

HW 2 - Gaussian, Derivative of Gaussian, Laplacian and Circularly Symmetric Filters

HW 2 due - March 4, 2008.

1. Gradient of an Image

Let us first consider a function depending on one variable. As an example we consider

$$f(x) = 1 + x - x^2$$

on the interval $0 \leq x \leq 1$. We can visualize the function geometrically by plotting it.

We can also give the function a physical interpretation. We can interpret the function as describing the temperature in a rod. According to the graph the rod is cooler towards the ends and is hotter around $x=1/2$.

In order to further study a function, we can take its derivative. For the function, $f'(x)=1-2x$. Whenever the derivative is close to zero at a point, a function is “flat” at that point. We see that the $f'(1/2)=0$ which agrees with the plot. According to the plot the graph is indeed flat at $x=1/2$. A large absolute value of the derivative indicates a steep slope of the function, as near $x=0$ and $x=1$ in the figure above. If the function represents the temperature in a rod, the magnitude of the derivative indicates “how fast the temperature is changing” while moving along the rod.

The fact that the derivative measures how fast something is changing is one of the key for image processing as we will see later. However, in order to study images, we first need to understand functions of two variables. As an example, consider

$$f(x, y) = x^2y^2 \quad -1 \leq x, y \leq 1$$

Geometrically this function can be visualized as surface.

Expt. 1

In MATLAB do the following:

```
[X,Y] = meshgrid(-1: .1: 1, -1: .1 : 1);
```

```
Z=(X.^2).*(Y.^2);
```

```
surf(X,Y,Z)
```

Note that X, Y are matrices and we are implementing $f(x, y) = x^2y^2$ on the grid defined above. The *surf* function creates a 3-dimensional surface of the function Z over a rectangular region. Note also that rather than the *surface* plot of Z , we can get its 2-D plot by *imagesc(Z)* or *imshow(Z)*.

We can also give a function of two variables a physical interpretation. We can think of a function $f(x,y)$ as describing the temperature in a plate. For each coordinate (x,y) of the plate, the function $f(x,y)$ returns the temperature. We saw earlier that the derivative can measure the “steepness” of a function of one variable. Can we obtain a similar measure for a function of two variables? The answer is yes. We can compute the partial derivatives of the function $f(x,y)$:

$$\frac{\partial f}{\partial x} = 2xy^2$$

$$\frac{\partial f}{\partial y} = 2x^2y$$

The partial derivative with respect to x measures how fast the function is changing in the x -direction and the partial derivative with respect to y measures how fast the function is changing in the y -direction. We see that in our example, the partial derivatives are zero along the coordinate axes $x=0$ and $y=0$. This agrees with our surface plot where it seems like the altitude of the surface is not changing if we imagine our-selves walking along any of the axes $x=0$ or $y=0$. By using the partial derivatives of a function we can introduce the *gradient vector* which we will denote as ∇f . The gradient vector is defined as

$$\nabla f(x, y) = \frac{\partial f}{\partial x}(x, y)i + \frac{\partial f}{\partial y}(x, y)j$$

Note that the gradient is a vector, that is, it has both a direction and a magnitude. The gradient has an important interpretation:

1. The direction ∇f of points in the direction of steepest slope uphill.
2. The magnitude of ∇f is

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

The magnitude of ∇f gives a measure of “how steep is the slope” in the direction of the gradient.

In our example (see equations 2 and 3 above) we have that at $x=y=1/2$

$$\nabla f = \frac{1}{4}i + \frac{1}{4}j$$

This means that if we stand on the surface at $x=y=0.5$ the steepest slope uphill is in the “north-east” direction which agrees with the surface plot above. The magnitude of the gradient can be thought of as a “steepness measure”. In other words, the magnitude of the gradient measures how much a function is changing at different points.

For our function $f(x, y) = x^2y^2$ we find that

$$|\nabla f| = 2\sqrt{x^2y^4 + y^2x^4}.$$

If we assign the color black to a point (x,y) where the magnitude of the gradient is zero and gradually assign brighter colors to points of increasingly larger gradient magnitude, we get the following “gradient map”.

The brighter a point is in the *gradient map*, the steeper is the slope of the function that we are measuring the gradient of. The gradient map above is brighter close to the corners where the function we are studying changes relatively fast. This agrees with the *surface plot* above, where it is clear that the function is steeper near the corners. So how can we use the ideas from the previous section for image processing? The idea is the following:

An image is a function of 2 variables Now comes the reason why this is so important:

- Since an image is a function of two variables, we can use the gradient to study it!

- An edge in an image is a region where we have sharp contrast, that is, a rapid change in color intensity. A rapid change in a functions value gives a large magnitude of the gradient.
- The gradient of an image will have large magnitude at edges!
- If the color is roughly constant in an area of the image, the function is roughly constant in that area. A function that is roughly constant gives a small value of the gradient in that area.

We now determine the gradient image of the previous example. Here we plot the contour surfaces of the gradient. Can you plot this gradient as a 2-D image? Then we show the *direction* of the gradient vector and its magnitude using Matlab's *quiver* function.

Expt. 2

In MATLAB do the following:

```
[X,Y] = meshgrid(-1: .1: 1, -1: .1 : 1);
```

```
Z=(X.^2).*(Y.^2);
```

```
[fx,fy] = gradient(Z);
contour(Z);
x=-1: .1: 1;
figure; contour(x,x,Z);
hold on; quiver(x,x,fx,fy);
hold off
```

The idea that the gradient of an image contains information about edges, can be used to extract an edge from an image. Before considering edge extraction, we look at the Gaussian function.

2. The Gaussian filter

Gaussian smoothing is used in many image processing applications. A particular application is pre-filtering before edge detection. The Gaussian filter is in fact the optimal smoothing filter for edge detection under the criteria used to derive the Canny edge detector. We experiment here with the Gaussian, Derivative of Gaussian, Laplacian filters and finally the circularly symmetrical filters.

Consider first the 1-D Gaussian distribution:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. We have also assumed that the mean of the distribution is zero. If X is a normally distributed random variable, then the probability density function of X is a gaussian function.

In 2-D, the Gaussian distribution is isotropic. That is, it has the same property in all directions. The property here is that of symmetry. Hence we have a circularly symmetric distribution:

$$G(x, y) = \frac{1}{(2\pi\sigma^2)} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

In image processing, we use the discrete version of the Gaussian function $G(x, y)$ as the impulse response $g[m, n]$ of a 2-D digital filter. In theory, the Gaussian distribution is non-zero everywhere, implying an

infinitely large filter. In practice, it is zero when we are more than 3 standard deviations away from the mean, so we can truncate the filter at this point. Below we see a integer-valued convolution kernel $g[m, n]$ that approximates a Gaussian with $\sigma = 1$

$$\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix} \frac{1}{23}$$

The non-isotropic 2-D Gaussian function can be written as

$$f(x, y) = Ae^{-\left(\frac{(x-x_o)^2}{2\sigma_x^2}\right) - \left(\frac{(y-y_o)^2}{2\sigma_y^2}\right)}$$

More generally, allowing for cross-terms, we can write the 2-D Gaussian function as

$$f(x, y) = Ae^{-\left(a(x-x_o)^2 + b(x-x_o)(y-y_o) + c(y-y_o)^2\right)}$$

Expt. 3

In MATLAB do the following:

- Obtain the surface map of the 2-D Gaussian function, using the *surf* function. Look at a 3×3 grid, 5×5 grid with various values for σ .
- By changing parameters a, b, c in the general formula, you can see an interesting display of the rotated, non-isotropic Gaussian functions. Use the following code:

```
A = 1; \
x0 = 0; y0 = 0; \
for theta = 0:pi/100:pi \
sigma_x = 1; \
sigma_y = 2; \
a = (cos(theta)/sigma_x)^2 + (sin(theta)/sigma_y)^2; \
b = -sin(2*theta)/(sigma_x)^2 + sin(2*theta)/(sigma_y)^2; \
c = (sin(theta)/sigma_x)^2 + (cos(theta)/sigma_y)^2; \

[X, Y] = meshgrid(-5:.1:5, -5:.1:5); \
Z = A*exp( - (a*(X-x0).^2 + b*(X-x0).*(Y-y0) + c*(Y-y0).^2) ); \
surf(X,Y,Z); shading interp; view(-36,36); axis equal; drawnow \
end
```

Once the discrete 2-D Gaussian $g[m, n]$ function has been obtained, it can be treated as a 2-D impulse response $h(m, n)$. Gaussian smoothing can then be performed on the image by convolving it with the filter. Since the filter is separable, filtering consists of 2 1-D convolutions. (Note that the Gaussian is in fact the *only* completely circularly symmetric operator which can be decomposed this way).

The Gaussian filter not only has uses in image processing, but also has a biological plausability. Computational biologists claim that some cells in the visual pathway often have an approximately Gaussian response.

Since the 2-D Fourier transform of the 2-D Gaussian function is also a 2-D Gaussian function, we have a 2-D low-pass filter. The degree of smoothing is determined by the standard deviation of the Gaussian. Furthermore, as opposed to a 2-D *box* filter,

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

which has a 2-D *Sinc* frequency response, the Gaussian frequency response shows no oscillations, so that we are quite certain about the range of frequencies present in the image after filtering.

3. The Laplacian

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (**see zero crossing edge detectors**). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single graylevel image as input and produces another graylevel image as output.

The zero crossing detector looks for places in the Laplacian of an image where the value of the Laplacian passes through zero — i.e. points where the Laplacian changes sign. Such points often occur at ‘edges’ in images — i.e. points where the intensity of the image changes rapidly, but they also occur at places that are not as easy to associate with edges. It is best to think of the zero crossing detector as some sort of feature detector rather than as a specific edge detector. Zero crossings always lie on closed contours, and so the output from the zero crossing detector is usually a binary image with single pixel thickness lines showing the positions of the zero crossing points.

The Laplacian $L(x,y)$ of an image with pixel intensity values $I(x,y)$ is given by:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

The discrete version of an approximation to the Laplacian is as follows:

$$\nabla^2 f(i, j) = f(i - 1, j) + f(i, j - 1) + f(i + 1, j) + f(i, j + 1) - 4f(i, j)$$

It is derived as follows:

At inter-pixel location $(i + \frac{1}{2}, j)$, the first derivative of the image function along the i axis is approximated by the first difference:

$$\nabla_i f(i + \frac{1}{2}, j) = f(i + 1, j) - f(i, j)$$

Similarly, the first difference at $(i - \frac{1}{2}, j)$ along the i -axis is:

$$\nabla_i f(i - \frac{1}{2}, j) = f(i, j) - f(i - 1, j)$$

The second derivative at (i, j) along the i -axis can be approximated by the first difference of the first difference. computed at positions $(i + \frac{1}{2}, j)$ and $(i - \frac{1}{2}, j)$. that is:

$$\nabla_i^2 f(i, j) = \nabla_i f(i + \frac{1}{2}, j) - \nabla_i f(i - \frac{1}{2}, j) = f(i + 1, j) - 2f(i, j) + f(i - 1, j) \quad (1)$$

Similarly, the second derivative at (i, j) along the j -axis can be approximated by

$$\nabla_j^2 f(i, j) = \nabla_j f(i, j + \frac{1}{2}) - \nabla_j f(i, j - \frac{1}{2}) = f(i, j + 1) - 2f(i, j) + f(i, j - 1) \quad (2)$$

Adding equations (1) and (2) we obtain the desired results.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Another implementation is the following:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

The two commonly used discrete approximations to the Laplacian filter are shown above. (Note, we have defined the second Laplacian using a negative peak because this is more common; however, it is equally valid to use the opposite sign convention.) Using one of these kernels, the Laplacian of an image can be calculated by convolving the 3×3 filter with the image.

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter first of all, and then convolve this hybrid filter with the image to achieve the required result. Doing things this way has two advantages:

Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations. The LoG ('Laplacian of Gaussian') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image. The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$LoG(x, y) == -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}}.$$

$$L[G\{f(x)\}] = L \star (G \star f) = (L \star G) \star f$$

A discrete kernel that approximates this function (for a Gaussian = 1.4) is shown in Figure 3.

Guidelines for Use

The LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (i.e. where the intensity gradient is zero), the LoG response will be zero. In the vicinity of a change in intensity, however, the LoG response will be positive on the darker side, and negative on the lighter side. This means that at a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:

- zero at a long distance from the edge,
- positive just to one side of the edge,
- negative just to the other side of the edge,
- zero at some point in between, on the edge itself.

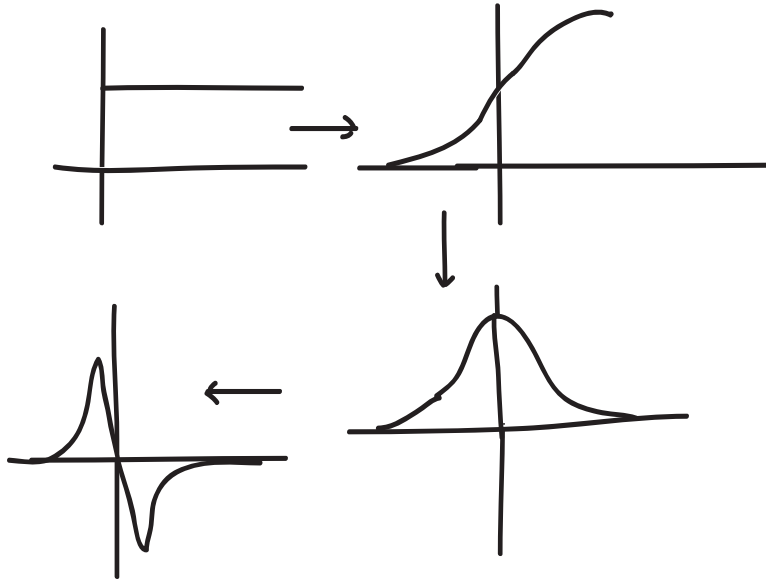


Figure 1:

Figure 1 illustrates the response of the LoG to a step edge.

Expt. 4

In MATLAB do the following:

We can now use the MATLAB built-in capabilities for generating filters with specified parameters. Use *fspecial* to derive a LoG 5×5 filter. Filter an image, say that of *bird*, with your 5×5 LoG filter.

Now do the same using the *edge* function:

```
bw = edge(bird, 'log')
```

which is described below. Comment.

4. Generating Gaussian, Laplacian, Laplacian of Gaussian & other Filters

Just as smoothing is a fundamental operation in image processing, so is the ability to take one or more spatial derivatives of the image. Differentiation, as we know emphasizes discontinuities in the image. Hence high frequency noise in an image is emphasized. The general solution to this problem is to combine the derivative operation with one that suppresses high frequency noise, in short, smoothing in combination with the desired derivative operation.

Matlab provides the *fspecial* command to create some well known filters, including the Gaussian, Laplacian, Log-of-Gaussian filter and other filters. The syntax is as below:

fspecial creates predefined filters.

$h = fspecial(type)$ creates a two-dimensional filter H of the specified type. (*fspecial* returns *h* as a computational molecule, which is the appropriate form to use with *filter2*. *type* is a string having one of these values:

'gaussian' for a Gaussian lowpass filter

'laplacian' for a filter approximating the two-dimensional Laplacian operator

'log' for a Laplacian of Gaussian filter

'average' for an averaging filter
'sobel' for a Sobel horizontal edge-emphasizing filter
'prewitt' for a Prewitt horizontal edge-emphasizing filter
'unsharp' for an unsharp contrast enhancement filter

Depending on TYPE, FSPECIAL can take additional parameters which you can supply. These parameters all have default values.

H = FSPECIAL('gaussian',N,SIGMA) returns a rotationally symmetric Gaussian lowpass filter with standard deviation SIGMA (in pixels). N is a 1-by-2 vector specifying the number of rows and columns in H. (N can also be a scalar, in which case H is N-by-N.) If you do not specify the parameters, FSPECIAL uses the default values of [3 3] for N and 0.5 for SIGMA.

H = FSPECIAL('laplacian',ALPHA) returns a 3-by-3 filter approximating the shape of the two-dimensional Laplacian operator. The parameter ALPHA controls the shape of the Laplacian and must be in the range 0.0 to 1.0. FSPECIAL uses the default value of 0.2 if you do not specify ALPHA.

H = FSPECIAL('log',N,SIGMA) returns a rotationally symmetric Laplacian of Gaussian filter with standard deviation SIGMA (in pixels). N is a 1-by-2 vector specifying the number of rows and columns in H. (N can also be a scalar, in which case H is N-by-N.) If you do not specify the parameters, FSPECIAL uses the default values of [5 5] for N and 0.5 for SIGMA.

H = FSPECIAL('average',N) returns an averaging filter. N is a 1-by-2 vector specifying the number of rows and columns in H. (N can also be a scalar, in which case H is N-by-N.) If you do not specify N, FSPECIAL uses the default value of [3 3].

H = FSPECIAL('sobel') returns this 3-by-3 horizontal edge finding and y-derivative approximation filter:

```
[1 2 1;0 0 0;-1 -2 -1].
```

To find vertical edges, or for x-derivates, use H' (H transpose).

H = FSPECIAL('prewitt') returns this 3-by-3 horizontal edge finding and y-derivative approximation filter:

```
[1 1 1;0 0 0;-1 -1 -1].
```

To find vertical edges, or for x-derivates, use H' .

H = FSPECIAL('unsharp',ALPHA) returns a 3-by-3 unsharp contrast enhancement filter. FSPECIAL creates the unsharp filter from the negative of the Laplacian filter with parameter ALPHA. ALPHA controls the shape of the Laplacian and must be in the range 0.0 to 1.0. FSPECIAL uses the default value of 0.2 if you do not specify ALPHA.

Example

```
I = imread('saturn.tif');  
h = fspecial('unsharp',0.5);  
I2 = filter2(h,I)/255;  
imshow(I), figure, imshow(I2)
```

See also CONV2, EDGE, FILTER2, FSAMP2, FWIND1, FWIND2.

Matlab provides the FILTER2 command to carry out 2-dimensional filtering using a filter (or mask) of your choice. You may supply your own filter, or of course, you may use special filters provided by Matlab using the FSPECIAL command. The syntax is as below:

FILTER2 Two-dimensional digital filter.

`Y = FILTER2(B,X)` filters the data in `X` with the 2-D FIR filter in the matrix `B`. The result, `Y`, is computed using 2-D correlation and is the same size as `X`.

`Y = FILTER2(B,X,'shape')` returns `Y` computed via 2-D correlation with size specified by 'shape':

'same' - (default) returns the central part of the correlation that is the same size as `X`.

'valid' - returns only those parts of the correlation that are computed without the zero-padded edges, `size(Y) < size(X)`.

'full' - returns the full 2-D correlation, `size(Y) > size(X)`.

`FILTER2` uses `CONV2` to do most of the work. 2-D correlation is related to 2-D convolution by a 180 degree rotation of the filter matrix (i.e. reflection about the x and y axes).

See also `FILTER`, `CONV2`.

4. Finding Edges

To find edges, you may use the `FILTER2` command in conjunction with the Sobel, LOG or other filters created using the `FSPECIAL` command. However, Matlab makes your work even simpler because it provides some special edge finding commands for you.

`EDGE` Find edges in intensity image. `BW = EDGE(I,METHOD)` returns a binary image `BW` of the same size as `I`, with 1's where the function finds edges in `I` and 0's elsewhere.

`METHOD` is a string having one of these values:

'sobel' (default) finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient of `I` is maximum.

'prewitt' finds edges using the Prewitt approximation to the derivative. It returns edges at those points where the gradient of `I` is maximum.

'roberts' finds edges using the Roberts approximation to the derivative. It returns edges at those points where the gradient of `I` is maximum.

'log' finds edges by looking for zero crossings after filtering `I` with a Laplacian of Gaussian filter.

'zerocross' finds edges by looking for zero crossings after filtering `I` with a filter you specify.

`BW = EDGE(I,METHOD,THRESH)` specifies the sensitivity threshold. `EDGE` ignores all edges that are not stronger than `THRESH`. If you do not specify `THRESH`, `EDGE` chooses the value automatically.

`BW = EDGE(I,METHOD,THRESH,DIRECTION)` specifies directionality for the 'sobel' and 'prewitt' methods. `DIRECTION` is a string specifying whether to look for 'horizontal' or 'vertical' edges, or 'both' (the default).

`BW = EDGE(I,'log',THRESH,SIGMA)` specifies the 'log' method, using `SIGMA` as the standard deviation of the Laplacian of Gaussian filter. The default `SIGMA` is 2; the size of the filter is `N-by-N`, where `N=CEIL(SIGMA*2)*2+1`. If `THRESH` is empty (`[]`), `EDGE` chooses the sensitivity threshold automatically.

`BW = EDGE(I,'zerocross',THRESH,H)` specifies the 'zerocross' method, using the specified filter `H`. If `THRESH` is empty (`[]`), `EDGE` chooses the sensitivity threshold automatically.

For the 'log' and 'zerocross' methods, if you specify a threshold of 0, the output image has closed contours, because it includes all of the zero crossings in the input image.

`[BW,THRESH] = EDGE(...)` returns the sensitivity threshold.

Class Support ——— I can be of class `uint8` or `double`. `BW` is of class `uint8`.

Example ——— Find the edges of the `alumgrns.tif` image using the Roberts method:

I = imread('alumgrns.tif'); BW = edge(I,'roberts'); imshow(I), figure, imshow(BW)

..... **2.** Look at the 2D filters generated using *fspecial*. Recall the 2D Gaussian,

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

and the Laplacian operator

$$\Delta^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

and for $G(x, y) = -\Delta^2\{f(x, y)\}$ in the discrete domain

$$G(m, n) = [f(m, n) - f(m, n - 1)] - [f(m, n + 1) - f(m, n)] \\ + [f(m, n) - f(m + 1, n)] - [f(m - 1, n) - f(m, n)]$$

and the Laplacian of a Gaussian (*LoG*) often called the “Mexican hat” in which Gaussian smoothing is performed before application of the Laplacian. Here, generate all the filters. You may want to look at the 3-D spatial plots using *mesh(h)* where *h* is the 2D impulse response. Amplitude frequency response comes from *freqz2(h)*. Look at the standard Prewitt, Sobel averaging filters. In particular look at the Gaussian, Laplacian and the LoG filters.

Expt. 5

In MATLAB do the following:

Generate 5×5 Gaussian filter $h[m, n]$. . Use *mesh(h)* to see a *wireframe* surface. Use *freqz2(h)* to determine its frequency response. Comment.

REPORT

1. Comment on Experiment 1.
2. Comment on Experiment 2.
3. Comment on Experiment 3.
4. Comment on Experiment 4.
5. Comment on Experiment 5.

References:

1. Image Restoration Techniques, Chapter 7 from MATLAB for Image Processing. Programs from IPO1.
2. <http://www.qi.tnw.tudelft.nl/Courses/FIP/noframes/fip-Derivati.html>
3. <http://amath.colorado.edu/outreach/demos/hshi/2001Spr/snake/snake.html#eq:fx>
4. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm>
5. http://www.math.hkbu.edu.hk/~cstong/sci3710/filter_tutor.html
6. http://en.wikipedia.org/wiki/Gaussian_function