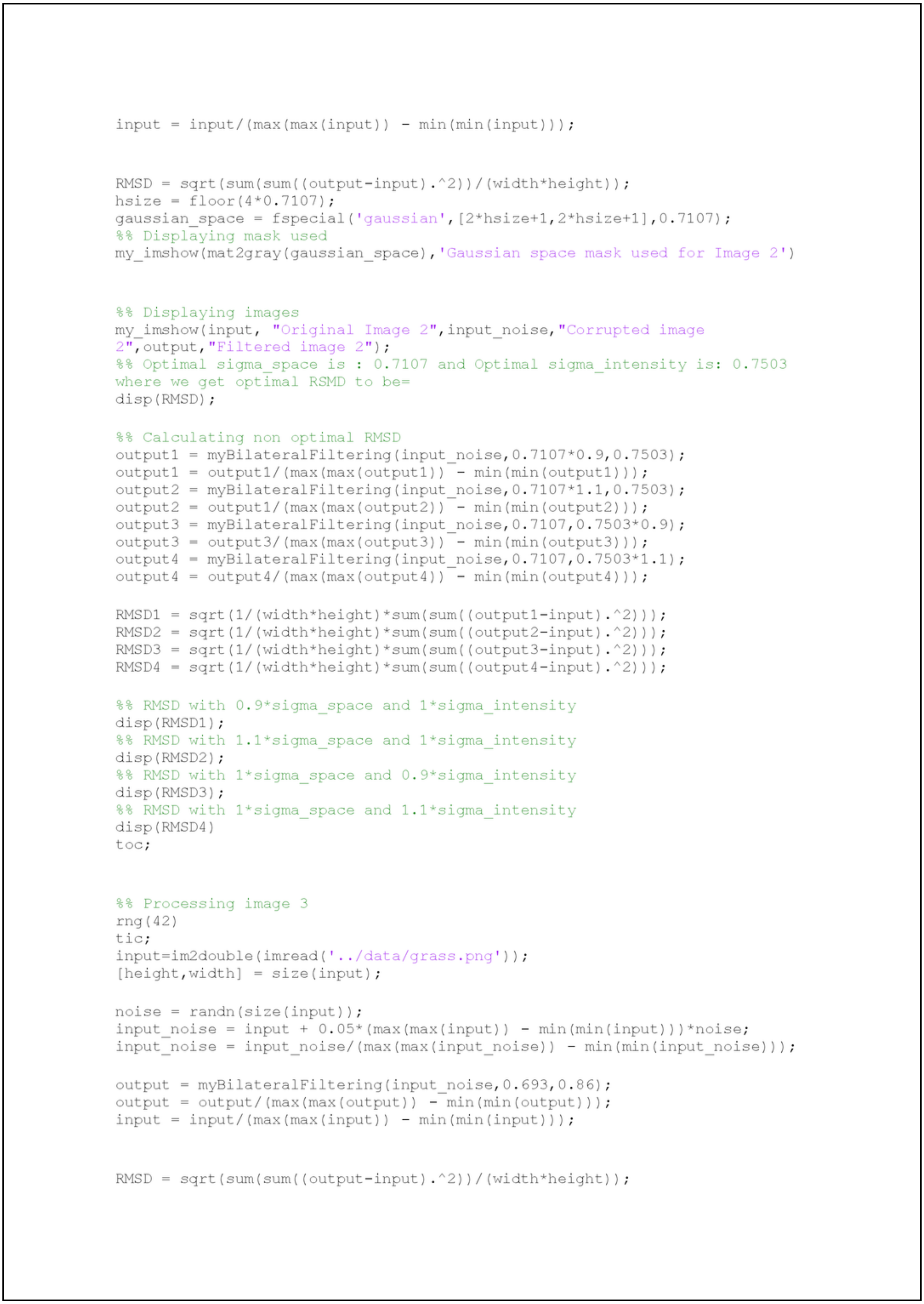
Q1) Unsharp Masking:

```
function output = myUnsharpMasking(input,sigma,hsize,scale)
%Sharpening images using unsharp mask
    gaussian_filt = fspecial('gaussian',hsize,sigma);
    input_blur = imfilter(input,gaussian_filt,'replicate','conv');
    unsharp_input = input - input_blur;
    output = scale*unsharp_input + input;
end


function output = myLinearContrastStretching(input,arg)
%Linear Contrast Stretching
    output = input;
    if(arg=='g')
        minimum =min(min(input));
        maximum =max(max(input));
        output =
cell2mat(arrayfun(@(z)pwl(minimum,maximum,z),input,'UniformOutput',false));
    else
        min_r =min( min(input(:,:,1)));%Minimum of red channel
        max_r =max( max(input(:,:,1)));%Maximum of red channel
        output(:,:,1) =
cell2mat(arrayfun(@(z)pwl(min_r,max_r,z),input(:,:,1),'UniformOutput',false
));

        min_g =min( min(input(:,:,2)));%Minimum of green channel
        max_g = max(max(input(:,:,2)));%Maximum of green channel
        output(:,:,2) =cell2mat(
arrayfun(@(z)pwl(min_g,max_g,z),input(:,:,2),'UniformOutput',false));

        min_b =min( min(input(:,:,3)));%Minimum of blue channel
        max_b = max(max(input(:,:,3)));%Maximum of blue channel
        output(:,:,3) =
cell2mat(arrayfun(@(z)pwl(min_b,max_b,z),input(:,:,3),'UniformOutput',false
));
    end
end
function op_intensity = pwl(min,max,in_intensity) %Mapping from [min,max]
to [0,255]
        in_intensity = double(in_intensity);
        x1 = double(min);
        y1 = 0;
        x2 = double(max);
        y2 = 255;

        op_intensity = (in_intensity-x1)*((y2 - y1)./(x2 - x1));
end
%% MyMainScript

tic;
%% Processing first image
%reading input image1
  storedStructure = load('../data/lionCrop.mat');
  input = storedStructure.imageOrig;
  input = (input-min(input(:)))/(max(input(:))-min(input(:)));
% Sharpening Image
% Optimal parameters: sigma = 1.5, hsize=5, scale=2
  output = myUnsharpMasking(input,1.5,5,2);
  output = (output-min(output(:)))/(max(output(:))-min(output(:)));
  my_imshow(input,'Input Image',output,'Sharpened Image');
  save

toc;

%% Processing second image

%reading input image1
  storedStructure = load('../data/superMoonCrop.mat');
  input = storedStructure.imageOrig;
    input = (input-min(input(:)))/(max(input(:))-min(input(:)));
%Sharpening image
% Optimal parameters: sigma = 1, hsize=7, scale=2
  output = myUnsharpMasking(input,1,7,2);
    output = (output-min(output(:)))/(max(output(:))-min(output(:)));
  my_imshow(input,'Input Image',output,'Sharpened Image');
% toc;
```

Q2) Bilateral Filtering:

```
function output = myBilateralFiltering(input,sigma_space,sigma_int)
%%Code for Bilateral filtering
    %Creating a gaussian weight in space common for all pixel windows
    hsize = floor(4*sigma_space);
    c1= sqrt(2*pi)*sigma_int;
    gaussian_space =
fspecial('gaussian',[2*hsize+1,2*hsize+1],sigma_space);
    output =zeros(size(input));
    [height,width] = size(input);
    input1 =  padarray(input, [hsize hsize], inf);%Infinite to avoid
interference in intensity weights due to padding
    for i = 1+hsize:width+hsize
        for j = 1+hsize:height+hsize
            window = input1( i-hsize:i+hsize, j-hsize:j+hsize);
            %Finding the intensity gaussian weights
            gaussian_int = exp(-(input1(i,j)-
window).^2/(2*sigma_int^2))/c1;
            %calculating output pixel value
            window(window == inf) = 0;%Setting padding to 0 while
accounting for weights
            numerator = sum(gaussian_int.*gaussian_space.*window);
            denominator = sum(gaussian_int.*gaussian_space);
            output(i-hsize,j-hsize) = numerator/denominator;
        end
    end
 end

%MyMainScript
rng(42);
tic;
%% Processing image 1

storedStructure = load('../data/barbara.mat');
input = storedStructure.imageOrig;
[height,width] = size(input);

noise = randn(size(input));
input_noise = input + 0.05*(max(max(input)) - min(min(input)))*noise;
input_noise = input_noise/(max(max(input_noise)) - min(min(input_noise)));
```

```
output = myBilateralFiltering(input_noise,1.178,0.1720);
output = output/(max(max(output)) - min(min(output)));
input = input/(max(max(input)) - min(min(input)));


RMSD = sqrt(sum(sum((output-input).^2))/(width*height));

hsize = floor(4*1.178);
gaussian_space = fspecial('gaussian',[2*hsize+1,2*hsize+1],1.178);
%% Displaying mask used
my_imshow(mat2gray(gaussian_space),'Gaussian space mask used for Image 1')


%% Displaying images
my_imshow(input, "Original Image 1",input_noise,"Corrupted image
1",output,"Filtered image 1");
%% Optimal sigma_space is : 1.178 and Optimal sigma_intensity is: 0.1720,
where we get optimal RSMD to be=
disp(RMSD);
%% Calculating non optimal RMSD
output1 = myBilateralFiltering(input_noise,1.178*0.9,0.1720);
output1 = output1/(max(max(output1)) - min(min(output1)));
output2 = myBilateralFiltering(input_noise,1.178*1.1,0.1720);
output2 = output1/(max(max(output2)) - min(min(output2)));
output3 = myBilateralFiltering(input_noise,1.178,0.1720*0.9);
output3 = output3/(max(max(output3)) - min(min(output3)));
output4 = myBilateralFiltering(input_noise,1.178,0.1720*1.1);
output4 = output4/(max(max(output4)) - min(min(output4)));

RMSD1 = sqrt(1/(width*height)*sum(sum((output1-input).^2)));
RMSD2 = sqrt(1/(width*height)*sum(sum((output2-input).^2)));
RMSD3 = sqrt(1/(width*height)*sum(sum((output3-input).^2)));
RMSD4 = sqrt(1/(width*height)*sum(sum((output4-input).^2)));

%% RMSD with 0.9*sigma_space and 1*sigma_intensity
disp(RMSD1);
%% RMSD with 1.1*sigma_space and 1*sigma_intensity
disp(RMSD2);
%% RMSD with 1*sigma_space and 0.9*sigma_intensity
disp(RMSD3);
%% RMSD with 1*sigma_space and 1.1*sigma_intensity
disp(RMSD4)

toc;


%% Processing image 2
rng(42)
tic;
input=im2double(imread('../data/honeyCombReal.png'));
[height,width] = size(input);

noise = randn(size(input));
input_noise = input + 0.05*(max(max(input)) - min(min(input)))*noise;
input_noise = input_noise/(max(max(input_noise)) - min(min(input_noise)));

output = myBilateralFiltering(input_noise,0.7107,0.7503);
output = output/(max(max(output)) - min(min(output)));
```



```
input = input/(max(max(input)) - min(min(input)));


RMSD = sqrt(sum(sum((output-input).^2))/(width*height));
hsize = floor(4*0.7107);
gaussian_space = fspecial('gaussian',[2*hsize+1,2*hsize+1],0.7107);
%% Displaying mask used
my_imshow(mat2gray(gaussian_space),'Gaussian space mask used for Image 2')


%% Displaying images
my_imshow(input, "Original Image 2",input_noise,"Corrupted image
2",output,"Filtered image 2");
%% Optimal sigma_space is : 0.7107 and Optimal sigma_intensity is: 0.7503
where we get optimal RSMD to be=
disp(RMSD);

%% Calculating non optimal RMSD
output1 = myBilateralFiltering(input_noise,0.7107*0.9,0.7503);
output1 = output1/(max(max(output1)) - min(min(output1)));
output2 = myBilateralFiltering(input_noise,0.7107*1.1,0.7503);
output2 = output1/(max(max(output2)) - min(min(output2)));
output3 = myBilateralFiltering(input_noise,0.7107,0.7503*0.9);
output3 = output3/(max(max(output3)) - min(min(output3)));
output4 = myBilateralFiltering(input_noise,0.7107,0.7503*1.1);
output4 = output4/(max(max(output4)) - min(min(output4)));

RMSD1 = sqrt(1/(width*height)*sum(sum((output1-input).^2)));
RMSD2 = sqrt(1/(width*height)*sum(sum((output2-input).^2)));
RMSD3 = sqrt(1/(width*height)*sum(sum((output3-input).^2)));
RMSD4 = sqrt(1/(width*height)*sum(sum((output4-input).^2)));

%% RMSD with 0.9*sigma_space and 1*sigma_intensity
disp(RMSD1);
%% RMSD with 1.1*sigma_space and 1*sigma_intensity
disp(RMSD2);
%% RMSD with 1*sigma_space and 0.9*sigma_intensity
disp(RMSD3);
%% RMSD with 1*sigma_space and 1.1*sigma_intensity
disp(RMSD4)
toc;


%% Processing image 3
rng(42)
tic;
input=im2double(imread('../data/grass.png'));
[height,width] = size(input);

noise = randn(size(input));
input_noise = input + 0.05*(max(max(input)) - min(min(input)))*noise;
input_noise = input_noise/(max(max(input_noise)) - min(min(input_noise)));

output = myBilateralFiltering(input_noise,0.693,0.86);
output = output/(max(max(output)) - min(min(output)));
input = input/(max(max(input)) - min(min(input)));


RMSD = sqrt(sum(sum((output-input).^2))/(width*height));
```

```
hsize = floor(4*0.693);
gaussian_space = fspecial('gaussian',[2*hsize+1,2*hsize+1],0.693);
%% Displaying mask used
my_imshow(mat2gray(gaussian_space),'Gaussian space mask used for Image 3')


%% Displaying images
my_imshow(input, "Original Image 3",input_noise,"Corrupted image
3",output,"Filtered image 3");
%% Optimal sigma_space is : 0.693 and Optimal sigma_intensity is: 0.86,
where we get optimal RSMD to be=
disp(RMSD);

%% Calculating non optimal RMSD
output1 = myBilateralFiltering(input_noise,0.693*0.9,0.86);
output1 = output1/(max(max(output1)) - min(min(output1)));
output2 = myBilateralFiltering(input_noise,0.693*1.1,0.86);
output2 = output1/(max(max(output2)) - min(min(output2)));
output3 = myBilateralFiltering(input_noise,0.693,0.86*0.9);
output3 = output3/(max(max(output3)) - min(min(output3)));
output4 = myBilateralFiltering(input_noise,0.693,0.86*1.1);
output4 = output4/(max(max(output4)) - min(min(output4)));

RMSD1 = sqrt(1/(width*height)*sum(sum((output1-input).^2)));
RMSD2 = sqrt(1/(width*height)*sum(sum((output2-input).^2)));
RMSD3 = sqrt(1/(width*height)*sum(sum((output3-input).^2)));
RMSD4 = sqrt(1/(width*height)*sum(sum((output4-input).^2)));

%% RMSD with 0.9*sigma_space and 1*sigma_intensity
disp(RMSD1);
%% RMSD with 1.1*sigma_space and 1*sigma_intensity
disp(RMSD2);
%% RMSD with 1*sigma_space and 0.9*sigma_intensity
disp(RMSD3);
%% RMSD with 1*sigma_space and 1.1*sigma_intensity
disp(RMSD4)
toc;

function my_imshow(varargin)

    numberColours = 200;
    colorScale = [[0:1/(numberColours-1):1]',[0:1/(numberColours-
1):1]',[0:1/(numberColours-1):1]'];
    figure('NumberTitle','off', 'position', [50, 50, 1200, 400]);

    num = nargin/2;
    for k = 1:num
        subplot(1,num,k);
        imagesc(varargin{2*k-1});
        title(varargin{2*k}, 'Fontsize', 12, 'Fontname', 'Cambria');
        % truesize;
        colormap(colorScale);
        daspect([1,1,1]);
        axis tight;
        colorbar;

    end
end
```

Q3) %% MyMainScript

```
%% Image 1: Barbara

% Reading inputB
inputB = load('../data/barbara.mat');
inputB = double(inputB.imageOrig);

%% Corrupting image with noise
[xb,yb] = size(inputB);
sd_b = 0.05*(max(max(inputB)) - min(min(inputB)));
corrupted_B = inputB + sd_b*randn(xb,yb);
corrupted_B_shrunk = corrupted_B(1:2:end,1:2:end);
inputB_s = inputB(1:2:end, 1:2:end);
% Patch-based filtering with optimal parameters
tic;
h_b = 1.25;
filtered_B = myPatchBasedFiltering(corrupted_B_shrunk,9,25,1.5,h_b);
RMSD_b = sqrt(sum(sum((filtered_B-inputB_s).^2))/(xb*yb));
my_imshow(inputB_s, 'inputB image', corrupted_B_shrunk, 'Corrupted image',
filtered_B, 'Filtered image');
toc;
%%% Optimal parameter values:
% The optimal value of the standard deviation (SD) is 1.25
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_b);

% Note:
% The mask has been scaled to the range [0, 255] for the purpose of
display.

mask = fspecial('gaussian', 9, 1.5);
my_imshow(mask, 'Mask to make patches isotropic')
% Patch-based filtering with sub-optimal parameters

% 0.9 SD
tic;
filtered_b_9 = myPatchBasedFiltering(corrupted_B_shrunk,9,25,1.5,h_b*0.9);
RMSD_b9 = sqrt(sum(sum((filtered_b_9-inputB_s).^2))/(xb*yb));
my_imshow(inputB_s, 'inputB image', corrupted_B_shrunk, 'Corrupted image',
filtered_b_9, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_b9);
toc;
% 1.1 SD
filtered_b_11 = myPatchBasedFiltering(corrupted_B_shrunk,9,25,1.5,h_b*1.1);
RMSD_b11 = sqrt(sum(sum((filtered_b_11-inputB_s).^2))/(xb*yb));
my_imshow(inputB_s, 'inputB image', corrupted_B_shrunk, 'Corrupted image',
filtered_b_11, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_b11);


%%% Note:
% Undersampling, by spatio-gaussian sampling, has not been done
% Instead, a spatio-Gaussian thresholding has been done to ensure patches
% far from the the pixel of interest have less contribution, and patches
% are isotropic

% Using random patch selection
% Undersample the image using spatial gaussian distribution. Poorer RSMD
% but large drop in execution time
outputRANDOM = myRandomPatchBasedFilter(corrupted_B,9,25,1.5,h_b,100);
```

```
[x,y] = size(corrupted_B);
RMSD_b_ran = sqrt(sum(sum((outputRANDOM-inputB).^2))/(x*y));
my_imshow(inputB, 'inputB image', corrupted_B, 'Corrupted image',
outputRANDOM, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_b_ran);



%% Image 2: Grass Noisy

% Reading inputB
inputG = im2double(imread('../data/grass.png'));

% Corrupting image with noise
[xg,yg] = size(inputG);
sd_g = 0.05*(max(max(inputG)) - min(min(inputG)));
corrupted_G = inputG + sd_g*randn(xg,yg);
corrupted_G = corrupted_G(1:2:end,1:2:end);
inputG = inputG(1:2:end,1:2:end);
%% Patch-based filtering with optimal parameters
tic;
h_g = 1.25;
filtered_G = myPatchBasedFiltering(corrupted_G,9,25,1.5,h_g);
RMSDg = sqrt(sum(sum((filtered_G-inputG).^2))/(xg*yg));
my_imshow(inputG, 'input image', corrupted_G, 'Corrupted image',
filtered_G, 'Filtered image');
toc;
%%% Optimal parameter values:
% The optimal value of the standard deviation (SD) is h
fprintf('The correspoding RMSD is %2.3f.\n', RMSDg);

%% Note:
% The mask has been scaled to the range [0, 255] for the purpose of
display.

mask = fspecial('gaussian', 9, 1.5);
my_imshow(mask, 'Mask to make patches isotropic')
%% Patch-based filtering with sub-optimal parameters

%% 0.9 SD
tic;
filtered_g_9 = myPatchBasedFiltering(corrupted_G,9,25,1.5,h_g*0.9);
RMSD_g9 = sqrt(sum(sum((filtered_g_9-inputG).^2))/(xg*yg));
my_imshow(inputG, 'input image', corrupted_G, 'Corrupted image',
filtered_g_9, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_g9);
toc;
%% 1.1 SD
filtered_g_11 = myPatchBasedFiltering(corrupted_G,9,25,1.5,h_g*1.1);
RMSD_g11 = sqrt(sum(sum((filtered_g_11-inputG).^2))/(xg*yg));
my_imshow(inputG, 'input image', corrupted_G, 'Corrupted image',
filtered_g_11, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_g11);


% Image 3: Honey Comb

%%Reading input
inputH = im2double(imread('../data/honeyCombReal.png'));
```

```
%% Corrupting image with noise
[xh,yh] = size(inputH);
sd_h = 0.05*(max(max(inputH)) - min(min(inputH)));
corrupted_H = inputH + sd_g*randn(xh,yh);
corrupted_H = corrupted_H(1:2:end,1:2:end);
inputH = inputH(1:2:end,1:2:end);
%% Patch-based filtering with optimal parameters
tic;
h_h = 1.25;
filtered_H = myPatchBasedFiltering(corrupted_H,9,25,1.5,h_h);
RMSD_h = sqrt(sum(sum((filtered_H-inputH).^2))/(xh*yh));
my_imshow(inputH, 'input image', corrupted_H, 'Corrupted image',
filtered_H, 'Filtered image');
toc;
%%% Optimal parameter values:
% The optimal value of the standard deviation (SD) is h
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_h);

%% Note:
% The mask has been scaled to the range [0, 255] for the purpose of
display.

mask = fspecial('gaussian', 9, 1.5);
my_imshow(mask, 'Mask to make patches isotropic')
%% Patch-based filtering with sub-optimal parameters

%% 0.9 SD
tic;
filtered_H_9 = myPatchBasedFiltering(corrupted_H,9,25,1.5,h_h*0.9);
RMSD_h9 = sqrt(sum(sum((filtered_H_9-inputH).^2))/(xh*yh));
my_imshow(inputH, 'input image', corrupted_H, 'Corrupted image',
filtered_H_9, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_h9);
toc;
%% 1.1 SD
filtered_h_11 = myPatchBasedFiltering(corrupted_H,9,25,1.5,h_h*1.1);
RMSD_h11 = sqrt(sum(sum((filtered_h_11-inputH).^2))/(xh*yh));
my_imshow(inputH, 'input image', corrupted_H, 'Corrupted image',
filtered_h_11, 'Filtered image');
fprintf('The correspoding RMSD is %2.3f.\n', RMSD_h11);

function output = myPatchBasedFiltering(input,ps,ws,a,h)
% Patch based filtering
% Neumann Boundary Padding
P = (ps-1)/2;
W = (ws-1)/2;
pad = P+W;
padded_input = padarray(input, [pad pad], 'replicate', 'both');

prob_Gaussian = fspecial('gaussian', ws, 1.6);
iso_gaussian = fspecial('gaussian', ps, a);
iso_gaussian = iso_gaussian/sum(iso_gaussian(:));

output = zeros(size(input));
    for i = pad+1 : pad+size(input,1)
        for j = pad+1: pad+size(input,2)
            weights = zeros(ws,ws);
            centre_patch = padded_input( (i-P : i+P),(j-P : j+P) );
            iso_centre_patch = centre_patch.*iso_gaussian;
```

```
%              window = padded_input( (i-(ws-1)/2 : i+(ws-1)/2),(j-(ws-1)/2
: j+(ws-1)/2) );
            for m = i-W:i+W-1
                for n = j-W:j+W-1
                    if(prob_Gaussian(m+1+W-i,n+1+W-j) >(5e-6)*rand())
                        % randomly selecting patches from spatial Gaussian
Isotropic (slide 44)
                        % Constant tuned to ensure enough patches,
                        % while keeping compute time low

                        %finding patch around (m,n)
                        patch = padded_input( (m-P : m+P),(n-P : n+P) );
                        iso_patch = patch.*iso_gaussian;
                        weights(m-i+W+1,n-j+W+1) = exp(-
1*sum(sum((iso_patch - iso_centre_patch).^2))/(2*h^2) );
                    end
                end
            end

            weight = double(weights)/double(sum(weights(:)));
            window = padded_input((i-W:i+W),(j-W:j+W));

             output(i-pad,j-pad) = sum(sum(window.*weight));
        end
    end
end

function output = myRandomPatchBasedFilter(input,ps,ws,a,h,num_samples)
% Patch based filtering
% Neumann Boundary Padding
P = (ps-1)/2;
W = (ws-1)/2;
pad = P+W;
padded_input = padarray(input, [pad pad], 'replicate', 'both');

iso_gaussian = fspecial('gaussian', ps, a);
iso_gaussian = iso_gaussian/sum(iso_gaussian(:));
sigma = ws/3;
output = zeros(size(input));
    for i = pad+1 : pad+size(input,1)
        for j = pad+1: pad+size(input,2)
            weights = zeros(ws,ws);
            centre_patch = padded_input( (i-P : i+P),(j-P : j+P) );
            iso_centre_patch = centre_patch.*iso_gaussian;

            coords = randn([num_samples,2])*sigma;
            coords(:,1) = coords(:,1)+i;
            coords(:,2) = coords(:,2)+j;
            denom = 0;
            nr = 0;
            for n = 1:num_samples
               a = abs(coords(n,:)-[i,j]);
               if abs(coords(n,1)-i) < W
                   if abs(coords(n,2)-j) < W
                   x=floor(coords(n,1));
               y=floor(coords(n,2));
               patch = padded_input( (x-P : x+P),(y-P : y+P) );
               iso_patch = patch.*iso_gaussian;
               weight = exp(-1*sum(sum((iso_patch -
iso_centre_patch).^2))/(2*h^2) );
```

```
               denom = denom+weight;
               nr = nr+weight*padded_input(x,y);
                   end
               end

              end

            output(i-pad,j-pad) =  nr/denom;
        end
    end
end
```

16D170005_16D100012_16D070001_filtering

ORIGINALITY REPORT

3%	2%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	zenith.ngdc.noaa.gov Internet Source	1%
2	Submitted to University of Southern California Student Paper	1%
3	www.challenge.nm.org Internet Source	<1%
4	arxiv.org Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

16D170005_16D100012_16D070001_filtering