

# Computational Vision

## Assignment #2--Laplacian and Gabor filters, FFT's

Due: Tuesday, March 16

In this assignment, we will explore the consequences of filtering images with different sizes of filters that have the form of a Laplacian of a Gaussian. We will also start working with fast Fourier transforms (fft's) and examine what happens to the frequency spectrum of images that are filtered with Gabor filters, similar to the receptive field organizations of simple cells in the striate cortex.

There are several points in this assignment where it is useful to view your images. You can view images in matlab using the function: **pcolor**. To view an image specified by the matrix, ImageMat, use the command:

**pcolor(ImageMat)**

You will need to specify a gray-scale colormap with the command:

**colormap(gray)**

Also, you can use the following commands to make your image look more like the one specified in the matrix:

```
axis square           %makes the vertical and horizontal dimensions equal
axis ij              %puts the image position 1, 1 at the top left corner
shading flat        %does not draw black grid lines between values
shading interp     %like shading flat, but better interpolation. Slower, too
axis([-128 128 -128 128]) %Gives a good sized image if your image window
                        %is slightly more than 1/2 the width of the screen.
```

You will be using two filters that have the shape of the Laplacian of a Gaussian. One has an inner diameter of about 4 pixels (and is called op4) and one has an inner diameter of about 8 pixels (op8). The MATLAB scripts for creating these filters are on the last page of this assignment.

### Problem 1: Zero Crossings at different scales.

In class we learned that using convolution operators of different sizes can allow us to detect image intensity changes that occur on different scales. Larger convolution operators perform more extensive smoothing of the image and are less sensitive to noise, but they have the disadvantage that they do not localize the positions of intensity edges as well as smaller operators. This problem explores these two properties.

#### A. Different sizes of operators:

Make a matrix with dimensions 100 x 100 that contains image intensity values for a square of intensity 100 against a gray background of intensity 50. The square should be 30 x 30 pixels and should be placed with its upper left corner at position (41, 41) in the image matrix. Thus, it should cover the region specified by Image(41:70, 41:70). Use the scripts at the end of this assignment to create 2 filters with the Laplacian of a Gaussian shape. Convolve your image with the smaller of the two filters (op4). You may use the MATLAB function **conv2**, the 2D convolution function, to do this. The **conv2** function takes two parameters representing the two functions to be convolved. In this case one is the image and the second is the operator that you are convolving the image. To convolve myImage with myFilter, use the following function call:

**conv2(myImage, myFilter);**

Print out the values in the region around the upper left-hand corner of the square (around position 40, 40) and note where the zero crossings occur. Note that the position of the edges will be shifted to the right and down by half the width of the operator (about 5 positions for the small operator) because of the way conv2 works. You should be able to see zero crossings for the rows and

columns at (40:50, 44:49). Now repeat the process with the large convolution operator (op8). Here the edge is shifted more. Try observing the region for rows and columns at: (50:60, 50:55). Note that there will be several rows and columns of zeros at the edges of the convolved images. These are not part of the zero crossings. The zero crossings are locations where there is a change from positive non-zero values to negative non-zero values or vice versa.

**Turn in:**

The commands you used to make the square image and perform the convolution.

A printout of the values of the two filtered images around the upper left corner of the internal square. (A 10 x 6 matrix should be sufficient to see the zero crossings at the corner and still print out in one set of columns).

On each printout of values, draw in the positions of the zero crossings and turn it in. (You can check your answer using the zero crossing script at the end of this assignment).

Turn in the answers to the following questions:

i) How do the zero crossings obtained from the large convolution operator compare to those obtained from the small one?

ii) What can you conclude about the ability of each of these operators to locate the position of intensity edges around features such as corners?

**B. Noisy images.**

Construct a noisy image using the function **rand**. This function creates a matrix of random values between 0 and 1. To create a matrix of values between upper and lower limits around some mean, you can call rand as follows:

**noiseMatrix = mean\*ones(i, j) + 2\*spread\*rand(i, j) - spread;**

where i and j are the number of rows and columns and the spread is the distance from the mean to the upper (or lower) limit of the noise.

Make a noisy square image (of dimensions 100 x 100) like that above, with the background at an intensity of 50, with a random variation with a spread of 20 above and below (i.e. the range will be between 30 and 70). Make the internal square (with dimensions 30 x 30, positioned the same as in part A) with a mean of 100 and a spread of 40 (i.e. the range will be between 60 and 140). Turn in the commands you use to create this image.

Now repeat the convolutions as in part A. Print out the values of the filtered images around the region of the corner of the square, as in part A. Draw the locations of the zero crossings on these printouts for the two filtered images.

**Turn in:**

The commands you used to make the square image and perform the convolution.

The printout of the values of the filtered images, with the zero crossings drawn in.

The answer to the following question: What can you say about the ability of the two operators to locate intensity edges in the presence of noise?

**C. Thresholding.**

Note that both the operators leave many more zero crossings in their output than there are actual edges in the original image. You can eliminate some of these spurious edges by thresholding. You can threshold the filtered image as follows:

**thresholdImage = round(Image/Threshold);**

This will eliminate zero crossings that occur from small fluctuations in image intensity.

Threshold the noisy image filtered by the big convolution operator (op8) with a threshold of 2000. Run the zero crossing function on both the original and the thresholded filter images. Print out the zero crossings for the rows and columns at: (50:60, 50:60).

Now do the same for the image filtered by the small operator (op4). Print out the zero crossings for the rows and columns for the range: (40:50, 44: 49). Try a threshold of 2000 and a threshold of 500.

**Turn in:**

A printout of the zero crossings around the corner in the 2 cases for the op8 filter (original and thresholded) .

A printout of the zero crossings around the corner for op4 without a threshold.

The answers to the following questions:

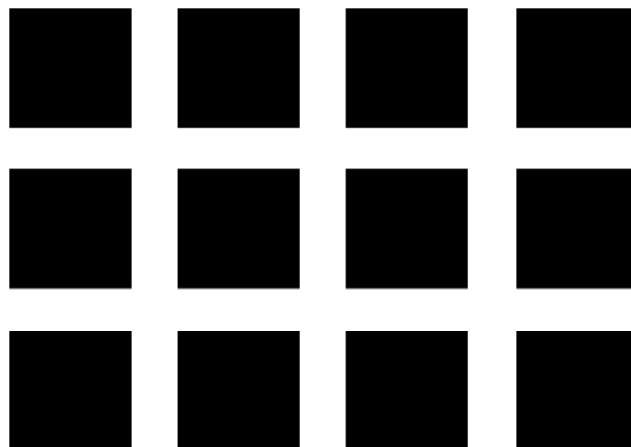
For the small operator, op4, can you find a threshold that eliminates the noisy zero crossings while preserving the real edge for this filtered image?

What does the answer to the previous question tell you about the effect of image noise on small convolution operators?

How does the effect of noise differ for the large and small operators?

**Problem 2: The Hermann Grid Illusion**

This problem explores the famous contrast illusion known as the Hermann grid. While reading a book on sound by John Tyndall in 1869 that contained a grid of bright lines on a black background, illustrated below, Hermann noticed that dark shadowy dots appeared at the intersections of the bright horizontal and vertical lines. Hermann reported his observation in 1870, and since this time the so-called Hermann grid illusion has been much studied and debated by vision scientists.



The following Matlab script will make a Hermann grid:

```
A = zeros(8, 8);  
B = ones(8, 4);  
C = ones(4, 12);  
squaremat = [A B; C]  
hermangrid = [squaremat squaremat; squaremat squaremat];  
hermangrid = [hermangrid hermangrid; hermangrid hermangrid];
```

Use **pcolor** to make this into an image. See if you can notice the dark circles at the intersections. You will see them better a bit away from where you are looking. If Matlab scales the image too big, you can see the circles by viewing the image from a distance.

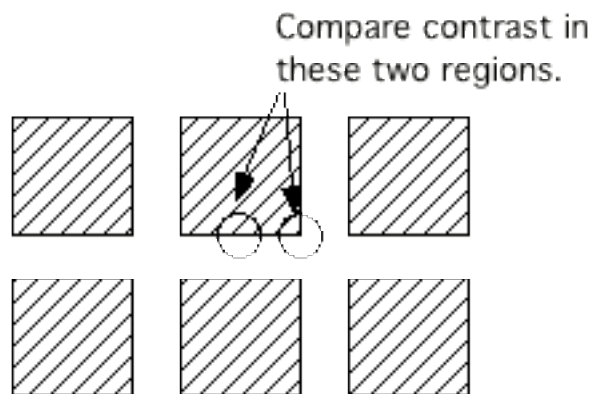
Create a filtered version of this image with the `op4` filter.

**A.** Write a function to compute the contrast at the zero crossings in this image (The function must find the zero crossings first, and then compute the contrast). The contrast is given as follows:

$$contrast = \sqrt{(dx)^2 + (dy)^2}$$

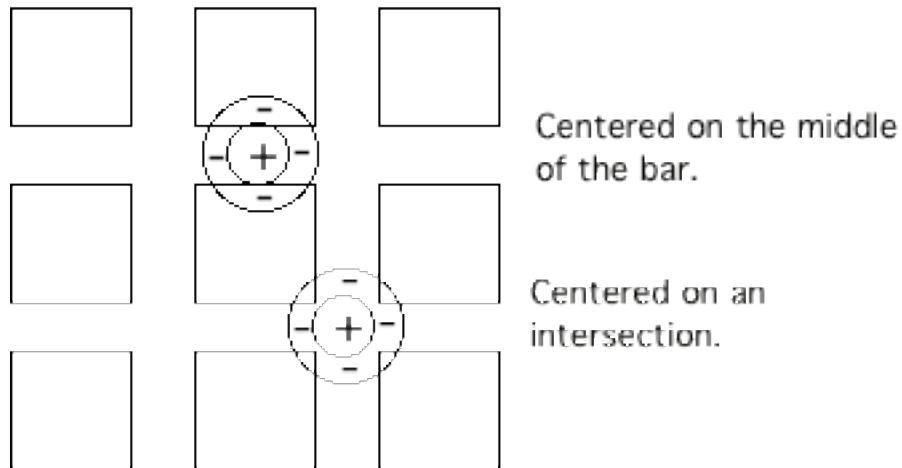
where  $dx$  is the difference between the two convolution values obtained at two adjacent locations in the  $x$  direction and  $dy$  is the difference between the two convolution values obtained at two adjacent locations in the  $y$  direction. This contrast is the magnitude of the gradient of the convolution result. Place the value of the contrast (rounded to an integer) in the position of the zero crossing in the output matrix (and put zero at the other positions). Turn in your code for this function.

Run your new contrast measure on your filtered grid image and observe the value of the contrast computed in the vicinity of an intersection of the grid and in the vicinity of the middle of a horizontal or vertical bar of the grid. These two locations are indicated in the diagram below:



Turn in a description of how the contrast differs for these two regions.

**B.** Turn in a printout of the values for a  $12 \times 12$  piece of the output showing one iteration of the grid. Explain why the contrasts are different in the vicinity of the grid intersections and along the edges between the intersections using a geometric argument. To assist you in answering this question, the following diagram shows the placement of the convolution operator when computing the convolution values for these two regions:



**Turn in:**

A printout of the contrast function you wrote for part A.

A description of the contrast differences from part A.

A 12 x 12 printout of the output values of your contrast function executed on the filtered image.

A written explanation of why the contrasts differ between the center of a bar and the intersection of 2 bars.

**Problem 3. Introduction to Gabor filters and Fast Fourier Transform (fft).**

**A. Creating a sinewave grating and finding its Fourier Transform.**

Use the Matlab function `linspace` to create a vector with 128 evenly spaced divisions between 0 and  $2\pi$  as follows:

```
A = linspace(0, 2*pi, 128);
```

Now create a sampled sinewave by taking the sine of the vector A:

```
B = sin(10*A);
```

Note that this sine wave has 10 cycles per image-length (which is 128 pixels).

Now create a 2D image of a 1D sinewave by repeating the row vector, B, 128 times:

```
for i = 1:128  
    sinImage(i, :) = B;  
end
```

Use `pcolor` to view the image.

Now use `fft2` (2D fast fourier transform) to view the frequency spectrum of the `sinImage`. `fft2` returns complex numbers for each frequency, but here we are interested only in the amplitude, so you can use the function `abs` to get the magnitude of the frequency function at each position. `fft2` puts the zero frequency (DC component) in the upper left corner of the output matrix and the first frequency component (1 cycle per image) in the next position, and so on. The negative frequencies are in the right half of the output matrix (which is symmetric to the left half). To view the Fourier spectrum with the zero in the center of the image, use the function `fftshift`. Thus, to view the spectrum of `sinImage`, the command would be:

```
fftImage = fftshift(abs(fft2(sinImage)))
```

View the fft of the image with `pcolor`. The DC component is at the center. You should see a dot at positions along the horizontal midline at  $\pm 10$ . Hand in a printout of this image.

### **B. Creating a Gabor filter and its Fourier transform.**

You can create an image of a gabor function by multiplying `sinImage` with a gaussian (using the `.*` for pairwise multiplication). Create the Gaussian as follows:

```
gfilter = zeros(128, 128);
for i = 1:128
    for j = 1:128
        gfilter(i, j) = 16*exp(-((i-64.5)^2 + (j - 64.5)^2)/16);
    end
end
```

View the image of your gabor filter with `pcolor`. Hand in a printout of the filter.

Perform an `fft2` on the gabor filter. View this with `pcolor`. Hand in a printout of the `fft2` image.

Turn in the answer to the following question:

How does the frequency spectrum of the gabor filter compare to the spectrum of the sinewave?

### **C. Creating a plaid grating and its Fourier transform.**

Create a plaid grating by adding together your vertical sinewave with a horizontal sinewave;

```
grating = sinImage + sinImage';
```

View this with `pcolor`.

Find the frequency spectrum for this plaid grating. Hand in a printout of the frequency spectrum.

Now create a different type of plaid grating by multiplying the horizontal and vertical gratings.

```
grating = sinImage .* sinImage';
```

View this with `pcolor`.

Find the frequency spectrum of this new plaid grating. Hand in a printout of the frequency spectrum.

Hand in the answer to the following questions:

How do the frequency spectra of the two plaid gratings compare?

What do you think would result from a convolution with the gabor filter you created above for each of the two plaid gratings, and why?

Convolve the two plaid gratings with the gabor filter. Print out the resulting images and frequency spectra. Do they fit with your prediction?

#### **Turn in:**

- A printout of the Fourier transform of the original sinewave image (part A).
- A printout of the Gabor filter image (part B).
- A printout of the Fourier transform of the Gabor filter (part B).
- A printout of the frequency spectra for each of the 2 plaid gratings in part C.
- A printout of the plaid images after convolution with the Gabor filter.
- A printout of the frequency spectra of the plaids convolved with the Gabor filter.
- The answers to the questions in parts B and C.

```

%Script to create the laplacian of a Gaussian function with inner diameter 4
op4 = zeros(11,11);
for i = 1:11
    for j = 1:11
        rsquare = (i- 6)^2 + (j-6)^2;          %distance from center of filter
        invsigsquare = 0.5;          %1/(sigma)^2
        op4(i,j) = round(25*(2-rsquare*invsigsquare) *exp(-0.5*rsquare/2.0));
    end
end
op4(6,6) = 48;          %This slight change to keep the filter balanced.

```

```

%Script to create the laplacian of a Gaussian function with inner diameter 8
op8 = zeros(23,23);
for i = 1:23
    for j = 1:23
        rsquare = (i- 12)^2 + (j-12)^2;
        invsigsquare = 0.125;
        op8(i,j) = round(25*(2-rsquare*invsigsquare)*...
            exp(-invsigsquare*rsquare/2.0));
    end
end
op8(11,12) = 42;          %Again, some small changes for balance.
op8(13, 12) = 42;
op8(12, 11) = 42;
op8(12,13) = 42;
op8(12,12) = 44;

```

```

%Simple zero crossing detector for a matrix named filtsquare
zerocross = zeros(65,65);
for i= 1:64
    for j = 1:64
        if filtsquare(i,j) * filtsquare(i+1,j) < 0
            zerocross(i,j) = 1;
        elseif filtsquare(i,j) * filtsquare(i,j+1) < 0
            zerocross(i,j) = 1;
        else
            zerocross(i,j) = 0;
        end
    end
end
end

```