Dear EECS 556 Class – Thanks for your attention and participation in class. I hope you learned as much from taking this class as I did from teaching it. The class performance on this exam was excellent and the median grade on the final was 91. Please accept my best wishes for an enjoyable summer. Cheers, DN

1. Transforms and transform coding.

a. Consider the *N*-length Fourier transform coefficients:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\exp\left(-i2p\,\frac{kn}{N}\right) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\left[\cos\left(2p\,\frac{kn}{N}\right) - i\sin\left(2p\,\frac{kn}{N}\right)\right].$$

For real valued *x*(*n*), we know that *X*(*k*) = *X**(*N*-*k*). We also know that *X*(0) and *X*(*N*/2) are real-valued. Thus, there are exactly *N* unique real-valued coefficients: *X*(0), *X*(*N*/2), real{*X*(*k*)}, and imag{*X*(*k*)} for *k*=1, 2, … *N*/2-1. The transform kernel for each of these is $\frac{1}{N}$ times [1 1 … 1], [1 -1 1 … -1], $\cos\left(2p\,\frac{kn}{N}\right)$, and $-\sin\left(2p\,\frac{kn}{N}\right)$, respectively, for *n*=0, 1, 2, … *N*-1. For an energy preserving transformation, we'll need to normalize each kernel to have unit energy. Ordering form highest to lowest energy usually means ordering from lowest frequency to highest frequency. So, the 8x8 transformation matrix is:

$$A = \frac{1}{\sqrt{8}}\left|\begin{array}{c}
1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1 \\
\sqrt{2}\cos\left(2p\,\frac{n}{8}\right) \\
-\sqrt{2}\sin\left(2p\,\frac{n}{8}\right) \\
\sqrt{2}\cos\left(2p\,\frac{2n}{8}\right) \\
-\sqrt{2}\sin\left(2p\,\frac{2n}{8}\right) \\
\sqrt{2}\cos\left(2p\,\frac{3n}{8}\right) \\
-\sqrt{2}\sin\left(2p\,\frac{3n}{8}\right) \\
1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1\ \ 1\ \ -1
\end{array}\right|$$

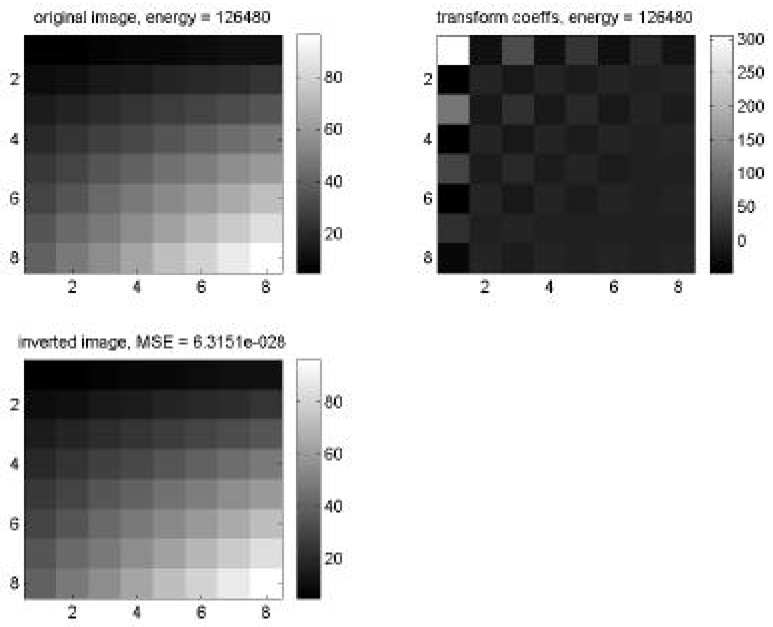
For *n*=0, 1, 2, … *N*-1.

b. Since the Fourier kernel is orthogonal and we have made the rows unit length, the *A* matrix is unitary and thus, the inverse transform is the transpose of *A*: *B* = *A'*.

c. The 8x8 transformation of some subimage *x* is simply the transform of the columns of *x* by *A* (e.g. *y* = *Ax*), followed by the transformation of the columns of *y'* by *A* (e.g. *X* = (*Ay'*)'). The net result is *X* = *AxA'* .

d. The 8x8 inverse transformation of some transform block *X* is simply the transform of the columns of *x* by *A'* (e.g. *y* = *A'X*), followed by the transformation of the columns of *y'* by *A'* (e.g. *x* = (*A'y'*)'). The net result is *x* = *A'XA* .

e. See plots and code. This transform can be implemented in a similar manner to transforms used in Homework #9. We could also get the transform coefficients directly from the output of `fft` or `fft2` and scale those appropriately. For larger blocks, this latter approach may be desirable for computational reasons $O(N\log N)$ vs. $O(N^2)$.



Matlab code:

```
% solution to final, problem 1
im = [1:8]'*[5:12];

% build transform matrix
N = 8; nn=[0:N-1]; rftmat = zeros([N N]);
rftmat(1,:) = ones([1 N]);
for lp = 1:N/2-1
    rftmat(lp*2,:) = sqrt(2)*cos(2*pi*lp*nn/N);
    rftmat(lp*2+1,:) = -sqrt(2)*sin(2*pi*lp*nn/N);
end
rftmat(N,:) = cos(2*pi*nn/2);
rftmat = rftmat./sqrt(N);

% inverse is rftmat' - this makes it a unitary (energy preserving) transform

% 2DFT
coeffs = zeros([8 8]);
iminv = zeros([8 8]);
% can do like solutions to HW#9 or compactly like this
coeffs = rftmat*im*rftmat';
coeffs = real(coeffs);

% 2D IDFT
iminv = rftmat'*coeffs*rftmat;

% check energy conservation and invertability
eim = sum(im(:).^2);
ecoeffs = sum(coeffs(:).^2);
mse = mean((im(:)-iminv(:)).^2);
subplot(221); imagesc(im); colormap(gray); colorbar;
title(['original image, energy = ' num2str(eim)]);
subplot(222); imagesc(coeffs); colormap(gray); colorbar;
title(['transform coeffs, energy = ' num2str(ecoeffs)]);
subplot(223); imagesc(iminv); colormap(gray); colorbar;
title(['inverted image, MSE = ' num2str(mse)]);
```

2. Deblurring.

a. The time function of the aperture will take on the form: rect(t/T), where T is the exposure time and the car (camera) position can be represented as $x = vt$. The image will then take on the form $\tilde{f}(x,y) = \frac{1}{T}\int_{-T/2}^{T/2} f(x - Mvt, y)dt$, where M is (de)magnification factor associated with the cameras optics. We can write the above as

$$\tilde{f}(x,y) = \frac{1}{T}\int_{-\infty}^{\infty} f(x - Mvt, y) rect(t/T)dt$$

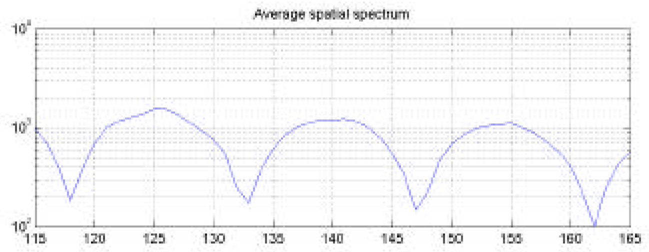
$$= \frac{1}{MvT}\int_{-\infty}^{\infty} f(x - x', y) rect(x'/MvT)dx'$$

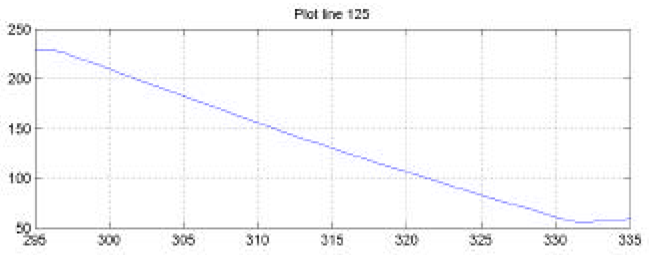
$$= f(x,y) ** \frac{1}{W} rect(x/W)d(y) \quad \text{where} \quad W = MvT$$

W is the image blur (in pixels) in the x direction. There is no blur in y.

b. The FT of the above is $\tilde{F}(u,v) = F(u,v)\text{sinc}(Wu)$. For the discretized space, we get $\tilde{F}(w_x, w_y) = F(w_x, w_y)\frac{\sin(Ww_x/2)}{W\sin(w_x/2)}$ or $\tilde{F}(k,l) = F(k,l)\frac{\sin(2pkW/2N)}{W\sin(2pk/2N)}$.

The continuous function has zeros every $1/W$. The discretized version has zeros every N/W. From the first plot, we can see that there are zeros roughly every 14 2/3 frequency bins and N=512. So, W = 512/(14 2/3) $\approx$ 34.9 pixels.

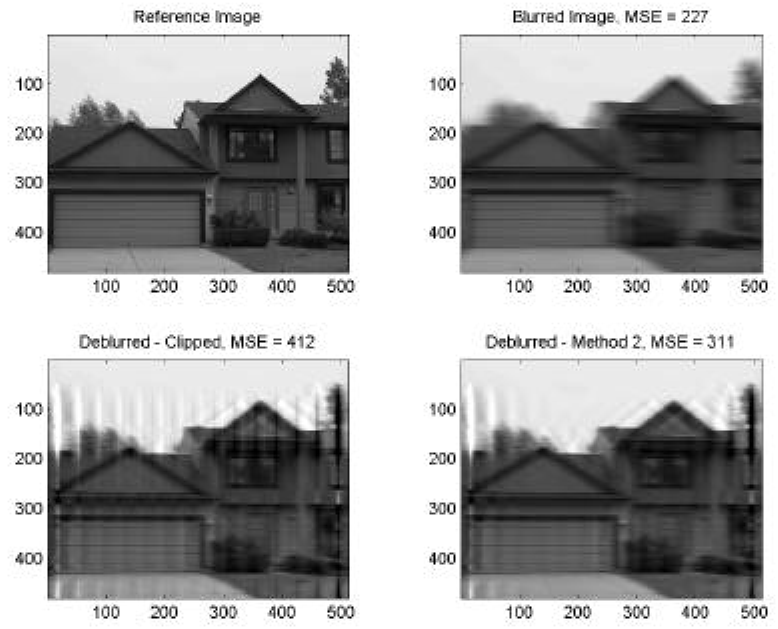




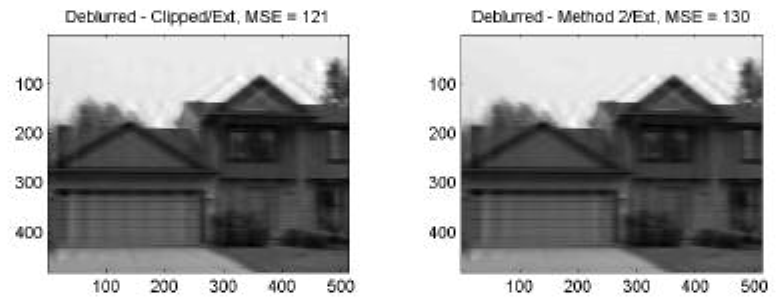
We also observe that rect blurring of step-like edges in an image will result in a ramp of length equal to the width of the rect. Here we see that the ramp goes from roughly from 296 to 331, a width of 35 pixels.

Based on these two observations, our estimate for the blur is 35 pixels. This was the size of the blur that was used in creating the blurred image.

c. See code and figures:


Reference Image
Blurred Image, MSE = 227
Deblurred - Clipped, MSE = 412
Deblurred - Method 2, MSE = 311
100
200
300
400
500

d. The edge artifacts result from the fact that the blur happen by linear convolution and the deblurring used circular convolution. Clearly, the circularity assumption was not real good. In our discussion of convolution, we discussed numerous ways to treat edge pixels. Zero padding will result in a discontinuity at the edge that is not desirable. Replication of the edge pixel is not desirable because it doesn't replicate the blurring character, but would be better then zero padding or circular convolution. The best solution is the replicate by mirroring around the edge. This only needs to be done in *x*.


Deblurred - Clipped/Ext, MSE = 121
Deblurred - Method 2/Ext, MSE = 130
100
200
300
400
500

e. The clipped/mirrored extension has the lowest MSE, but method 2 with mirrored extensions is better visually (less ringing off of the edges).

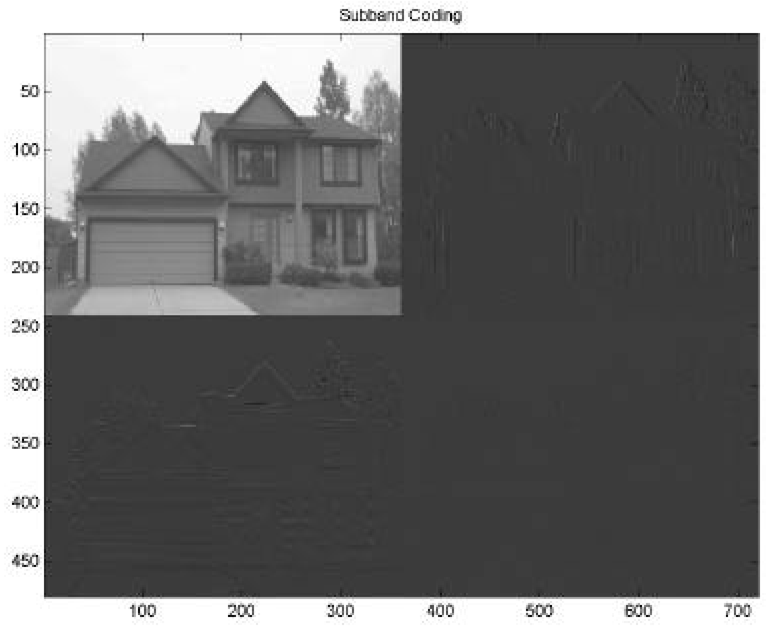
f. The non-extended deblurring kernels were dominated by edge artifacts and the MSE was not affected in a reliable way by the change in model parameters. For the extended images, the MSE increased with change in model parameters. For the clipped/extended method $\frac{d(MSE)}{dW} \approx 4.5$ and method 2/ext. $\frac{d(MSE)}{dW} \approx 2$ for a unit change in *W*. Thus, method 2 is more robust to model errors. One can also plot MSE vs. *W* for these cases and also see that method 2 is more robust.

Matlab code:

```
load finalimage2;
% blurho is blurred image of house
% refim is to be used only for calculation of MSEs and comparison

bhwin = blurho.*(ones([480 1])*hanning(512)');
spec = sum(abs(fft(bhwin,[],2)),1);
figure(1)
subplot(211); semilogy(spec); title('Average spatial spectrum')
axis([115 165 1e2 1e4]); grid
subplot(212); plot(blurho(125,:)); title('Plot line 125')
axis([295 335 50 250]); grid

% find spectral response of blur
W = 35;
[sx sy] = size(blurho);
[xx yy] = ndgrid([-sx/2:sx/2-1]./sx, [-sy/2:sy/2-1]./sy);
B = sinc(W.*yy);

gam = 3; msk = 1./abs(B) < gam;
H1 = 1./B .*(msk + (1-msk).*abs(B).*gam);
a = 0.03;
H2 = (B(sx/2+1,sy/2+1)^2 + a)/(B(sx/2+1,sy/2+1)^2)*B./(B.^2+a);

HB = fftshift(fft2(blurho));

imres1 = real(ifft2(fftshift(HB.*H1)));
imres2 = real(ifft2(fftshift(HB.*H2)));

mseblur = mean((refim(:) - blurho(:)).^2);
msere1 = mean(mean((imres1-refim).^2));
msere2 = mean(mean((imres2-refim).^2));

figure(2)
subplot(221); imagesc(refim,[0 255]); colormap(gray);
title('Reference Image');
subplot(222); imagesc(blurho,[0 255]); colormap(gray);
title(['Blurred Image, MSE = ' int2str(mseblur)]);
subplot(223); imagesc(imres1,[0 255]); colormap(gray);
title(['Deblurred - Clipped, MSE = ' int2str(msere1)]);
subplot(224); imagesc(imres2,[0 255]); colormap(gray);
title(['Deblurred - Method 2, MSE = ' int2str(msere2)]);

% extend by mirrorred reflection (do in x only)
blurhoext = [blurho(:,256:-1:1) blurho blurho(:,512:-1:257)];

[sx sy] = size(blurhoext);
[xx yy] = ndgrid([-sx/2:sx/2-1]./sx, [-sy/2:sy/2-1]./sy);
B = sinc(W.*yy);

gam = 3; msk = 1./abs(B) < gam;
H1 = 1./B .*(msk + (1-msk).*abs(B).*gam);
a = 0.03;
H2 = (B(sx/2+1,sy/2+1)^2 + a)/(B(sx/2+1,sy/2+1)^2)*B./(B.^2+a);

HB = fftshift(fft2(blurhoext));

imres1ex = real(ifft2(fftshift(HB.*H1)));
imres2ex = real(ifft2(fftshift(HB.*H2)));
imres1ex = imres1ex(:,257:768);
imres2ex = imres2ex(:,257:768);

msere1ex = mean(mean((imres1ex-refim).^2));
msere2ex = mean(mean((imres2ex-refim).^2));

figure(3); clf
subplot(221); imagesc(imres1ex,[0 255]); colormap(gray);
title(['Deblurred - Clipped/Ext, MSE = ' int2str(msere1ex)]);
subplot(222); imagesc(imres2ex,[0 255]); colormap(gray);
title(['Deblurred - Method 2/Ext, MSE = ' int2str(msere2ex)]);
```

3. VQ – Please see discussion of VQ in Lim, pp. 598-606.
   a. Gain from using VQ is very large. This is a bi-variate Gaussian RV with a correlation of 0.9 between x_1 and x_2. The variance of x_2, given x_1 is greatly reduced (to $(1 - r^2)$). This means that many fewer bits are necessary to code of x_2, once x_1 is coded.
   b. Gain from VQ is minimal or none. Since $f(x_1, x_2) = f(x_1)f(x_2)$ the RV's are independent and little gain can be made. A nominal gain might be achievable based on packing considerations (e.g. see Lim, Fig. 10.13).
   c. Again, since $f(x_1, x_2) = f(x_1)f(x_2)$ the RV's are independent and little gain can be made.
   d. Here, a small to modest amount of gain can be achieved though VQ. x_1 and x_2 are correlated, though not strongly ($r = -1/11$)
   e. Here, a large gain is possible through the use of VQ. There is a high degree of correlation ($r = 0.75$) and we can see that ½ of the reconstruction levels are necessary for the same distortion as scalar quantization.

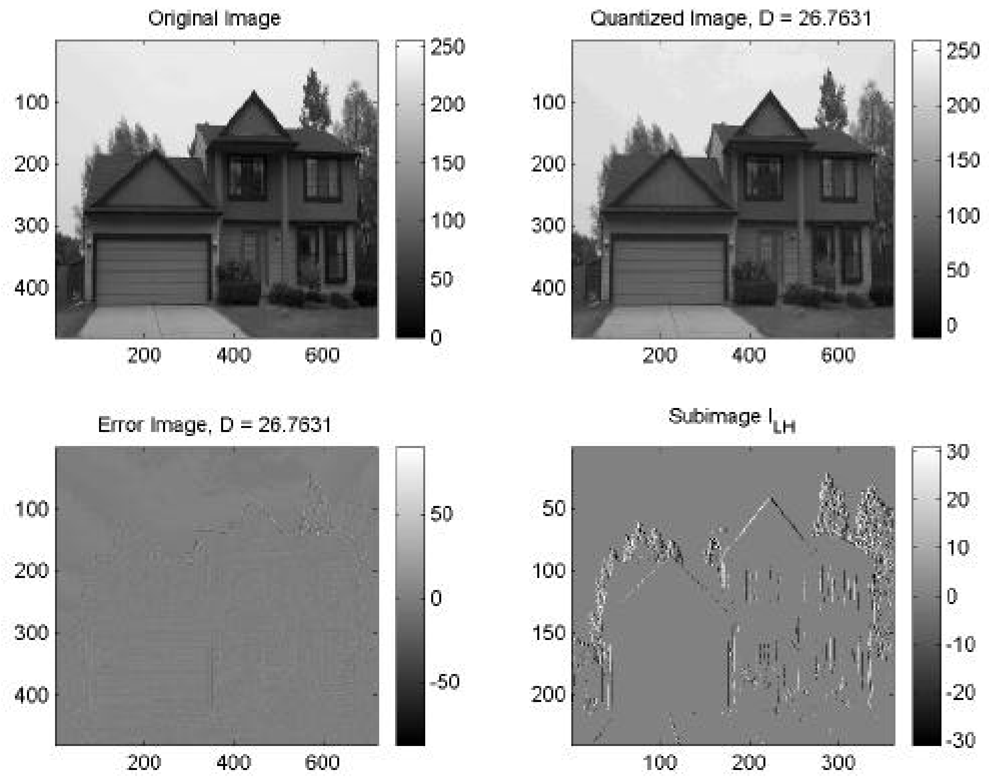
4. Subband Coding.
   a. Energy in original image and in subband coding image were both 6.8368e+009. It was energy preserving.



   b. See code. Error was zero – inversion was exact.
   c. Using the bit allocation formula of Lim, Equation 10.15 we get optimal bit allocation of 5.2828, 1.2277, 1.2214, 0.2681, respectively. The bit allocation used is pretty close to the optimal.
   d. See code.
   e. The optimal value for a is 30.9 and decision thresholds were +/- 15.45.
   f. A Huffman code for $\{-a, 0, a\}$ is {10, 0, 11}. The bit rate for I_{HL} and I_{LH} is 1.06 bits/pixel and thus we achieved our desired bit rate of 1.5 bits/pixel.

g. The entropy was $H = 0.3345$, so our coding scheme did not come very close to the optimal. There are lots of ways of getting closer to the entropy. First, we could concatenate codes (like Shannon's 1st Theorem) and derived a new codebook. Seeing the spatial correlations of I_{LH} in the figure below (Shannon's result assumes independence), one logically thinks of using alternate quantization techniques like VQ or coding schemes that incorporate run length coding for efficient coding of the long runs of zeros.

h. The average bit rate here was $(5 + 1.06 + 1.06 + 0)/4 = 1.78$ bits/pixel (our target was 2 bits/pixel). The distortion was comparable to VQ encoded with 2 bits/pixel ($D_{subband} = 26.8$ vs. $D_{VQ} = 26.4$).



Matlab code:

```
% solution problem 4
load ../hw7image;
% forward transform
subimLL = (ho(1:2:end,1:2:end) + ho(2:2:end,1:2:end) + ho(1:2:end,2:2:end) +
ho(2:2:end,2:2:end))/2;
subimHL = (ho(1:2:end,1:2:end) - ho(2:2:end,1:2:end) + ho(1:2:end,2:2:end) -
ho(2:2:end,2:2:end))/2;
subimLH = (ho(1:2:end,1:2:end) + ho(2:2:end,1:2:end) - ho(1:2:end,2:2:end) -
ho(2:2:end,2:2:end))/2;
subimHH = (ho(1:2:end,1:2:end) - ho(2:2:end,1:2:end) - ho(1:2:end,2:2:end) +
ho(2:2:end,2:2:end))/2;
im2 = [subimLL subimLH; subimHL subimHH];
figure(1); imagesc(im2); colormap gray; title('Subband Coding')
eorig = sum(ho(:).^2)
etrans = sum(im2(:).^2)
% inverse transform
newho = zeros(size(ho));
newho(1:2:end,1:2:end) = (subimLL + subimHL + subimLH + subimHH)/2;
newho(2:2:end,1:2:end) = (subimLL - subimHL + subimLH - subimHH)/2;
newho(1:2:end,2:2:end) = (subimLL + subimHL - subimLH - subimHH)/2;
newho(2:2:end,2:2:end) = (subimLL - subimHL - subimLH + subimHH)/2;
inverror = mean((ho(:)-newho(:)).^2)
% variacens and bit rates
vll = var(subimLL(:));
vhl = var(subimHL(:));
vlh = var(subimLH(:));
vhh = var(subimHH(:));
vm = (vll*vhl*vlh*vhh).^(1/4);
bll = (2 + 0.5*log2(vll/vm))
bhl = (2 + 0.5*log2(vhl/vm))
blh = (2 + 0.5*log2(vlh/vm))
bhh = (2 + 0.5*log2(vhh/vm))
% quantize HH and LL
subimHH = zeros(size(subimHH));
L = 2^5;
mn = min(subimLL(:));
mx = max(subimLL(:));
delt = (mx-mn)/L;
subimLL = round((subimLL - mn + delt/2)./delt).*delt - delt/2 + mn;

% find best a
dat = [subimHL(:); subimLH(:)];
mm = max(abs(dat));
a = 1; astep = 0.1;
qtest = zeros(size(dat));
qtest(find(dat > a/2)) = a;
qtest(find(dat < -a/2)) = -a;
mse = mean((dat - qtest).^2);
mseold = mse+1;
while (mseold > mse) % loop through values
    mseold = mse;
    a = a+astep;
    qtest = zeros(size(dat));
    qtest(find(dat > a/2)) = a;
    qtest(find(dat < -a/2)) = -a;
    mse = mean((dat - qtest).^2);
end
a = a-astep
qtest = zeros(size(dat));
qtest(find(dat > a/2)) = a;
qtest(find(dat < -a/2)) = -a;
% probabilities, average bit rate, entropy
pposa = sum(qtest > 0)/length(qtest)
pnega = sum(qtest < 0)/length(qtest)
pzero = sum(qtest == 0)/length(qtest)
bbar = pposa*2 + pnega*2 + pzero
h = -(pposa*log(pposa) + pnega*log2(pnega) + pzero*log2(pzero))
% quantize HL and LH
subimHL = reshape(qtest(1:end/2),size(subimHL));
subimLH = reshape(qtest(end/2+1:end),size(subimHL));
% inverse transform
newho = zeros(size(ho));
newho(1:2:end,1:2:end) = (subimLL + subimHL + subimLH + subimHH)/2;
newho(2:2:end,1:2:end) = (subimLL - subimHL + subimLH - subimHH)/2;
newho(1:2:end,2:2:end) = (subimLL + subimHL - subimLH - subimHH)/2;
newho(2:2:end,2:2:end) = (subimLL - subimHL - subimLH + subimHH)/2;
%distortion
D = mean((ho(:)-newho(:)).^2)
figure(2); subplot(221);
imagesc(ho); colormap(gray); colorbar; title('Original Image');
subplot(222); imagesc(newho); colormap(gray); colorbar;
title(['Quantized Image, D = ' num2str(D)]);
subplot(223); imagesc(ho-newho); colormap(gray); colorbar;
title(['Error Image, D = ' num2str(D)]);
```

5. Systems and transforms.

a. $F(u,v) = 63\operatorname{sinc}(9u)\operatorname{sinc}(7v)$

b. Sampling with a 2D comb function yields:

$$f_s(x,y) = XY \sum_{n,m=-\infty}^{\infty} d(x-nX, y-mY)g(nX, mY)$$

$$= 4 \sum_{n,m=-\infty}^{\infty} d(x-2n, y-2m)\operatorname{rect}(2n/9)\operatorname{rect}(2n/7)$$

$$F_s(u,v) = \sum_{k,l=-\infty}^{\infty} G(u-\tfrac{k}{X}, v-\tfrac{l}{Y}) = \sum_{k,l=-\infty}^{\infty} 63\operatorname{sinc}(9(u-\tfrac{k}{2}))\operatorname{sinc}(7(v-\tfrac{l}{2}))$$

c. There are two correct answers for this part. First, we can the appropriate substitutions into $F_s(u,v)$:

$$F_d(w_x, w_y) = \frac{1}{XY} F_s(u,v)\Big|_{u=\frac{w_x}{2pX}, v=\frac{w_y}{2pY}}$$

$$= \sum_{k,l=-\infty}^{\infty} \frac{63}{4} \operatorname{sinc}(9(\frac{w_x}{4p} - \frac{k}{2}))\operatorname{sinc}(7(\frac{w_x}{4p} - \frac{l}{2}))$$

Alternatively, we can recognize that rect(2*n*/9)rect(2*m*/7) $= \begin{cases} 1 & |n| \le 2, |m| \le 1 \\ 0 & otherwise \end{cases}$, which transforms to $F_d(w_x, w_y) = \dfrac{\sin(5w_x/2))}{\sin(w_x/2)} \dfrac{\sin(3w_y/2)}{\sin(w_y/2)} = 15 \cdot \operatorname{diric}(w_x, 5)\operatorname{diric}(w_x, 3)$ or to $F_d(w_x, w_y) = (1 + 2\cos(w_x) + 2\cos(2w_x))(1 + 2\cos(w_y))$. While these three answers look very different, they are in fact exactly the same.

d. Using the Dirichlet form from part c., the discrete Fourier series coefficients can be written as: $\tilde{F}(k,l) = F_d(w_x, w_y)\Big|_{w_x=\frac{2pk}{N}, w_x=\frac{2pl}{M}} = \dfrac{\sin(\frac{5pk}{N}))}{\sin(\frac{pk}{N})} \dfrac{\sin(\frac{3pl}{M})}{\sin(\frac{pl}{M})}$

e. The DFT can be written as:

$$F(k,l) = \begin{cases} \tilde{F}(k,l) & k = 0,\ldots,N-1, l = 0,\ldots,M-1 \\ 0 & otherwise \end{cases}.$$