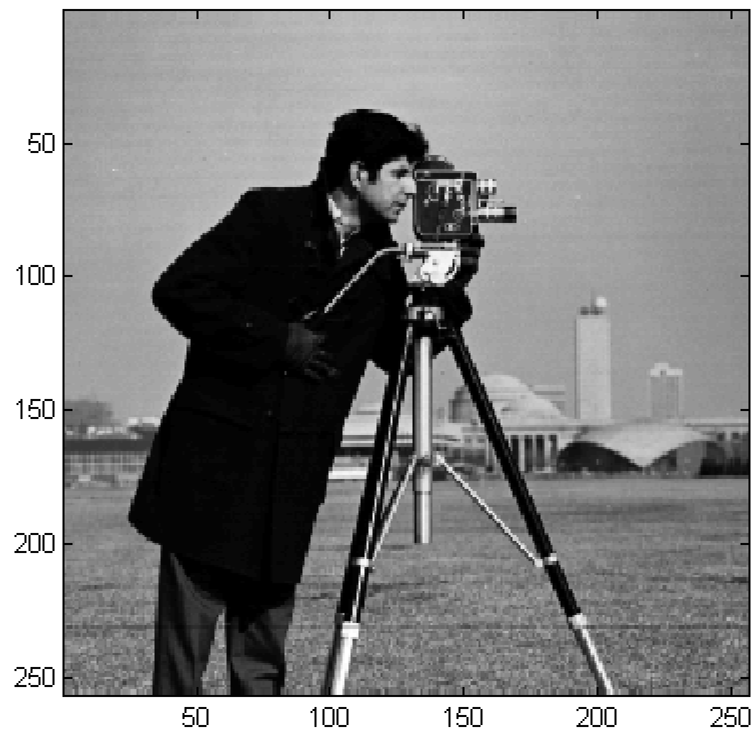
Lab 8: Image Processing and Compression, Dan Wortmann, May 9th, 2014

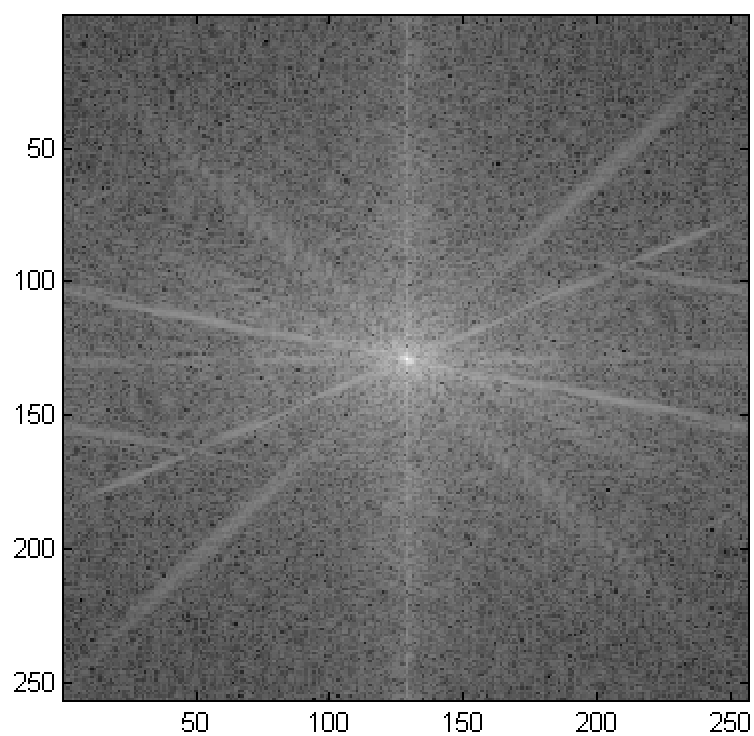
Table of Contents

Warm up

3.1

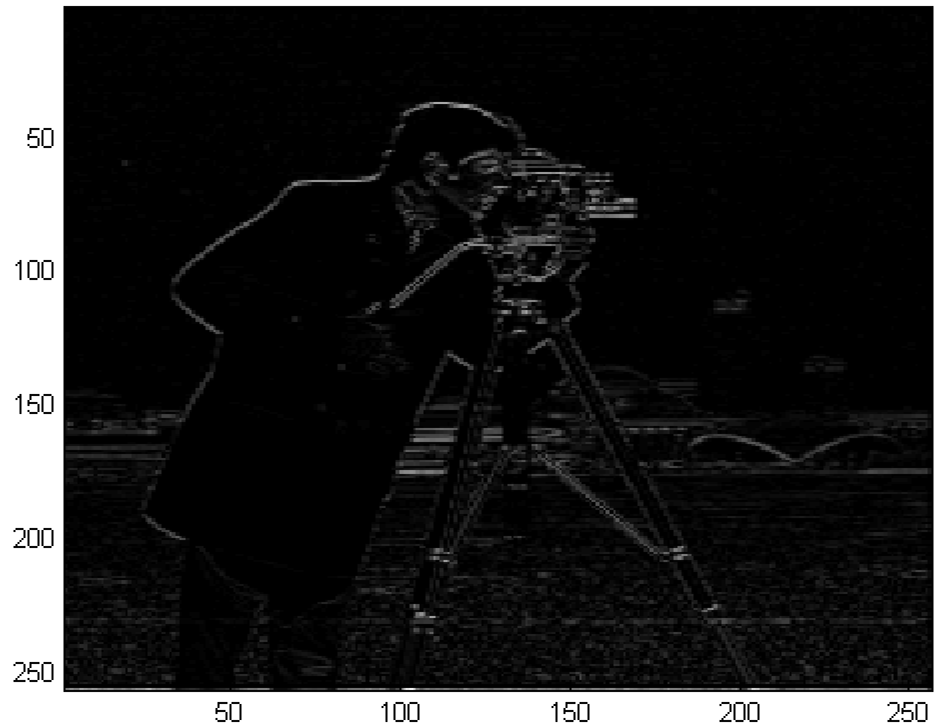
```
clc
clear
load cameraman.mat
figure(1)
imagesc(x)
colormap(gray)
axis('square')
figure(2)
spec_x = fft2(x);
imagesc(fftshift(log10(abs(spec_x))));
colormap(gray)
axis('square')

% When taking the fft of the image we get a display of several diagonal
% lines that are coming from the center of image. This may be caused by the
% contrasting colors between the cameraman, the cameran and the light
% background. So as you transfer from the dark jacke to the sky, it gives a
% prominant diagnal.
```

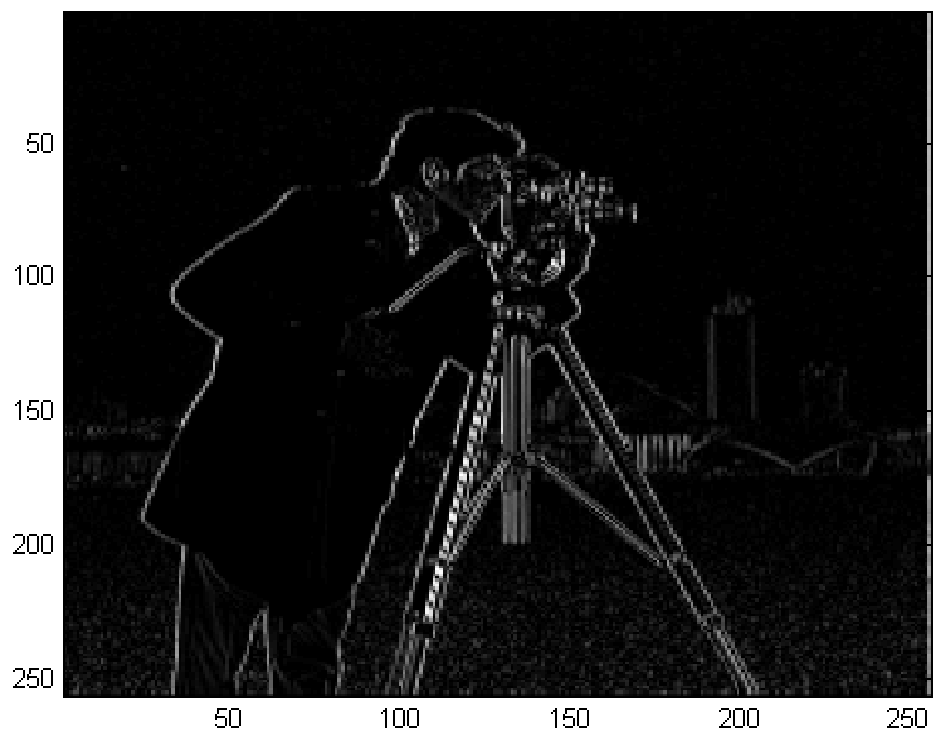

50
100
150
200
250
50
100
150
200
250


50
100
150
200
250
50
100
150
200
250

3.2

```
hh = [1/4 1/4; -1/4 -1/4];
hv = [1/4 -1/4; 1/4 -1/4];

yh = conv2(x,hh,'same');
yv = conv2(x,hv,'same');

figure(3)
imagesc(abs(yh));
colormap(gray)

figure(4)
imagesc(abs(yv));
colormap(gray)

%The two filters seems to detect edges in contrasting colors. As we see
%clear lines on colors that have large changes from black to white, but
%very dim - if any lines - on a lot of the gray background.

hlp = [1/4 1/4; 1/4 1/4];
ylp = conv2(x,hlp,'same');

figure(5)
imagesc(x - ylp)
colormap(gray);

%Blurring the image causes the original values to be dilluded and have
%smoother transitions. However once we subtract the original values from
%slightly smaller blurred values we get almost zeroes every EXCEPT where
%there is a sharp peak in contrast of the original image.
```


50
100
150
200
250
50
100
150
200
250


50
100
150
200
250
50
100
150
200
250

3.3

```
img = fft2(x);
filter = fft2(hlp, 256, 256);

result = real(ifft2(img .* filter, 256, 256));

figure(6)
imagesc(abs(x - result))
colormap(gray);
```

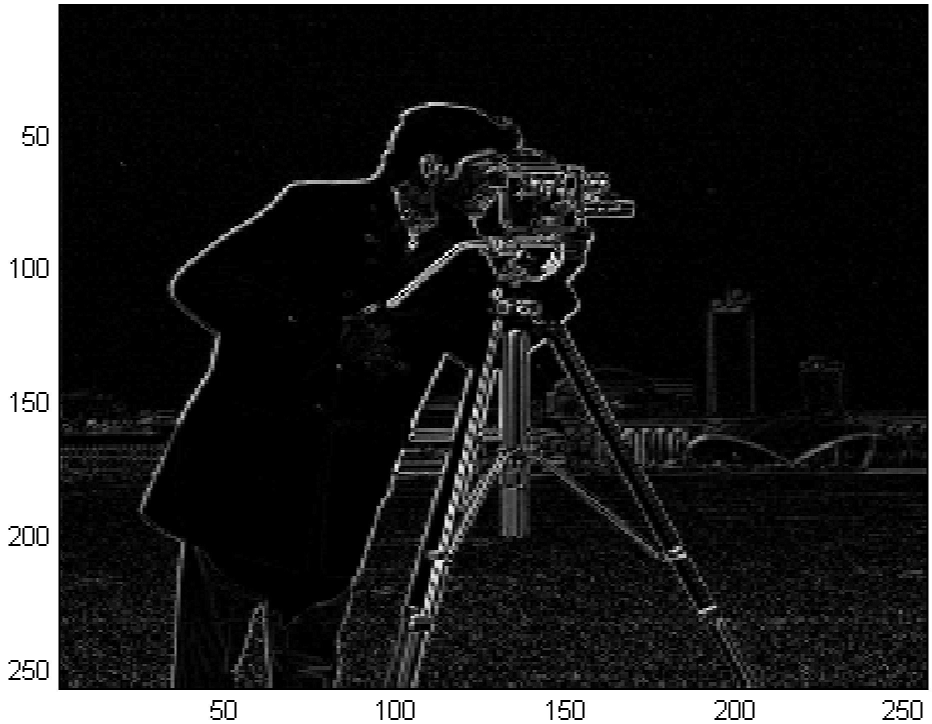


Image Processing, the DFT and DCT

```
clc
clear
close all

load nimes_france.mat
load blur.mat
load blurinv.mat
```

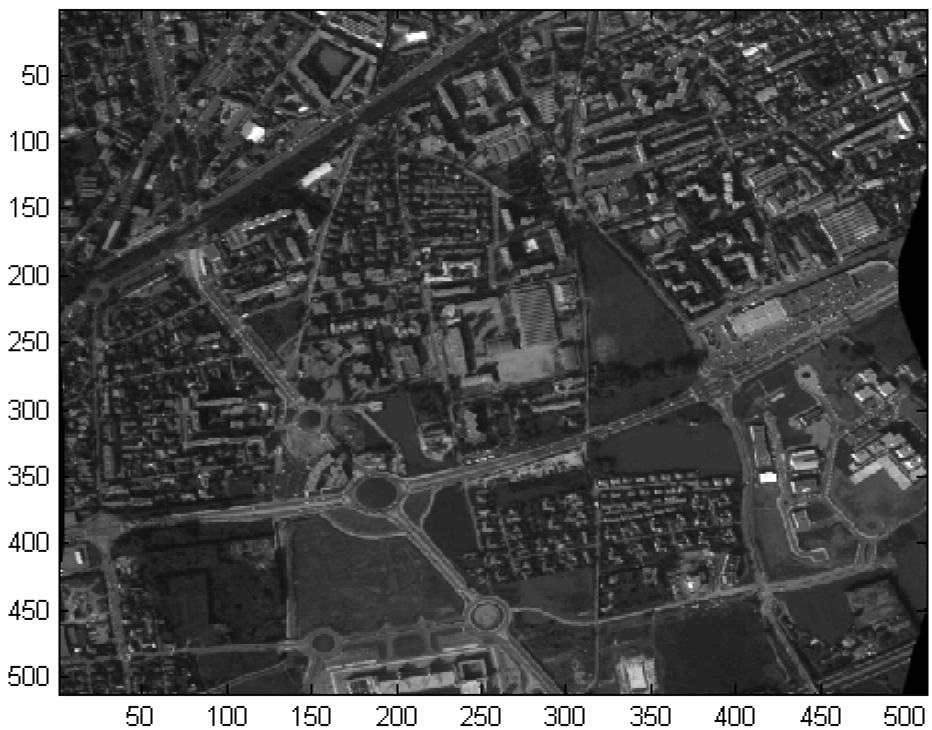
4.1 Image Restoration

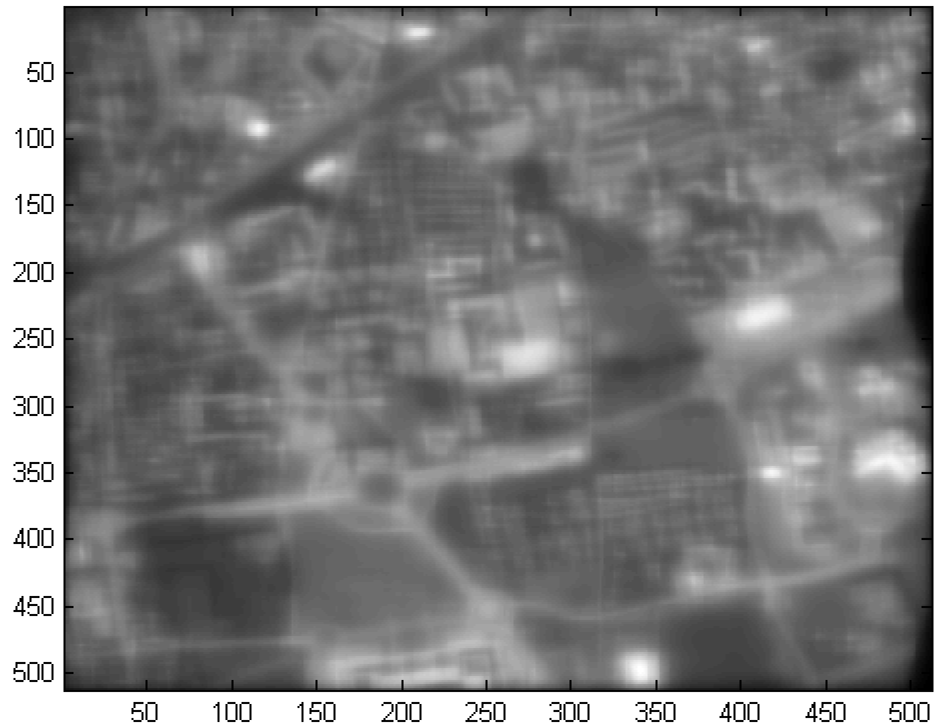
```
%Original
figure(1)
imagesc(x);
colormap(gray)

% a)
blurred = conv2(x, h, 'same');

figure(2)
imagesc(blurred);
colormap(gray)

% b)
```

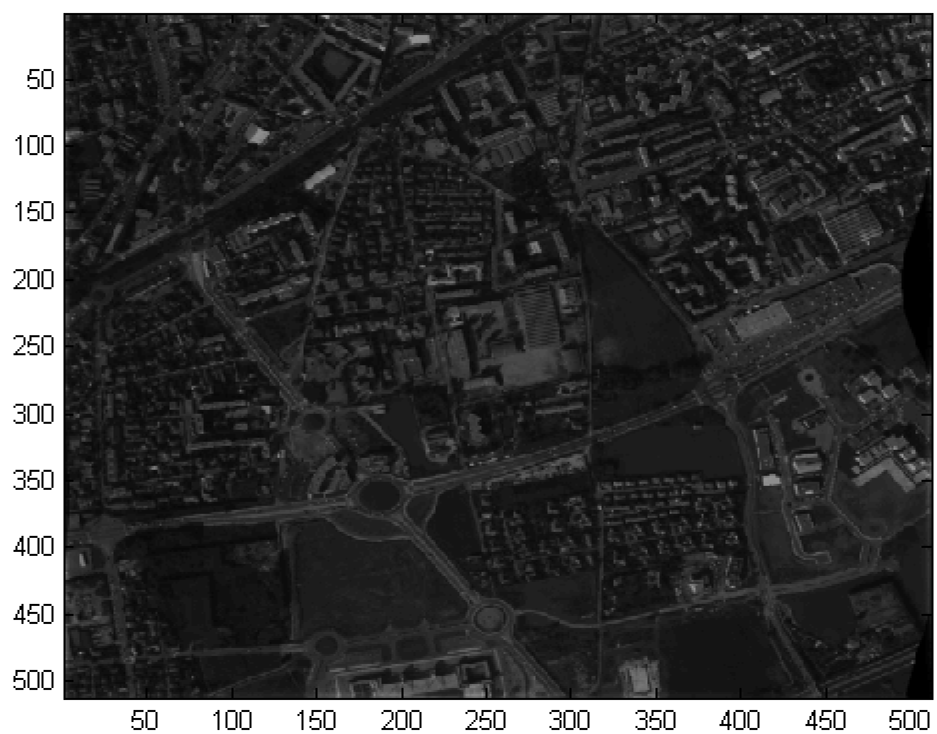
```
tic;
unblurred = conv2(blurred, invh, 'same');
conv2time = toc

figure(3)
imagesc(unblurred);
colormap(gray)

% c)
tic
DFT_blurred = fft2(blurred);
DFT_filter = fft2(h, 512, 512);

restored = ifft2(DFT_blurred ./ DFT_filter, 512, 512);
dfttime = toc

figure(4)
imagesc(restored)
colormap(gray);

% d)
N = 512;
shift = exp(-1j*2*pi*21/N*(0:N-1)')*exp(-1j*2*pi*21/N*(0:N-1));
restored1 = real(ifft2(fft2(restored).*shift));

figure(5)
imagesc(restored1)
colormap(gray);

%When we use the DFT method, we treat the image as an infinitely periodic
%string of images. This makes us pick up weird edge effects caused by the
%filter picking up parts of the bordering 'images'. By shifting the image
%slightly before taking the ifft, we do not read from the copies of the
%image and only end up restoring a single instance of the original.

% e)
blurred_noise = conv2(x,h,'same') + randn(size(x));

DFT_blurred_noise = fft2(blurred_noise);
DFT_filter_noise = fft2(h, 512, 512);

restored_noise = ifft2(DFT_blurred_noise ./ DFT_filter_noise, 512, 512);

figure(6)
imagesc(restored_noise)
colormap(gray);

%Dramatic effect on the overall image quality, although we definitely see
%the image it looks like an old tv turned onto a static station.
Hf = DFT_filter_noise;

improvedRestored = fft2(blurred_noise).*conj(Hf)./(abs(Hf).^2 + 0.001);

figure(7)
```

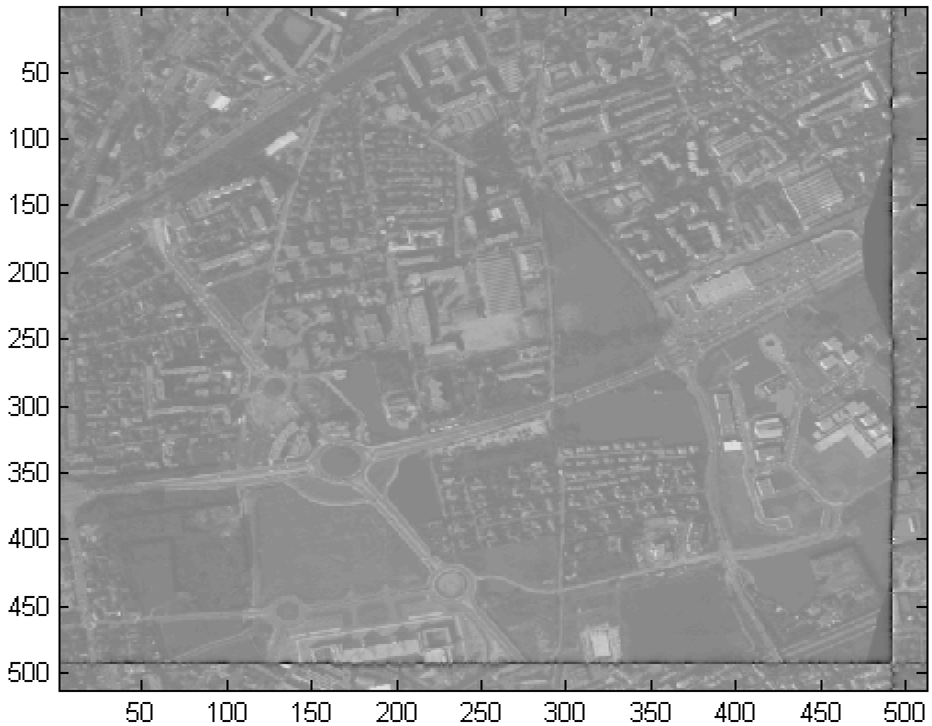
```
imagesc(abs(ifft2(improvedRestored)))
colormap(gray);

% By changing the slight offset in the denominator of the function we see
% drastic changes in the blurriness of the image if we increase it, and a
% large increase to static interference if we decrease the offset. This is
% caused by the relative weight to the values of the DFTs. If we divide by
% a small number, we can get relatively large values quickly so we can't
% decrease it too much. The same goes for increasing it too much and not
% filtering out the noise well enough.
```

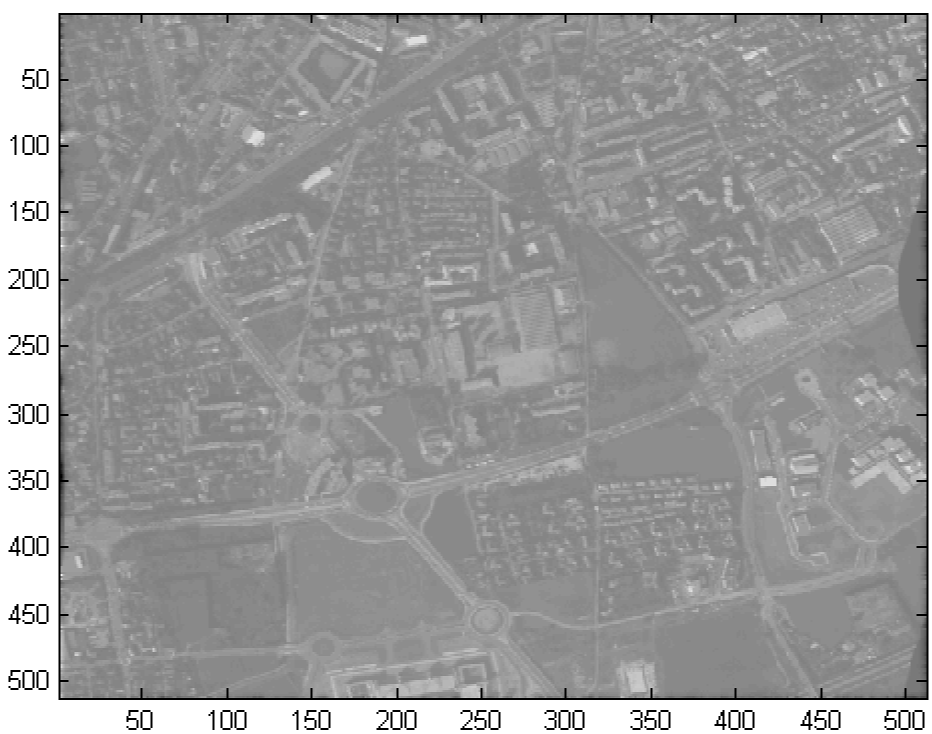
```
conv2time =

    0.0185


dfttime =

    0.0098
```

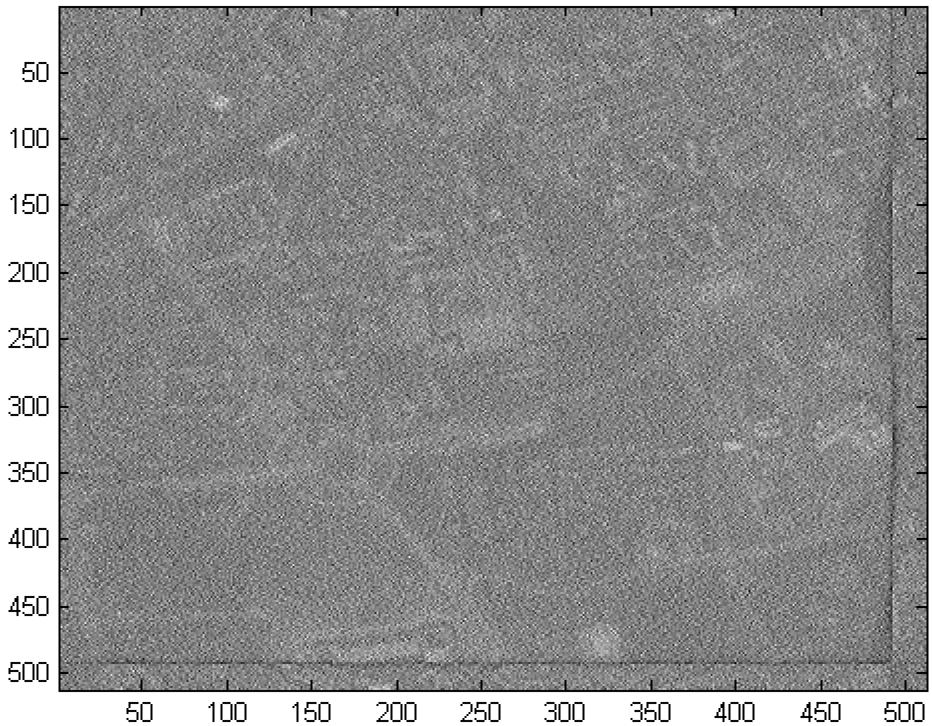


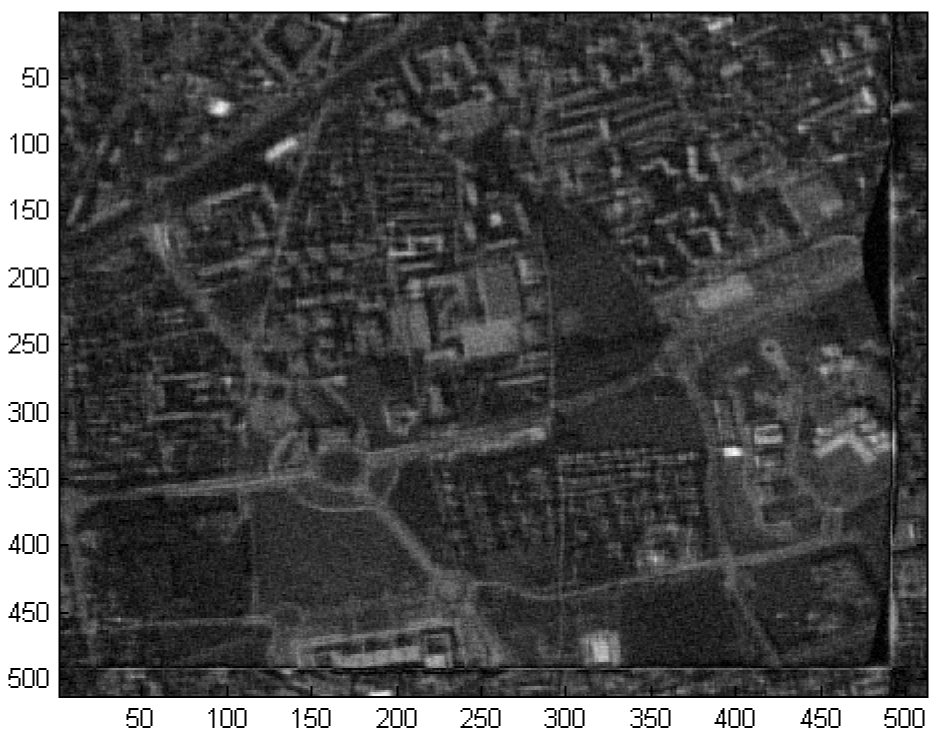

50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500


50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500

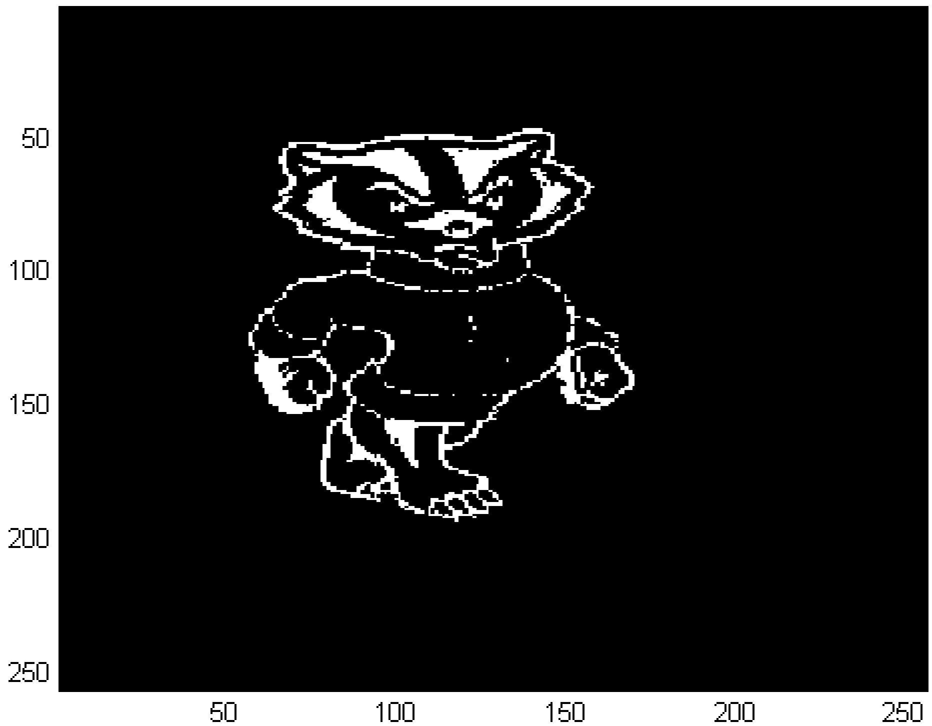

50
100
150
200
250
300
350
400
450
500
50 100 150 200 250 300 350 400 450 500


50
100
150
200
250
300
350
400
450
500
50 100 150 200 250 300 350 400 450 500

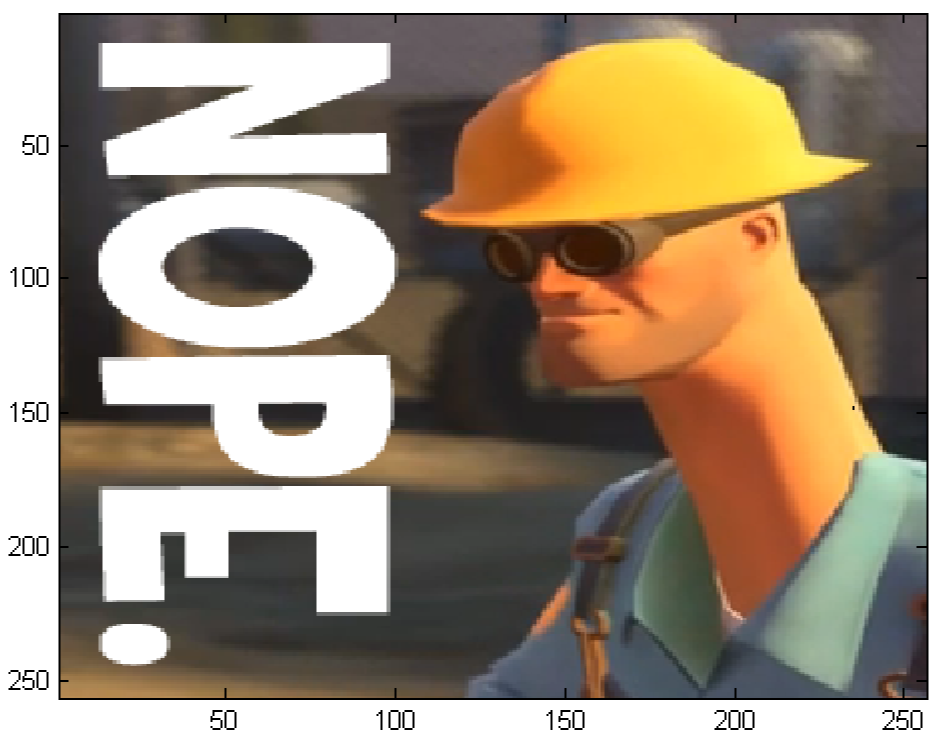
4.2 Image Quantization and Digital Watermarking

```
clc
clear
load cam_wm.mat

% a)
%To get the LSB, just determine whether the number is even - 0 or odd - 1
%for each pixel in the original image.

[M, N] = size(y);
watermark = zeros(M);

for m = 1:M
    for n = 1:N
        if(mod( y(m,n), 2) == 0)
            watermark(m,n) = 0;
        else
            watermark(m,n) = 1;
        end
    end
end

%Bucky the Badger
figure(8)
imagesc(watermark)
colormap(gray);

% b)
%For simplicity I will use the watermark I extracted from the previous step
%and implant it into another picture.

% load imagine using imread
x = imread('lab8matlab.jpg');
% convert data from int to double
%x = im2double(x_int);

figure(9)
imagesc(x)
colormap(gray)
title('Original Image')

%put the watermark into the picture
M = 256; N = 256;

for m = 1:M
    for n = 1:N
        if(watermark(m,n) == 1)
           if(mod(x(m,n),2) == 0)
               x(m,n) = x(m,n) + 1;
           end
```

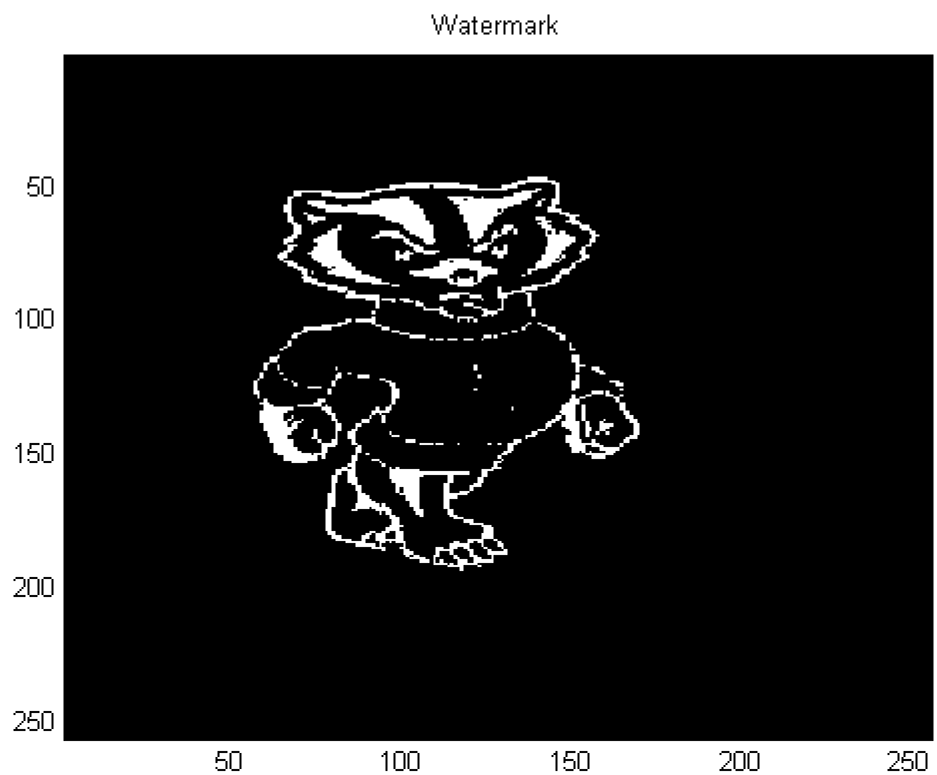
```
        else
            if(mod(x(m,n),2) == 1)
                x(m,n) = x(m,n) - 1;
            end
        end
    end
end

figure(10)
imagesc(x)
colormap(gray)
title('Watermarked Image')

%Now extract the watermark again
watermark1 = zeros(M);

for m = 1:M
    for n = 1:N
        if(mod( y(m,n), 2) == 0)
            watermark1(m,n) = 0;
        else
            watermark1(m,n) = 1;
        end
    end
end

%Bucky the Badger - again from the new image
figure(11)
imagesc(watermark1)
colormap(gray);
title('Watermark')
```



Original Image




Watermarked Image
NOPE!
50
100
150
200
250
50
100
150
200
250


Watermark
50
100
150
200
250
50
100
150
200
250

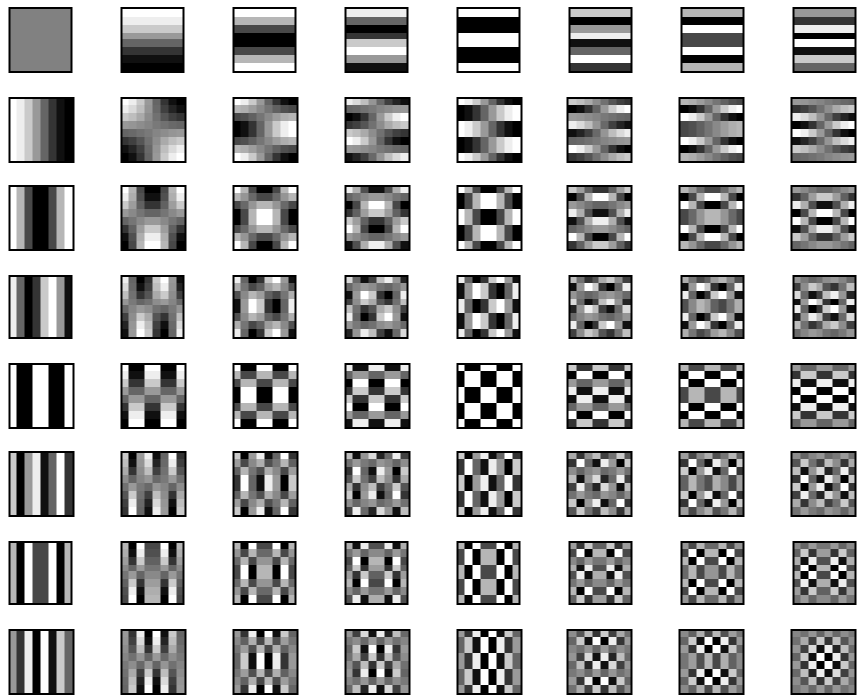
4.3 Image Compression

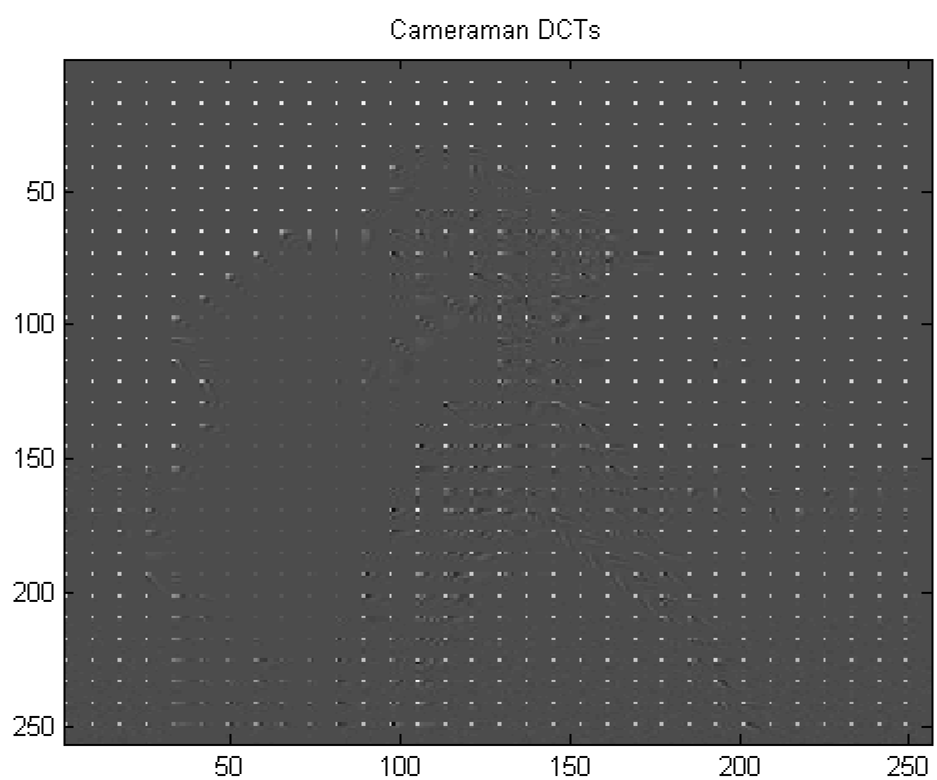
```
clc
clear
close all

% a)

figure(12)
N=8;
for m=1:N
    for n=1:N
        e=zeros(N,N);
        e(m,n)=1;
        b = idct2(e);
        subplot(8,8,m+(n-1)*N)
        imagesc(b)
        colormap(gray)
        set(gca,'Xtick',[])
        set(gca,'Ytick',[])
        axis('square')
    end
end

% b)

load cameraman.mat

[M,N] = size(x);
Mblocks = M/8;
Nblocks = N/8;

% compute DCT of 8x8 subimages
for m = 1:Mblocks
    for n=1:Nblocks
        Mrange = (m-1)*8+1:(m-1)*8+8;
        Nrange = (n-1)*8+1:(n-1)*8+8;
        block = x(Mrange,Nrange);
        DCTblock = dct2(block);
        y(Mrange,Nrange) = DCTblock;
    end
end

figure(13)
imagesc(y)
colormap(gray);
title('Cameraman DCTs')

%Since the DCT divides the picture up into blocks of 8x8, the information
%stored is in smaller blocks that only hold the coefficients used to
%reconstruct the original picture. This leads to smaller space used for
%saving data.
```

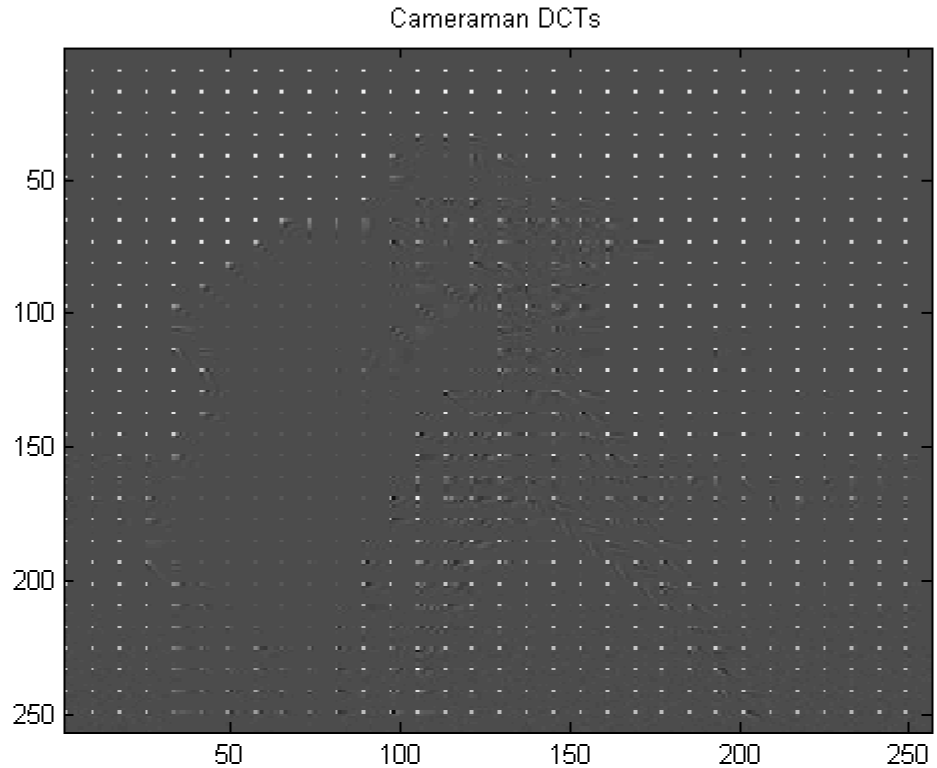
```
% c)

Q = [16 11 10 16 24 40 51 61;
    12 12 14 19 26 58 60 55;
    14 13 16 24 40 57 69 56;
    14 17 22 29 51 87 80 62;
    18 22 37 56 68 109 103 77;
    24 35 55 64 81 104 113 92;
    49 64 78 87 103 121 120 101;
    72 92 95 98 112 100 103 99];

[M,N] = size(x);
Mblocks = M/8;
Nblocks = N/8;

% compute DCT of 8x8 subimages
for m = 1:Mblocks
    for n=1:Nblocks
        Mrange = (m-1)*8+1:(m-1)*8+8;
        Nrange = (n-1)*8+1:(n-1)*8+8;
        block = x(Mrange,Nrange);
        DCTblock = dct2(block);
        quantized_DCTblock = round(DCTblock./Q);
        y(Mrange,Nrange) = DCTblock;
        yy(Mrange,Nrange) = quantized_DCTblock;
    end
end

figure(14)
imagesc(y)
colormap(gray);
title('Cameraman DCTs')

figure(15)
imagesc(yy)
colormap(gray);
title('Cameraman DCTs - quantized')

[M,N] = size(x);
Mblocks = M/8;
Nblocks = N/8;
nonZeroCount = 0;

for m = 1:M
    for n = 1:N
        if(yy(m,n) ~= 0)
            nonZeroCount = nonZeroCount + 1;
        end
    end
end

compressionRatio = nonZeroCount / (M*N) %0.1478

% d)
```

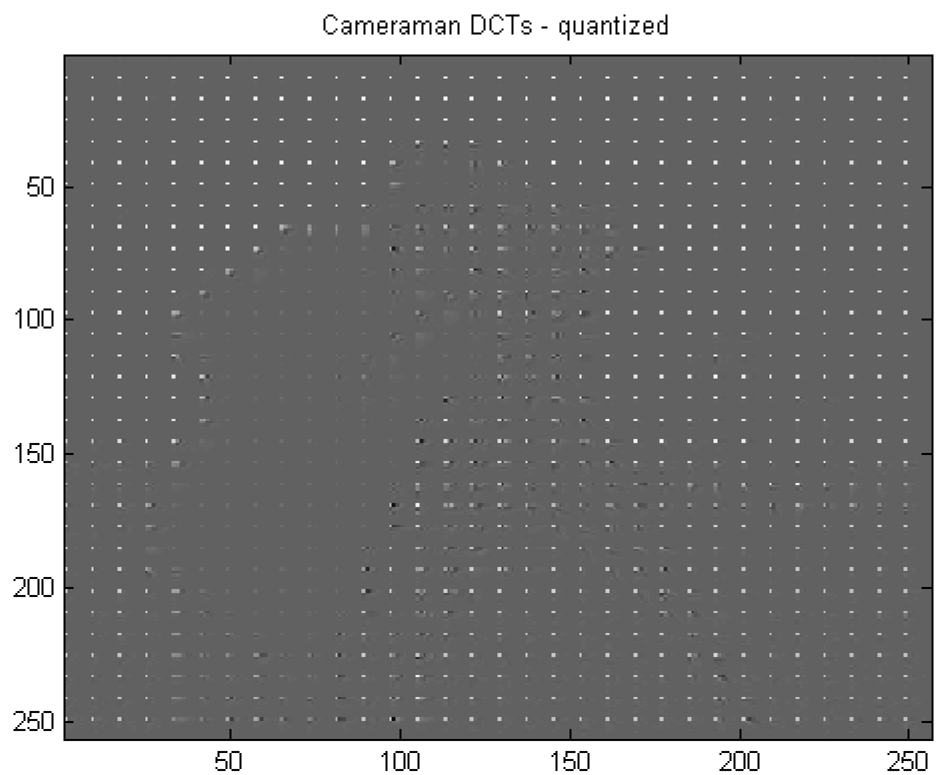
```
reconstructed = zeros(N);
scalar = 1;
% compute inverse DCT of 8x8 subimages
for m = 1:Mblocks
    for n=1:Nblocks
        Mrange = (m-1)*8+1:(m-1)*8+8;
        Nrange = (n-1)*8+1:(n-1)*8+8;
        block = yy(Mrange,Nrange);
        inverseDCTblock = round(block.*(Q*scalar));
        block = idct2(inverseDCTblock);
        reconstructed(Mrange,Nrange) = block;
    end
end

figure(16)
imagesc(reconstructed)
colormap(gray);
title('Cameraman DCTs - reconstructed')

%This reconstruction results in the original image, however there are some
%noticeable blurs in areas of a lot of detail and color change. This was
%particularily highlighted around the camera and the edges of the camera
%mans jacket. As we decrease the value of the scalar the image gets more
%and more distorted. As we reach values of around 1/100 we start to see a
%drastic change in the image quality where we can hardly make out the
%camera man. On the opposite end, when we increase the scalar over 1 we see
%minimal improvements to the image. After values of about 100 we see less
%blotches in solid colors around the camera man, however the general
%sharpness of the image is about the same as with a scalar of 1.
```

```
        compressionRatio =

            0.1478
```




Cameraman DCTs
50
100
150
200
250
50
100
150
200
250


Cameraman DCTs
50
100
150
200
250
50
100
150
200
250


Cameraman DCTs - quantized
50
100
150
200
250
50
100
150
200
250

Published with MATLAB® R2013a