COMP 646 Assignment 3

Christopher Hundt 110220945

December 6, 2004

Question 1

(a) These filters are bandpass. As the center of the band gets closer to 0 the half-height width must be smaller to have a 1 octave bandwidth. The values of m that result in close to one octave bandwidth for various values of k_0 are plotted in figure 1. These were found by trying all combinations and for each k_0 picking the m that gives a bandwidth closest to 1. See also the attached `gabor.m`.

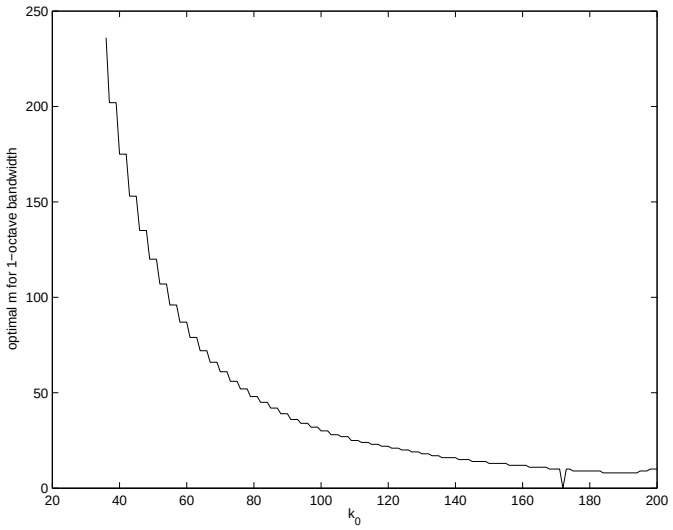


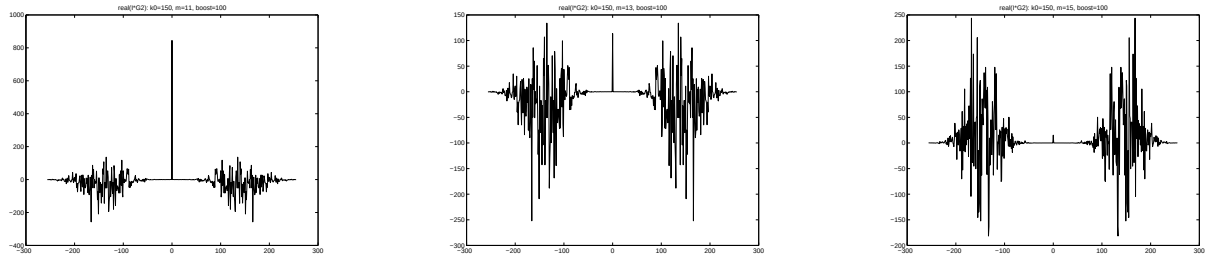
Figure 1: Values of m that yield a near-one octave bandwidth for various values of k_0

(b) We note that Gaussian fblurring is even in the x-frequency domain, in that $\hat{B}(k_x, k_y) = \hat{B}(-k_x, k_y)$. It follows that

$$\hat{f}_{\sin}(0,0) = i\left(\hat{B}(k_0,0)^m - \hat{B}(-k_0,0)^m\right) = i\left(\hat{B}(k_0,0)^m - \hat{B}(k_0,0)^m\right) = 0.$$

Thus changing the mean value will have no effect on the sine Gabor filter. On the other hand, the cosine Gabor filter will have a spike at $(k_x, k_y) = (0,0)$, where the value was 0 before. For smaller bandwidths (i.e., greater values of m), the spike will shrink because for $k_0 \neq 0$ $\hat{B}(0,0) < 1$ so raising $\hat{B}$ to higher powers results in a smaller value in the convolved image. These results are summarized in figure 2. See also the attached `gabor2.m`.

(c) In each case the phase disparities tended to group around two points, one at the phase shift ϕ corresponding to the motion, and one at $\phi + \pi$. See figure 3. Naturally, the phase shift increased as speed increased and also as filter frequency increased. For instance, if $k_0 = 40$ then one cycle is $512/40 \approx 13$



(a) bandwidth 1.15 ($m = 11$) (b) bandwidth 1.06 ($m = 13$) (c) bandwidth 0.97 ($m = 15$)

Figure 2: The effect on $\hat{G}_{\cos}(\cdot, 0) \cdot \hat{I}$ of adding 100 to the mean value of the image before filtering. For these examples $k_0 = 150$.

pixels so for $v_x = 3$ pixels the phase difference should be about 90 degrees, representing 1/4 of a cycle (figure 3(f)). On the other hand if $v_x = 2$ the phase difference should be about 60 degrees, 1/6 of a cycle (figure 3(e)), which is also what you might expect for $k_0 = 30$ and $v_x = 3$ (figure 3(c)). See also `gabor3.m`.

Question 2

(a) For the MATLAB code, see the attached `edgedetect.m`. The Gaussian filter code was taken from `http://www.cc.gatech.edu/classes/AY2000/cs7495_fall/participants/sashag/ps0/d2gauss.m`.

For an edge threshold, the mean of the gradient magnitude plus one standard deviation seemed to work. To demonstrate the results, the image is darkened and the edges shown in white superimposed over the image. See figure 4.

(b) The effect of increasing the standard deviation of the Gaussian blur was to increase the number of points which were detected as edges. This is because blurring has the effect of smoothing the image, which means that the first derivative will be also vary smoothly, so the second derivative will be smaller, and will be less likely to jump from positive negative without first passing close to 0. Thus the locus of points where it is near-zero will increase. See figure 4.

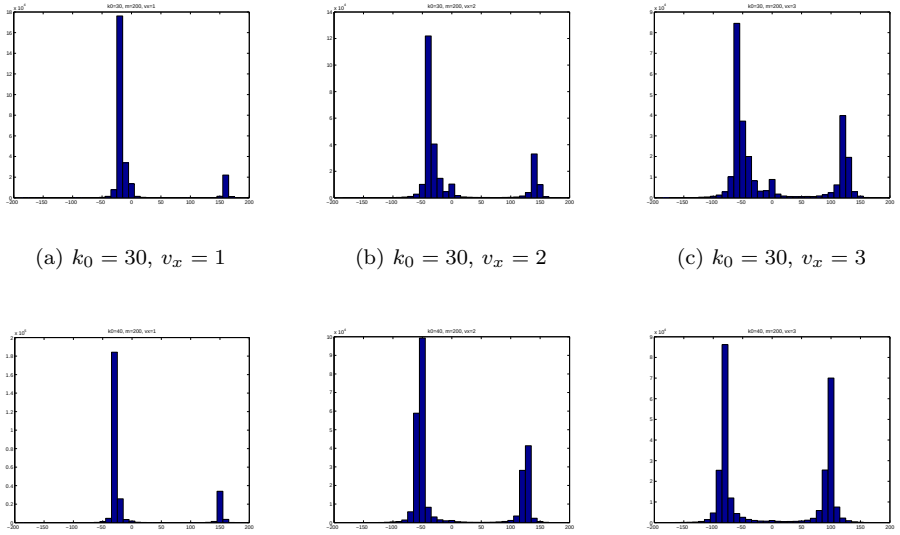
Question 3

By the inverse convolution theorem,

$$\mathbf{F} I(x) W(x) = \frac{1}{N} \hat{I}(k) * \hat{W}(k).$$

We note that

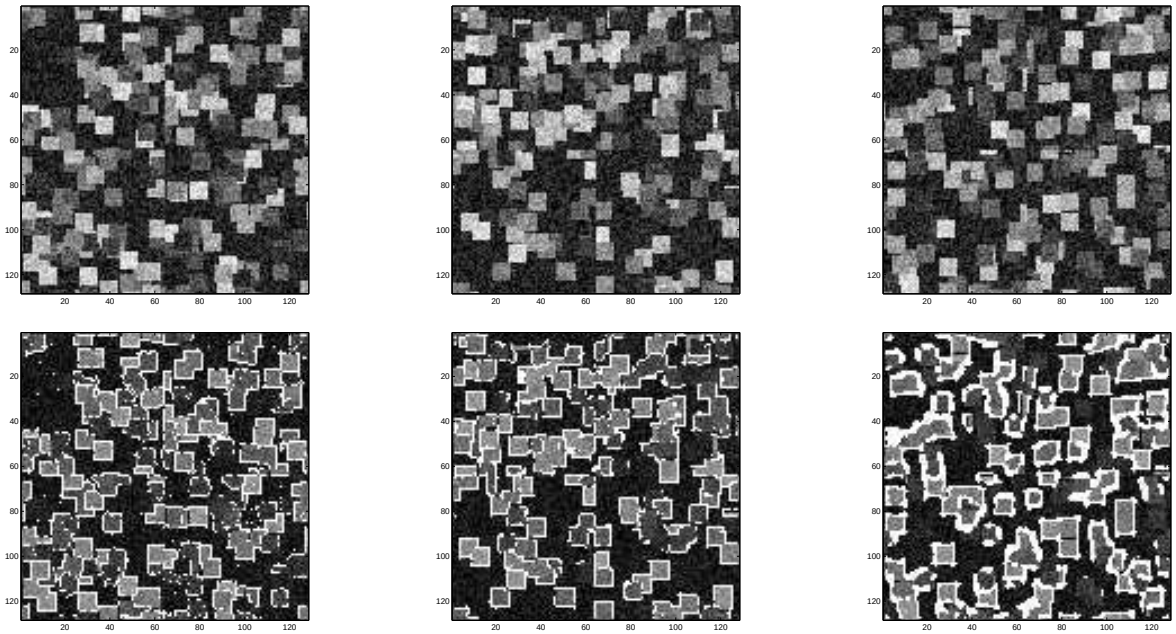
$$W(x) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{W}(k) = \frac{1}{N} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{W}(k) e^{i \frac{2\pi}{N} k x}$$



(a) $k_0 = 30$, $v_x = 1$ (b) $k_0 = 30$, $v_x = 2$ (c) $k_0 = 30$, $v_x = 3$

(d) $k_0 = 40$, $v_x = 1$ (e) $k_0 = 40$, $v_x = 2$ (f) $k_0 = 40$, $v_x = 3$

Figure 3: The effect of changing k_0 and v_x on the distribution of phase disparities. For all examples $N = 512$ and $m = 200$.



(a) $\sigma = 0.5$ (b) $\sigma = 1$ (c) $\sigma = 2$

Figure 4: Edges detected for various blur standard deviations. On top are the original images and below them are the originals darkened with the detected edge points coloured white.

and

$$\begin{aligned} W(x) &= 1 - \cos\left(\frac{2\pi}{N}x\right) \\ &= \frac{1}{N}\left[-\frac{N}{2}\cos\left(\frac{-2\pi}{N}x\right) + N - \frac{N}{2}\cos\left(\frac{2\pi}{N}x\right)\right] \\ &= \frac{1}{N}\left[-\frac{N}{2}\cos\left(\frac{-2\pi}{N}x\right) + N\cos(0) - \frac{N}{2}\cos\left(\frac{2\pi}{N}x\right)\right] \\ &= \frac{1}{N}\left(-\frac{N}{2}e^{i\frac{2\pi}{N}\cdot -1\cdot x} + Ne^{i\frac{2\pi}{N}0\cdot x} - \frac{N}{2}e^{i\frac{2\pi}{N}\cdot 1\cdot x}\right) \end{aligned}$$

since

$$i\left(-\frac{N}{2}\sin\left(\frac{2\pi}{N}x\right) + N\sin(0) - \frac{N}{2}\sin\left(\frac{-2\pi}{N}x\right)\right) = 0.$$

It follows that $\hat{W}(0) = N$, $\hat{W}(-1) = \hat{W}(1) = -N/2$, and $\hat{W}(k) = 0$ for all other values of k. When used in convolution, this defines a sharpening operation. Thus multiplying the image by W has the same effect as sharpening on $\hat{I}$. To demonstrate this, figure 5 shows a two-dimensional version of W applied to an image, and its effect on the Fourier transform of the image. For comparison, the effect of a sharpen convolution is also shown. See also `window.m`.

Question 4

(a) In analogy to the uniqueness constraint, we could make the assumption that a single sound source will only have a single disparity value, where disparity is not in terms of physical location but in terms of time between a sound reaching the left and right ears. This assumption is somewhat justified, for in the simple case (ignoring echoes, shape of the head, etc.) a given sound will reach the two ears at two specific times and it should not be possible to match that specific sound in more than one way. Thus if a sound is heard that seems to have more than one disparity, it is reasonable to attribute the sound to two sources, one for each disparity.

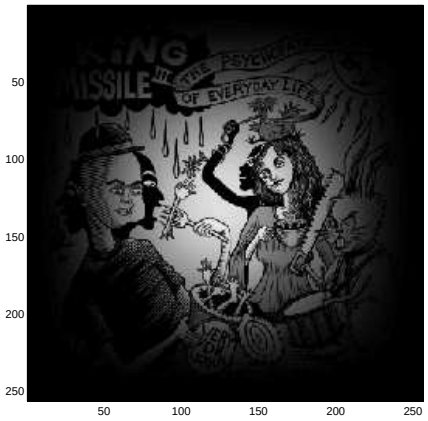
Then, the only way the disparity for a source will change would be for the source to move relative to the ears, and such movement should occur continuously. Thus if the disparity of a sound suddenly changes it is reasonable to suppose that the new disparity is due to a different sound. This is analogous to the continuity constraint in Marr and Poggio's paper.

(b) These constraints break down if we do not ignore complicating factors. For example, echoes will essentially destroy the uniqueness constraint. As a sound bounces around a room, the disparities of the echoes heard by the ear can vary wildly and even change in sign (for instance, if a sound source is to left of the head and bounces off a wall to the right of the head). Yet in most cases the echoes are not heard separately but are attributed to the single source. In this sense the source has multiple disparities.

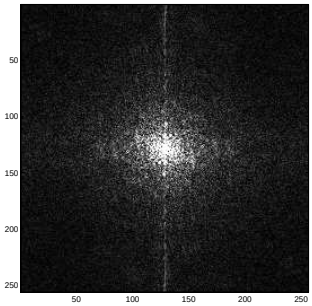
The uniqueness constraint can also be affected by the shape of head and shoulders, particularly if the sound is changing pitch or has multiple frequencies. In such a case some frequencies will reach the ears at different relative times than other frequencies, so there will be multiple disparities for a single source corresponding to different frequencies.

In both these cases, changing the frequency composition and location of the source can have sharp effects on the disparity. For example, some echoing, resonance, or damping effects might occur only for a narrow range of frequencies or only for certain locations of the source (or both). As the source enters the "target" location or frequency, there would be a sudden change in disparity for some frequencies without any corresponding change in source.

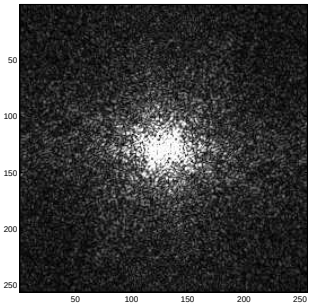
(a) The original image



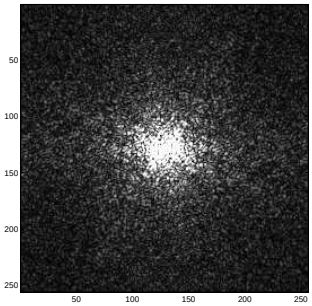
(b) The filtered image



(c) The Fourier transform of the original image



(d) The Fourier transform of the filtered image



(e) The Fourier transform of the original image, convolved with a sharpening function (and scaled)

Figure 5: The effect of a two-dimensional window function on an image and its Fourier transform

```
%  gabor.m
%
%  Find Gabor filters with bandwidth 1.

N = 512; range = (-N/2:N/2-1);

% Set up blur
B = zeros(N,1);
B(1) = .5; B(2) = .25; B(N) = .25;
FB = fft(B);

testk0=49; testm=128;

bestm = zeros(N/2,3);
for k0 = 1:N/2;
   bandwidth = zeros(N,2);
   for m = 1:N/2;
      % compute Gabor filter
      G2 = power(circshift(FB,k0),m) + power(circshift(FB,k0),m);
      x1 = [0 1.5];
      x2 = [0 1.5];
      x1found = 0;
  % loop through, looking for half-height points
      for i = 1:N/2;
         current = G2(i);
         if (x1found == 0);
            if (abs(current-0.5)<=abs(0.5-x1(2)));
               x1(2) = current;
               x1(1) = i;
            else;
               x1found = 1;
            end
        elseif (abs(current-0.5)<=abs(0.5-x2(2)));
            x2(2) = current;
            x2(1) = i;
        end
      end
      bandwidth(m,:) = [log(x2(1)/x1(1))/log(2) abs(x1(2)-0.5)+abs(x2(2)-0.5)];
  % record m-values where half-height is closest to 1
      if (abs(bandwidth(m,1)-1) < abs(bestm(k0,2)-1));
         bestm(k0,2) = bandwidth(m,1);
         bestm(k0,1) = m-1;
         bestm(k0,3) = bandwidth(m,2);
      end
   end
end
```

```
%  gabor2.m
%
%  Raise mean value and find the effect on the image convolved with Gabor filters.

N = 512;
num_tiles = N/4; width = 32;
maxnoise = 0;  %  Between 0 and 1.  Low maxnoise means low noise.

boosts = [100];
range = (-N/2:N/2-1);

% make blur
B = zeros(1,N);
B(1) = .5; B(2) = .25; B(N) = .25;
B = B'*B;
FB = fftn(B);

testk0 = 150;
testms = [11 12 13 14];

% Make image
Ilayer = zeros(N+3*width,N+3*width);
for n = 1:num_tiles;
   x = ceil((N+2*width)*rand);
   y = ceil((N+2*width)*rand);
   Ilayer(x+1:x+width,y+1:y+width) = rand;
end
Icrop =  Ilayer(width+1:N+width,width+1:N+width);
Icrop = Icrop + maxnoise * 2*(rand(N,N) - .5);

for i =1:length(testms);
   testm = testms(i);
   for j = 1:length(boosts);
      boost = boosts(j);

      Icrop = Icrop + boost; % add extra to mean value
      FI = fftn(Icrop);
      Icrop = Icrop - boost; % restore original (we only need FI)

      % make Gabor filters
      G1 = sqrt(-1)*(power(circshift(FB,[0 testk0]),testm) - \
           power(circshift(FB,[0 -testk0]),testm));
      G2 = power(circshift(FB,[0 testk0]),testm) + \
           power(circshift(FB,[0 -testk0]),testm);

      FIG1 = FI.*G1; FIG2 = FI.*G2;

      figure; plot((-N/2:N/2-1),fftshift(real(FIG2(1,:))));
      title(sprintf('real(I*G2): k0=%d, m=%d, boost=%d',testk0,testm,boost));
    end
end
```

```
%  gabor3.m
%
%  Compute phase disparities for gabor-filtered images
%  from shiftimage.m.

N = 512;

num_tiles = N/4;
width = 32;
maxnoise = 0;  %  Between 0 and 1.  Low maxnoise means low noise.
variance = maxnoise^2/3;

boost = 100;

range = (-N/2:N/2-1);

B = zeros(1,N);
B(1) = .5;
B(2) = .25;
B(N) = .25;

B = B'*B;

FB = fftn(B);

testk0s = [20 30 40];
testms = [200 200 200];

vxs = [1 2 3];

% make the image
Ilayer = zeros(N+3*width,N+3*width);
for n = 1:num_tiles;
   x = ceil((N+2*width)*rand);
   y = ceil((N+2*width)*rand);
   Ilayer(x+1:x+width,y+1:y+width) = rand;
end
Icrop =  Ilayer(width+1:N+width,width+1:N+width);
clear Ilayer
Icrop = Icrop + maxnoise * 2*(rand(N,N) - .5);

disparities = zeros(N,N,length(testms),length(vxs));

for i =1:length(testms);
   testm = testms(i);
   testk0= testk0s(i);
   for j = 1:length(vxs);
      vx = vxs(j);

      tempIcrop = Icrop + boost;
      tempIcrop = tempIcrop-mean(tempIcrop(:));
```

```
      FIL = fftn(tempIcrop);
      FIR = fftn(circshift(tempIcrop, [0 vx]));

      G1 = sqrt(-1)*(power(circshift(FB,[0 testk0]),testm)
           - power(circshift(FB,[0 -testk0]),testm));
      G2 = power(circshift(FB,[0 testk0]),testm)
           + power(circshift(FB,[0 -testk0]),testm);

      FIRG2 = FIR.*G2;
      FIRG1 = FIR.*G1;
      FILG2 = FIL.*G2;
      FILG1 = FIL.*G1;

      disparities(:,:,i,j) = atan(real(ifftn(FILG1))./real(ifftn(FILG2)))
                             - atan(real(ifftn(FIRG1))./real(ifftn(FIRG2)));

      figure; data=real(disparities(:,:,i,j))/pi*180; hist(data(:),[-180:10:180]);
      title(sprintf('k0=%d, m=%d, vx=%d',testk0,testm,vx));
      print('-deps', sprintf('gabor%d-%d', testk0, vx));
      close;
   end
end
```

```
%  edgedetect.m
%
%  Take a single frame from shiftimage.m and detect its edges.

clear

N = 128;
k = 10;
epsilon = 0.5;
blurvariance = 2;

num_tiles = 2*N;
width = 8;
maxnoise = 0.2;  %  Between 0 and 1.  Low maxnoise means low noise.

% Generate the image
Ilayer = zeros(N+3*width,N+3*width);
for n = 1:num_tiles;

      x = ceil((N+2*width)*rand);
      y = ceil((N+2*width)*rand);

      Ilayer(x+1:x+width,y+1:y+width) = rand;
end
Icrop =  Ilayer(width+1:N+width,width+1:N+width);
clear Ilayer
Icrop = Icrop + maxnoise * 2*(rand(N,N) - .5);
FI = fftn(Icrop);

% Make derivative filters
Dx = zeros(N,N);
Dx(1,2) = -1;
Dx(1,N) = 1;
FDx = fftn(Dx);

Dy = zeros(N,N);
Dy(2,1) = -1;
Dy(N,1) = 1;
FDy = fftn(Dy);

% Make Gaussian filter
FG = fftn(d2gauss(N,blurvariance,N,blurvariance,0));

% Get direction of derivative at every pixel
xderiv = ifftn(FDx.*FG.*FI);
yderiv = ifftn(FDy.*FG.*FI);
grad = atan(real(yderiv)./real(xderiv));

% Compute second derivatives
Gx2 = ifftn(power(FDx,2).*FG.*FI);
Gy2 = ifftn(power(FDy,2).*FG.*FI);
```

```
Gxy = ifftn(FDx.*FDy.*FG);

% Make steerable filter
d2G = power(cos(grad),2).*Gx2 + power(sin(grad),2).*Gy2 + 2*cos(grad)*sin(grad).*Gxy;

% Define minimum gradient magnitude as mean + 1 deviation
threshold = abs(mean([xderiv(:)' yderiv(:)'])) + std([xderiv(:)' yderiv(:)']);

% Find edges
edges = zeros(N,N);
for i = 2:N-1;
   for j = 2:N-1;
      if (((abs(xderiv(i,j)) > threshold) || (abs(yderiv(i,j)) > threshold))
          && abs(real(d2G(i,j)))<epsilon);
         edges(i,j) = 1;
      end
   end
end

% Display/print image
todisplay = fftshift(Icrop);
figure;
imagesc(todisplay);
%print -deps original-2.eps
axis('square');
colormap('gray');

% Display/print image with edges outlined
figure;
withedges = todisplay+edges*1.5*max(todisplay(:));
imagesc(min(1.5*max(todisplay(:)),withedges));
%print -deps edges-2.eps
axis('square');
colormap('gray');
```

```
%  window.m
%
%  Make the window function and observe its effect on the
%  Fourier transform of the image.

N = 256;

W = zeros(1,N);

for x = 1:N;
    W(x) = 1-cos(2*pi*x/N);
end

W = W' * W;

I = double(imread('KM-III-CD-CVR.gif'));

new = I.*W;

FI = fft2(I);

Fnew = fft2(new);

figure;
imagesc(fftshift(min(100000,abs(FI))));
axis('square');
colormap('gray');
print -deps window-orig.eps;

figure;
imagesc(fftshift(min(100000,abs(Fnew))));
axis('square');
colormap('gray');
print -deps window-filtered.eps;

% sharpening function:
S = zeros(1,N);
S(1) = .5;
S(2) = -.25;
S(N) = -.25;
S = S' * S;

check = ifft2(fft2(FI).*fft2(S));

figure;
imagesc(fftshift(min(25000,abs(check))));
axis('square');
colormap('gray');
print -deps window-check.eps;
```