Assignment 4 Solutions

CS283, Computer Vision

1. (a) One possible implementation of the affine factorization algorithm is included below.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 4
% Question 1, part a
%
% Affine factorization
%
function [M, T, X] = AffineFactorization(x)

if (nargin ~= 1),
    error('Invalid number of inputs');
end;

if (size(x, 1) < 4) || (size(x, 2) ~= 2) || (size(x, 3) < 2),
    error('Input x has incorrect size');
end;

num_pts=size(x, 1);
num_ims=size(x, 3);

% compute camera translations
centroids = mean(x);
T = reshape(centroids, [num_ims * 2, 1]);

% register image coordinates
x = x - repmat(centroids, [num_pts, 1, 1]);

% construct measurement matrix
W = transpose(reshape(x, [num_pts, 2 * num_ims]));

% compute SVD, enforce rank 3, and assign output
[u, d, v] = svd(W);
M = [u(:, 1) * d(1, 1) u(:, 2) * d(2, 2) u(:, 3) * d(3, 3)];
X = v(:, 1:3);
```

We can use the following simple script to apply the above code on the provided data, and visualize the results. The resulting plot is shown in Fig. 1.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 4
% Question 1, part a
%
% Plot results of affine factorization
%
load ../data/corners.mat
[M T X] = AffineFactorization(corners);
figure;
plot3(X(:,1),X(:,2),X(:,3),'.'); axis equal;
```

We observe that, even though parallel lines are preserved, angles between planes are not. To understand this, consider what we are solving for, X and M. However, in the affine factorization algorithm, only their product MX is constrained. Therefore, for any invertible 3×3 matrix A, corresponding to an affine transformation, the algorithm cannot discriminate between MX and

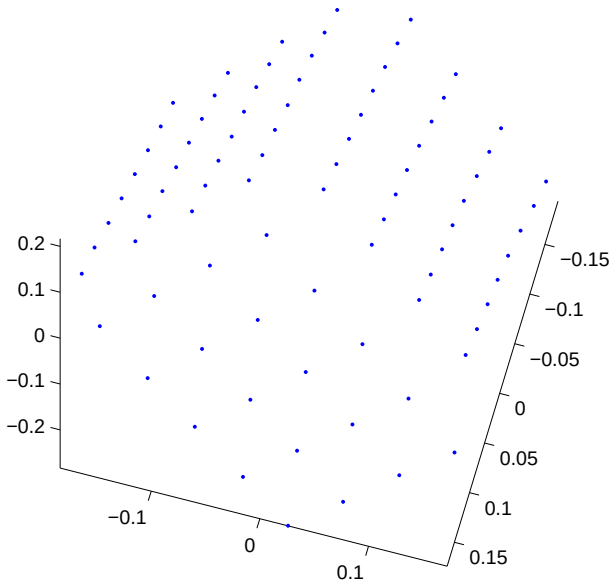

Figure 1: Recovered points using the affine factorization algorithm.

$(MA)\left(A^{-1}X\right)$. Therefore, the recovered points X may differ from the true world coordinates by some affine transformation A. Under affine transformations, parallelism is preserved but angles are not. This explains our observation.

(b) We use the following script to evaluate the RMS projection error.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 4
% Question 1, part b
%
% Calculate reconstruction RMS error
%
x = transpose(reshape(corners, 84, 14));
reconError = x - (M * transpose(X) + repmat(T, 1, 84));
nm = numel(reconError) / 2;
Erms = sqrt((1 / nm) * sum(reconError(:) .^ 2));
```

The RMS error is equal to about 1.42. Care needs to be taken when using RMS error, mean-square error, or other variants, to make sure exactly which type of error is meant and what formula is used.

(c) **Bonus question.** One possible implementation for performing the upgrade of the affine reconstruction of part (a) is included below. The recovered points are shown in Fig. 2.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 4
% Question 1, part c, bonus
%
```

```
% Affine factorization upgrade
%
function [Mout, Xout, A] = upgrade(Min, Xin)

% check inputs
if (nargin<2),
    error('Invalid number of inputs');
end;

if (size(Xin, 1) < 4) || (size(Xin, 2) ~= 3),
    error('Input Xin had incorrect size');
end;

num_pts = size(Xin, 1);
num_ims = size(Min, 1) / 2;

% allocate space for output
Mout = Min;
Xout = Xin;

% Compute affine A using least squares of Tomasi/Kanade 'metric constraints'
% with eye(3) for initialization.
options = optimset('Display', 'Iter',...
                   'TolFun', 1e-10,...
                   'MaxFunEvals', 1e4,...
                   'LargeScale', 'off');

Po=[1 0 0 0 1 0 0 0 1]';
P = lsqnonlin(@upgrade_fun, Po, [], [], options, Min);
A = reshape(P(1:9), [3, 3]);

% compute and apply scale change
Xt = transpose(A \ transpose(Xin));
d3 = mean(sqrt(sum(Xt .^ 2, 2)));
A = A * (eye(3) .* d3);

% adjust Min and Xin
Mout = Min * A;
Xout = transpose(A \ transpose(Xin));

end

%
% Loss function to be used by the optimization routine
%
function F = upgrade_fun(P, M)

num_ims = size(M, 1) / 2;

% rehape input
A = reshape(P, [3, 3]);

% compute vector of error terms
ZeroSkew = diag(M(1:2:end-1, :) * A * transpose(A) * transpose(M(2:2:end,:)));
KnownRatio = diag(M(1:2:end-1, :) * A * transpose(A) * transpose(M(1:2:end-1, ...
    :)))...
           - diag(M(2:2:end, :) * A * transpose(A) * transpose(M(2:2:end, :)));

F=[ZeroSkew; KnownRatio];

end
```

The reprojection error of the upgraded construction is about 1.42, almost the same as before. This is expected, if we consider what the upgrade of the affine reconstruction does. As we discussed earlier, performing affine factorization (or, equivalently, minimizing the RMS reprojection error) produces multiple solutions (MA) for the camera matrix and $\left(A^{-1}X\right)$ world coordinate matrix, for any non-singular 3×3 matrix A. In other words, all of these solutions have the same RMS

reprojection error. The upgraded reconstruction selects among these solutions the one that satisfies the constraints of "square pixels and zero skew". Though the upgraded solution is "better" in the sense that it has these additional desirable characteristics, in terms of the RMS reprojection error alone it is as good as any of the other solutions.

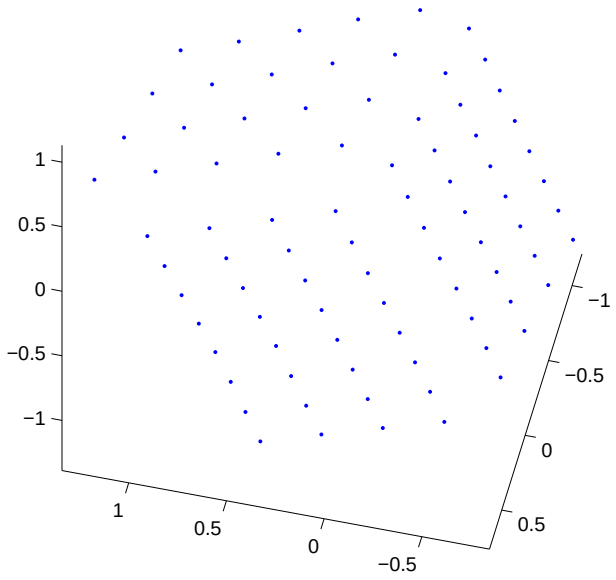

Figure 2: Recovered points after upgrading the result of the affine factorization algorithm.

2. (a) As the two-dimension (discrete) Fourier transform is separable, we can consider each dimension independently of the other. We discuss the x dimension, and the same apply identical to y. The corresponding real parts of I, for each $u = 0, \ldots, M-1$, have the form

$$\mathrm{Re}\left(\phi_u\left[x\right]\right) = \cos\left(u\frac{2\pi x}{M}\right),\ x = 0, \ldots, M-1 \tag{1}$$

From Eq. (1), we can see that the M-dimensional vector $\mathrm{Re}\left(\boldsymbol{\phi}_u\right)$ for each u is formed by evaluating the cosine function

$$h_u\left(x\right) = \cos\left(ux\right),\ x \in \left[0, 2\pi\right], \tag{2}$$

at intervals of length $\frac{2\pi}{M}$. Compare the possible values for x in Eq. (2) and Eq. (1); the difference is important: h_u is a continuous function, and $\mathrm{Re}\left(\boldsymbol{\phi}_u\right)$ is a vector created by *discretely sampling* h_u with *sampling frequency* M. Among all functions h_u, the one with the largest *continuous* frequency is that corresponding to the largest value of u, that is, $u = M$. We may be tempted to say that the same is true for the vectors $\mathrm{Re}\left(\boldsymbol{\phi}_u\right)$, but this would be incorrect: we need to consider the interaction between the continuous frequency of the function being sampled, and the sampling frequency. To visualize this, we take the case $M = 10$, and plot the 10 vectors $\mathrm{Re}\left(\boldsymbol{\phi}_u\right)$ in Fig. 3. To emphasize that these vectors arise from discretely sampling continuous trigonometric signals, but are not continuous themselves, we plot them using the command `stem` instead of the regular `plot`.

We can observe from this figure that the vector with the largest frequency is the one for $u_{\max} = 5$, where the vector is a sequence of alternating $+1$ and -1 values. The fact that $u_{\max} = \frac{M}{2}$, that

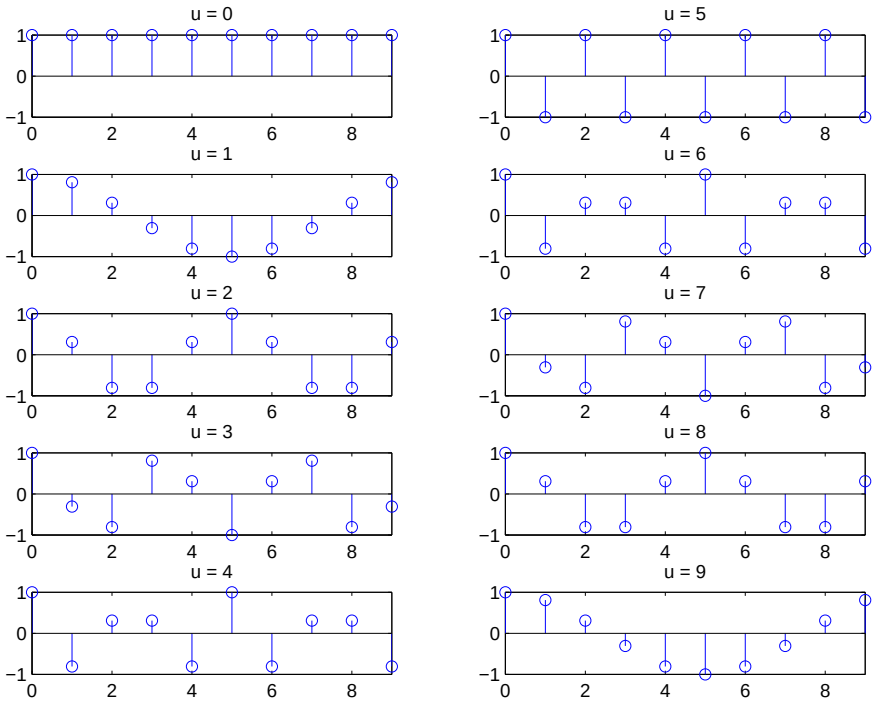

Figure 3: The vectors $\mathrm{Re}\,(\boldsymbol{\phi}_u)$, $u = 0, \ldots, 9$.

is, *half the sampling frequency* is no consequence; in fact there is a special name for this quantity, the *Nyquist frequency*. Also notable is that the vector $\mathrm{Re}\,(\boldsymbol{\phi}_0)$ is constant and equal to 1. The corresponding Fourier coefficient is the average of the signal $\boldsymbol{f}$, or in engineering terminology, its *DC compoment* ("DC" in this case coming from electrical engineering and standing for "direct current", as opposed "AC" or "alternating current").

Getting back to the problem set after this lenghty discussion, we have seen that when M is even, the highest frequency is at exactly $\frac{M}{2}$. When M is odd, through a similar reasoning we can see that the highest frequency is achieved at two values, $u = \frac{M-1}{2}$ and $u = \frac{M+1}{2}$ (we cannot achieve exactly the Nyquist frequency).

Going back to 2D, from the separability of the (discrete) Fourier transform, we arrive at four cases:

- if both M and N are even, the highest frequency is achieved at at $(u, v) = \left(\frac{M}{2}, \frac{N}{2}\right)$;
- if M is even and N is odd, the highest frequency is achieved at at $(u, v) = \left(\frac{M}{2}, \frac{N-1}{2}\right)$, and $(u, v) = \left(\frac{M}{2}, \frac{N+1}{2}\right)$;
- if M is odd and N is even, the highest frequency is achieved at at $(u, v) = \left(\frac{M-1}{2}, \frac{N}{2}\right)$, and $(u, v) = \left(\frac{M+}{2}, \frac{N}{2}\right)$; and finally
- if M and N are odd, the highest frequency is achieved at at $(u, v) = \left(\frac{M-1}{2}, \frac{N-1}{2}\right)$, $(u, v) = \left(\frac{M-1}{2}, \frac{N+1}{2}\right)$, $(u, v) = \left(\frac{M+1}{2}, \frac{N-1}{2}\right)$, and $(u, v) = \left(\frac{M+1}{2}, \frac{N+1}{2}\right)$.

(b) From the definition of the discrete Fourier transform, we have

$$\mathscr{F}\left[f(x,y)(-1)^{x+y}\right] = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)(-1)^{x+y}\exp\left(-j\frac{2\pi ux}{M}\right)\exp\left(-j\frac{2\pi vy}{N}\right) \tag{3}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left(j2\pi\left(\frac{x}{2}+\frac{y}{2}\right)\right)\exp\left(-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)\right) \tag{4}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left(-j2\pi\left(\frac{\left(u-\frac{M}{2}\right)x}{M}+\frac{\left(v-\frac{N}{2}\right)y}{N}\right)\right) \tag{5}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp\left(-j\frac{2\pi\left(u-\frac{M}{2}\right)x}{M}\right)\exp\left(-j\frac{2\pi\left(v-\frac{N}{2}\right)y}{N}\right) \tag{6}$$

$$= F\left(u-\frac{M}{2}, v-\frac{N}{2}\right), \tag{7}$$

where the last equality follows directly from the definition of the discrete Fourier transform.

3. We list the code for all the parts at the end of this question.

(a) We show in Fig. 4 both the magnitude of the centered DFT and its logarithm. The logarithm allows us to see the details of the frequency structure. In the magnitude picture, though, the very high frequency content corresponding to the periodic noise is easier to detect.

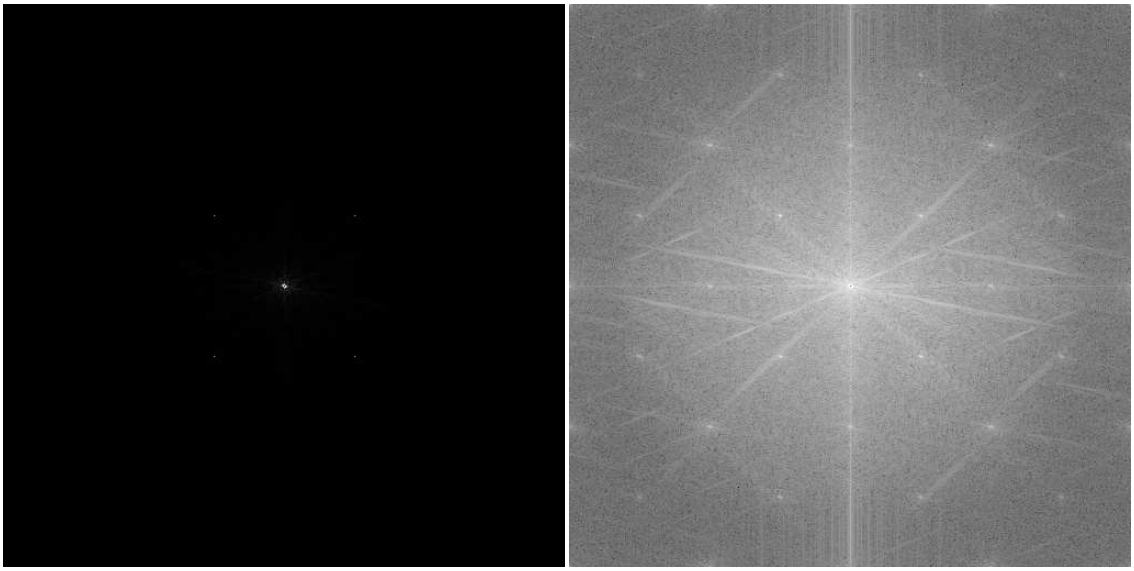

Figure 4: Magnitude (right) and logarithm of magnitude (left) of centered discrete Fourier transform of noisy image.

From Fig. 4, we can manually estimate the radial center D_0 to be approximately equal to 90.5. For the width w, we do not want to use a very large value, as then we will be removing frequency components that correspond to the original image and not to noise. On the other hand, a value of w that is too small reduces robustness, as we do not have exact values for D_0 and the noise frequency content. Any values between 3 and 10 would work well. Here, we choose $w = 8$ as a compromise. The resulting filter is shown in Fig. 5

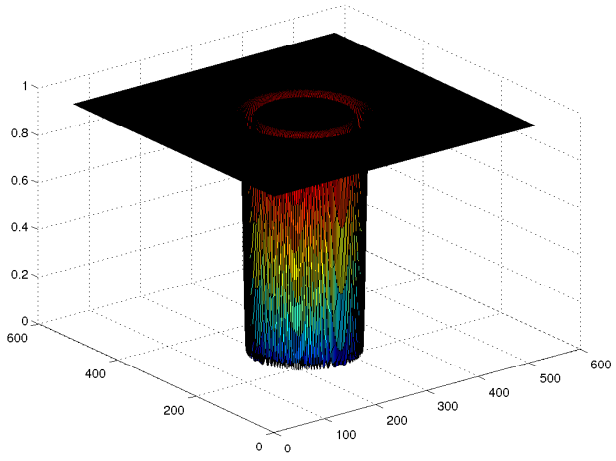

Figure 5: Transfer function of band-reject filter with radial center $D_0 = 90.5$ and width $w = 8$.

(b) The results of denoising using the procedure described by the exercise and the filter of the previous part are shown in Fig. 6. We see that the "noise", as captured in the difference between the original and the denoised image, exhibits a periodic structure, as suggested by its frequency content.

(c) By definition, the impulse response of a filter is the output we get when we use a delta function as its input. Remember that the convolution theorem tells us that the output of a filter with transfer function $H(u, v)$ given an input $f_{\text{in}}(x, y)$ can be obtained using two Fourier transforms, a forward and an inverse:

$$f_{\text{out}}(x, y) = \mathscr{F}^{-1}\left[H(u, v) \cdot \mathscr{F}\left[f_{\text{in}}(x, y)\right](u, v)\right](x, y). \tag{8}$$

However, we know that the Fourier transform of the delta function is equal to 1,

$$\mathscr{F}\left[\delta(x, y)\right] = 1. \tag{9}$$

Therefore, for the impulse response $h(x, y)$ of the filter, we have,

$$\begin{aligned} h(x, y) &= \mathscr{F}^{-1}\left[H(u, v) \cdot \mathscr{F}\left[\delta(x, y)\right](u, v)\right](x, y) && (10)\\ &= \mathscr{F}^{-1}\left[H(u, v) \cdot 1\right](x, y) && (11)\\ &= \mathscr{F}^{-1}\left[H(u, v)\right](x, y). && (12) \end{aligned}$$

Therefore, we see that the impulse response of the filter is equal to the inverse Fourier transform of its transfer function. Note that this applies to both continuous and discrete signals, filters and Fourier transforms.

Using the above, we calculate the impulse response and show it in Fig. 7. Note that, in general, the impulse response can be a complex-valued function, as is indeed the case here. Therefore, it cannot properly be visualized by using directly the function `imshow`. Instead, we show the logarithm of its amplitude $|h(x, y)|$, with the logarithm being used in order to suppress the very large magnitude differences and to make details visible.

(d) The results of denoising the image using a filter with width $w = 20$ and $w = 50$ are shown in Fig. 8 and Fig. 9, respectively (we show them in the same format as in part b).

As we briefly discussed in the first part of this problem, using a large w results in many frequency components corresponding to the original image being removed (filtered out). As we see from the results, these relatively high frequencies correspond to edge content. Therefore, filtering with these very wide filters results in edges being smoothed and showing much stronger ringing artifacts. What is remarkable is that, in the difference between the original and the "denoised" image, we

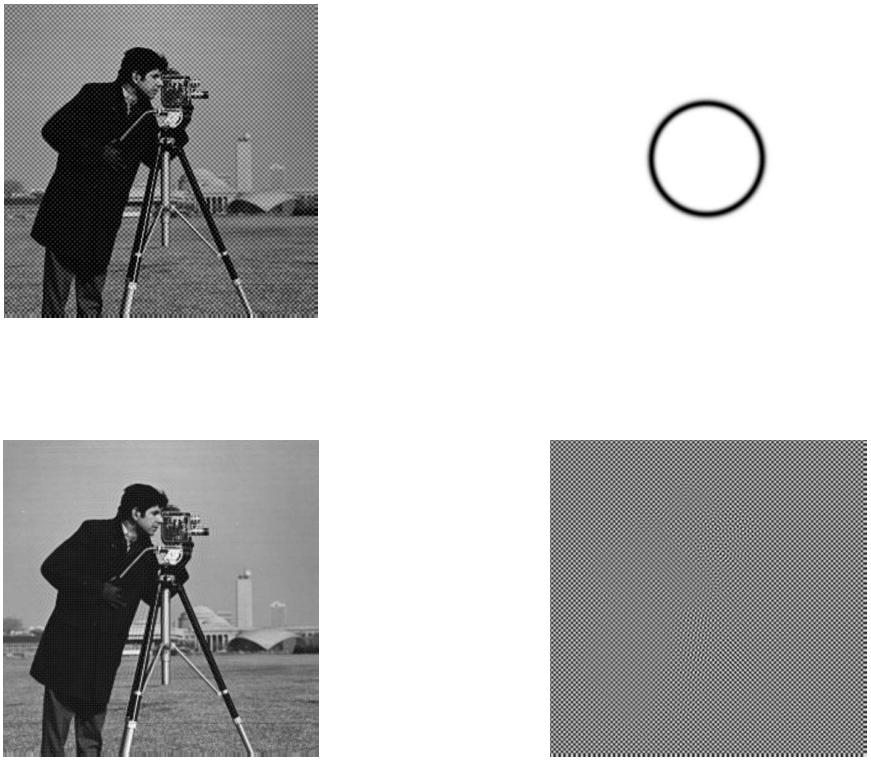

Figure 6: First row: original image (left) and transfer function of filter used for denoising (right). Second row: denoised image (left) and difference between original and denoised image.

Figure 7: Impulse response (logarithm of magnitude) of filter used for denoising.

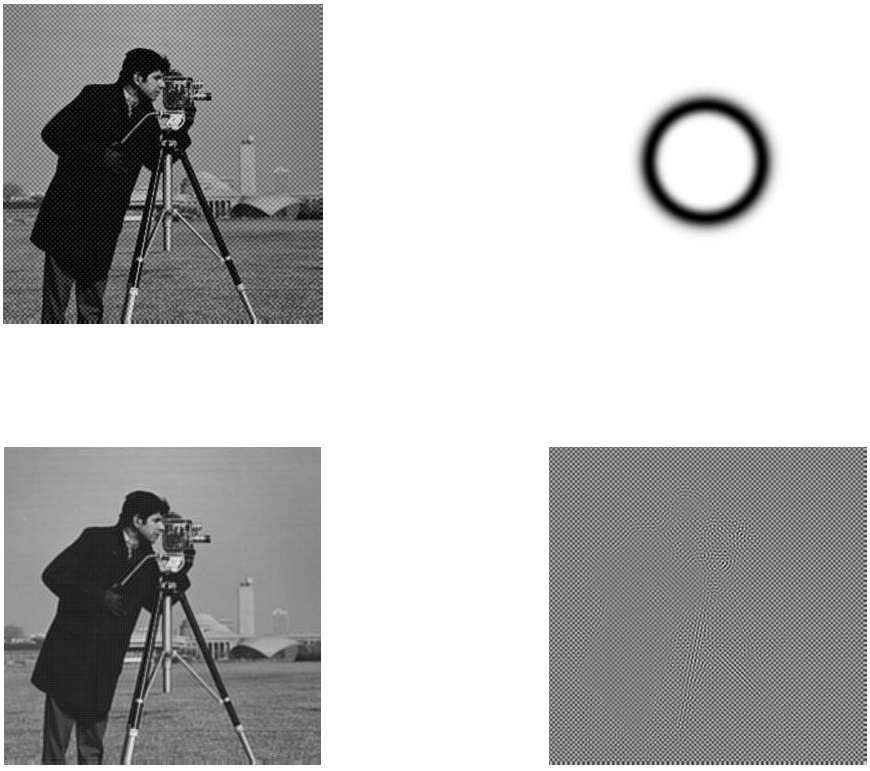

Figure 8: First row: original image (left) and transfer function of filter used for denoising (right). Second row: denoised image (left) and difference between original and denoised image (case $w = 20$).

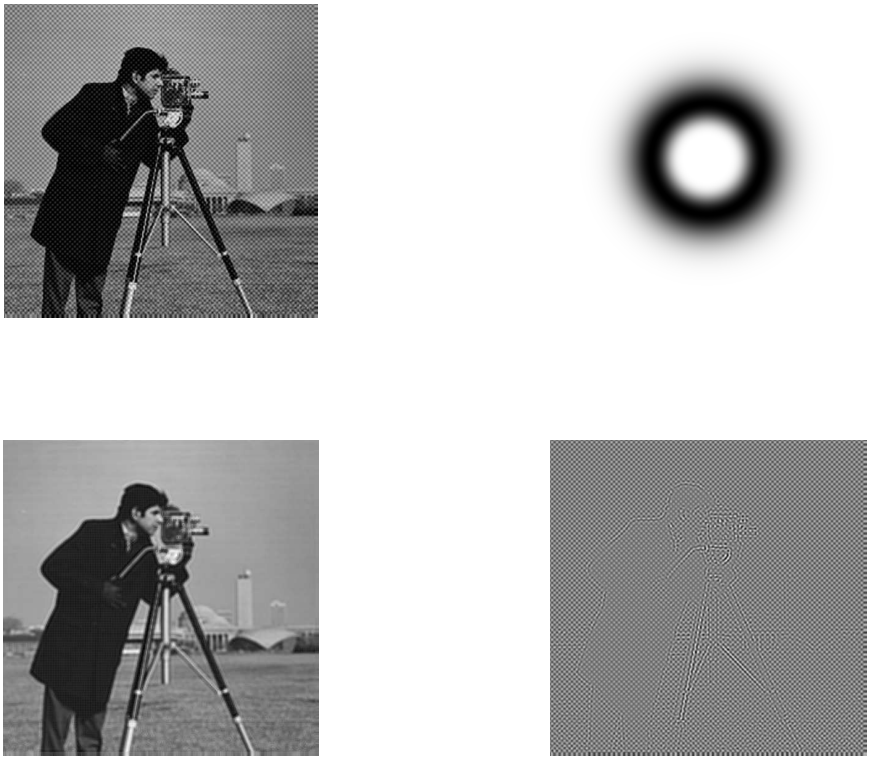

Figure 9: First row: original image (left) and transfer function of filter used for denoising (right). Second row: denoised image (left) and difference between original and denoised image (case $w = 50$).

can see some of the contours of the image scene. This confirms what we just mentioned, that the frequency components being filtered out correspond to edge content.

Below we provide one possible version of a script for solving question 3.

```
%%
% CS283 Fall 2015
% Student: Ioannis Gkioulekas
% Homework assignment 4, question 3
%
% Denoise image corrupted with periodic domain by using linear filtering

% load original image
I = imread('../data/Cameraman_PeriodicNoise.tif');
I = im2double(I);

%% Part a

% get size of image
[M, N] = size(I);

% create grid for centering
[X, Y] = meshgrid(0:M - 1,0:N - 1);

% create centered fourier transform of original image
If = fft2((-1) .^ (X + Y).* I);

% set DC component to zero
Iftemp = If;
Iftemp(M / 2 + 1, N / 2 + 1) = 0;

% display magnitude of centered DFT of image
figure; imshow(abs(Iftemp), []);

% display logarithm of magnitude of centered DFT of image
figure; imshow(log(abs(Iftemp)), []);

% grid for frequency domain (same as one for centering, we just rename
% variables for notational purposes)
U = X;
V = Y;

% set filter parameters
D0 = 90.5;
w = 8;

% find distances from center in frequency domain
D = hypot(U - M / 2, V - N / 2);

% create transfer function at each point in the frequency domain grid
H = 1 - exp(-((D .^ 2 - D0 ^ 2) ./ (w * D)).^2 / 2);

% display filter transfer function as surface
sampleFactor = 2;
figure; surf(U(1:sampleFactor:end, 1:sampleFactor:end),...
             V(1:sampleFactor:end, 1:sampleFactor:end),...
             H(1:sampleFactor:end, 1:sampleFactor:end));

%% Part b

% linear filtering in frequency domain (multiplication)
IfFiltered = If .* H;

% return filtered image to spatial domain and decenter
IFiltered = ifft2(IfFiltered) .* ((- 1) .^ (X + Y));

% display results
```

```
figure;
subplot(221); imshow(I, []);
subplot(222); imshow(H, []);
subplot(223); imshow(real(IFiltered), []);
subplot(224); imshow(I - IFiltered, []);

%% Part c

% get the impluse response as the (decentered) inverse Fourier transform of
% the transfer function
h = abs(ifft2(H) .* ((- 1) .^ (X + Y)));

% display logarithm of impulse response
figure; imshow(log(h - min(h(:))), []);
```