Elec4622/Elec9722 Laboratory Project 2, 2018 S2

David Taubman

September 7, 2018

1 Introduction

This is the second of three mini-projects to be demonstrated and assessed within the regular scheduled laboratory sessions. This project is due in Week 11. The project is worth a nominal 10 marks, out of the 30 marks available for the laboratory component of the course. However, there are two optional taskss which attract **a lot of potential bonus marks**.

In this project, you will implement and understand the Laplacian of Gaussian (LOG) filter. LOG filtering plays an important role in edge detection for image analysis. From an educational point of view, LOG filters also provide an excellent demonstration of the important connection between discrete and continuous LSI operators – derivative operators in particular. Further, the project will help to familiarize you with scale-space concepts. In the remainder of this introduction, we provide a brief overview of the theory behind LOG filtering.

1.1 The Laplacian Operator in Continuous Space

Let $f(\mathbf{s})$ denote a spatially continuous signal. Throughout this treatment $\mathbf{s} \equiv (s_1, s_2)$ is a two-dimensional spatial location, although everything works for any number of spatial dimensions. In continuous space, the gradient operator $\bigtriangledown$ is well-defined. Specifically, $\bigtriangledown f$ is the vector field

$$\bigtriangledown f(\mathbf{s}) = \begin{pmatrix} \frac{\partial f(\mathbf{s})}{s_1} \\ \frac{\partial f(\mathbf{s})}{s_2} \end{pmatrix}$$

which measures the rate of change of intensity in each of the vertical and horizontal directions.

In the Fourier domain, we know that

$$\widehat{\bigtriangledown f}(\boldsymbol{\omega}) = \begin{pmatrix} j\omega_1 \cdot \hat{f}(\boldsymbol{\omega}) \\ j\omega_2 \cdot \hat{f}(\boldsymbol{\omega}) \end{pmatrix} = \begin{pmatrix} \hat{D}_1(\boldsymbol{\omega}) \\ \hat{D}_2(\boldsymbol{\omega}) \end{pmatrix} \cdot \hat{f}(\boldsymbol{\omega})$$

where $\hat{D}_1(\boldsymbol{\omega}) = j\omega_1$ and $\hat{D}_2(\boldsymbol{\omega}) = j\omega_2$ are the transfer functions of the horizontal and vertical derivative operators, respectively. Of course, differentiation is an LSI operator (i.e., a filter), so that differentiation is nothing other than convolution in the continuous space domain, using PSF's $D_1(\mathbf{s}) = \mathcal{F}^{-1}(j\omega_1)$ and $D_2(\mathbf{s}) = \mathcal{F}^{-1}(j\omega_2)$, respectively.

The Laplacian operator $\bigtriangledown^2$produces the scalar field

$$\bigtriangledown^2 f(\mathbf{s}) = \frac{\partial^2 f(\mathbf{s})}{\partial s_1^2} + \frac{\partial^2 f(\mathbf{s})}{\partial s_2^2}$$

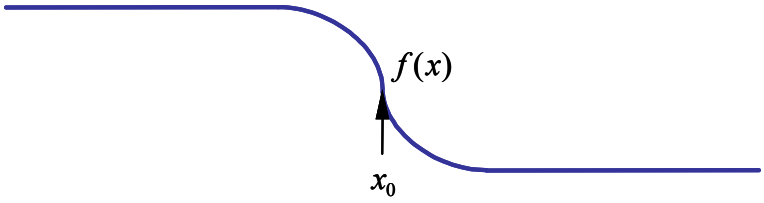

Figure 1: *Horizontal profile of a vertical edge.*

In vector calculus, the Laplacian operator is sometimes written as $\bigtriangledown \cdot \bigtriangledown$. In fact, in the Fourier domain, the Laplacian operator behaves exactly like a dot product. Specifically, we have

$$\begin{aligned}\widehat{\bigtriangledown^2 f}(\boldsymbol{\omega}) &= \underbrace{\begin{pmatrix}\hat{D}_1(\boldsymbol{\omega})\\ \hat{D}_2(\boldsymbol{\omega})\end{pmatrix}^t \cdot \begin{pmatrix}\hat{D}_1(\boldsymbol{\omega})\\ \hat{D}_2(\boldsymbol{\omega})\end{pmatrix}}_{\widehat{\bigtriangledown}^t\widehat{\bigtriangledown}} \cdot \hat{f}(\boldsymbol{\omega}) \\ &= \underbrace{\left(\hat{D}_1^2(\boldsymbol{\omega}) + \hat{D}_2^2(\boldsymbol{\omega})\right)}_{\widehat{\bigtriangledown^2}(\boldsymbol{\omega})} \cdot \hat{f}(\boldsymbol{\omega})\end{aligned}$$

Of course, the Laplacian operator is also LSI, so that the operation can also be viewed as a convolution in the space domain.

One important property of the laplacian operator is that it is isotropic. That is, the operation is invariant to rotation. This is easy to see, since

$$\widehat{\bigtriangledown^2}(\boldsymbol{\omega}) = \hat{D}_1^2(\boldsymbol{\omega}) + \hat{D}_2^2(\boldsymbol{\omega}) = -\left(\omega_1^2 + \omega_2^2\right)$$

is circularly symmetric in the Fourier domain, meaning that the operator's PSF must also be circularly symmetric.

Together, the second derivative and isotropic properties of the Laplacian operator have useful implications for its behaviour in the presence of edges. Suppose the spatially continuous image $f(\mathbf{s})$ contains a vertical edge, so that

$$f(s_1, s_2) = f_{\text{edge}}(x)|_{x=s_2},$$

where f_{edge} is the edge profile illustrated Figure 1. A reasonable definition for the location of the edge is the point $s_2 = x_0$ at which $\frac{\partial^2}{\partial x^2} f_{\text{edge}}$ equals 0. Now

$$\bigtriangledown^2 f(\mathbf{s}) = \frac{\partial^2 f(\mathbf{s})}{\partial s_1^2} + \frac{\partial^2 f(\mathbf{s})}{\partial s_2^2} = \frac{\partial^2}{\partial s_2^2} f_{\text{edge}}(s_2),$$

since all vertical derivatives of a vertical edge are zero. It follows that the edge location coincides with the zero crossing of the Laplacian $\bigtriangledown^2 f(\mathbf{s})$. Although we have considered only horizontal edges, the rotational invariance of the Laplacian operator ensures that the same property should hold for edges of all orientations. That is, the location of an edge in any orientation is well described by the zero crossings of $\bigtriangledown^2 f(\mathbf{s})$.

1.2 The LOG Operator in Continuous Space

One major problem with the Laplacian operator, as it stands, is that derivatives are very sensitive to noise. One way to see this is that the Laplacian operator amplifies high frequency components by $\left(\omega_1^2 + \omega_2^2\right)$. This amplification process strongly affects white noise which has constant power at all frequencies. The noise-free signal itself usually has most of its energy at lower frequencies. Even "high frequency" features such as sharp edges produce power spectra which are located at DC in the parallel direction and decay as $\frac{1}{\omega}$ in the transverse direction.

To reduce the sensitivity to noise, we can think of first applying a low-pass filter to the image $f(\mathbf{s})$. Gaussian filters are a natural candidate since they do not impact the circular symmetry of the Laplacian operator. The LOG operation may then be expressed in the Fourier domain as

$$\hat{g}_\sigma(\boldsymbol{\omega}) = \underbrace{\widehat{\bigtriangledown^2}(\boldsymbol{\omega}) \cdot e^{-\frac{1}{2}\sigma^2 \boldsymbol{\omega}^t \boldsymbol{\omega}}}_{\widehat{\nabla^2_\sigma}(\boldsymbol{\omega})} \cdot \hat{f}(\boldsymbol{\omega})$$

Here, σ^2 is the variance of the Gaussian PSF

$$G_\sigma(\mathbf{s}) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\mathbf{s}^t\mathbf{s}}.$$

Notice that instead of low-pass filtering f and then taking the second derivatives of the result, we can just filter f directly with ∇^2_σ, where

$$\widehat{\nabla^2_\sigma}(\boldsymbol{\omega}) = -\left(\omega_1^2 + \omega_2^2\right) \cdot e^{-\frac{1}{2}\sigma^2\left(\omega_1^2+\omega_2^2\right)} \tag{1}$$

In fact ∇^2_σ is the just $\bigtriangledown^2 G_\sigma$, having PSF

$$\nabla^2_\sigma(\mathbf{s}) = \frac{\partial^2}{\partial s_1^2} G_\sigma(\mathbf{s}) + \frac{\partial^2}{\partial s_2^2} G_\sigma(\mathbf{s}) \tag{2}$$

It follows that the zero crossings of the LOG filtered image, $g_\sigma(\mathbf{s})$, identify the locations of edges in the (Gaussian) low-pass filtered original image. The single parameter σ^2, determines the *scale* of the edge detection process: large values of σ^2 eliminate all but the lowest frequency components so that the edges of larger image features are detected; small values of σ^2 allow more rapid changes in the filtered image, so that the smaller image features can be detected.

1.3 Discrete LOG Filters

As discussed in Chapter 2 of your lecture notes, and also in lectures, Gaussian filtering can be implemented directly by discrete convolution with PSF

$$G_\sigma[\mathbf{n}] = G_\sigma(\mathbf{s})\big|_{\mathbf{s}=\mathbf{n}},$$

so long as the variance is sufficiently large to ensure that G_σ is effectively Nyquist bandlimited – i.e., $\widehat{G_\sigma}(\boldsymbol{\omega}) \approx 0$ for $\boldsymbol{\omega} \notin [-\pi, \pi]^2$. According to equation (1), $\widehat{\nabla^2_\sigma}(\boldsymbol{\omega})$ also decays exponentially, but grows only quadratically, so the same argument shows that the LOG operator can be implemented with high accuracy, by discrete convolution with

$$\nabla^2_\sigma[\mathbf{n}] = \nabla^2_\sigma(\mathbf{s})\big|_{\mathbf{s}=\mathbf{n}},$$

so long as σ^2 is sufficiently large.

You should expand equation (2), both for your practical implementation in this project and also to convince yourself that $\nabla_\sigma^2[\mathbf{n}]$ can be well approximated by an FIR filter. You need to make your own (sensible) judgements concerning the region of support for this FIR filter.

2 Tasks

The tasks presented below build progressively on one another. For demonstration purposes, you may like to create a separate project for each task, within a single workspace. You can do this by using the "File → New → Project" option in Visual C++, replacing the "Create new solution" option with "Add to solution".

Task 1: (3 marks) Write a C/C++ program which can perform LOG filtering on a monochrome or colour BMP image, writing the result to a new BMP image.

- Your program should accept the value of σ as one of its command-line arguments, for testing purposes. This means that the program should dynamically adjust its filter coefficients (and filter region of support) according to the value of σ.
- For this task, you should implement the FIR filter directly, using floating point arithmetic.
- Your program should accept a real-valued scaling factor α as another of its command-line arguments, that allows the LOG filtered values to be scaled prior to writing them to the output image file, so as to facilitate visual inspection.
- Noting that the sample values output by your LOG filter are naturally centred about 0, you should add 128 to all values prior to writing the output file.
- Be prepared to comment on the images you recover using the algorithm.

Task 2: (2 marks) The LOG filter itself is not separable; however, it is a sum of two separable filters. In this task, you modify the program created in Task 1 to exploit this separability for a more efficient implementation.

- Processing should still be performed in floating point arithmetic, for simplicity.
- Use release builds to compare the speed of your two implementations (Task 1 and Task 2). Make an electronic record of your results, but also be prepared to demonstrate the timing during the laboratory demonstration in Week 10.

Task 3: (2 marks) Modify your program to write a sequence of output images, corresponding to progressively larger values of σ, so that you can put this image sequence into a single (MFLEX) video file with the Media Interface tools and play it back. The resulting sequence of images is a type of scale space. Your modified program should accept arguments that identify minimum and maximum values for σ, i.e., $\sigma_{\min}$ and $\sigma_{\max}$, along with the total number of scales N_σ, producing N_σ images whose scales σ are spaced logarithmically between $\sigma_{\min}$ and $\sigma_{\max}$.

Task 4: (3 marks) Modify your program from Task 2 or Task 3 (your choice) to produce a candidate edge map for each scale σ – only one scale if you modify Task 2.

- The output from your program should be an image with only two values: 0 or 255, where 255 corresponds to a potential edge location.
- Notionally, the value 255 should be written to locations that correspond to zero crossing in the underlying spatially continuous LOG-filtered image. However, since you only have a discrete LOG-filtered image you will have to come up with (or research) an appropriate heuristic based on the immediate neighbours of each pixel, to determine whether a zero-crossing is likely to occur nearby.
- It is acceptable for your program to work only with monochrome input images, but to support colour images in this task it is sufficient to perform the same operation on each colour plane, producing a colour edge map where a non-zero value (255) in any colour plane corresponds to a potential zero-crossing in the plane's LOG-filtered sample values.

Task 5: (optional, **up to 2 bonus marks**) Modify your program from Task 2 to work with integer sample values and all integer arithmetic – additions, multiplications and shifts.

Task 6: (optional, **up to 2 bonus marks**) Modify your program from Task 4 to add a simple morphological dilation stage, in which the potential zero-crossings are considered to be the foreground pixels, and the structuring set for the dilation operation is the 3×3 cross

$$A = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

It is important that you be able to quickly show the edge maps you obtain both before and after dilation, so both images should be output by your program.

3 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. There will be some opportunity to do this within the first half of the Week 10 laboratory session. However, you must be prepared to demonstrate your work and be assessed individually at any time in the last half of that session **you cannot expect demonstrators to mark your work only during the final hour of the laboratory session**.

In order to obtain the full marks for any given task, you must have a working program to demonstrate and you must be able to explain how it works and **answer questions very quickly**.

You may feel free to re-use code from the previous laboratory sessions, so long as you understand it. You may also discuss the project with other students in the class, but **your programs should be your own original work**. As with assessment tasks in other subjects, UNSW regards **plagiarism as a serious offense**; for a first offense, the penalty would be loss of all marks for the project.