Elec4622 Laboratory Project 1, T2 2024

David Taubman

June 12, 2024

1 Introduction

This is the first of three mini-projects to be demonstrated and assessed within the regular scheduled laboratory sessions. This project is officially due in your Week 5 laboratory session, but for this first project only, there will be an opportunity to get assessed or re-assessed in Week 6. Note that there is no possibility to further extend submission of your work beyond Week 6, which is already effectively an extended submission date. Moreover, in order to qualify for assessment or re-assessment in Week 6, you must have uploaded all initial work on the project to a Moodle-based submission portal by the end of your laboratory session in Week 5.

The project is worth a nominal 10 marks, out of the 50 marks available for the laboratory component of Elec4622. However, there are optional elements which can attract **a lot of bonus marks**! You will probably not do all the optional elements, although you are welcome to try. In any event, some bonus marks can really help you a lot down the track, so you are recommended to attempt at least some of the options.

In this project you will design and implement Gaussian and related filters to extract edge-related information from an input image at various scales. Gaussian filters have some very nice propertie that have been studied in lectures and at least one tutorial question so far, so you should revise that material from your lecture notes first. You may also find it useful (though not essential) to read the "Image Analysis" chapter of your typeset lecture notes for the course.

To do the project, you should read the following notes very carefully, drawing lots of diagrams on your own to help ensure that you understand what is going on. You will find that details are very important. This is critical for your development as an engineer – engineers know how to go from a general big picture down to the details. Running high level functions, such as one often does in Matlab, does not lead to a deep understanding, since it conceals so many of the details from view. In this project, you are forced to come to grips with the details and that is exactly what employers want from graduates that they employ.

Get ready to put a lot of work into this project, but also to learn a huge amount from it!

2 Gradients, Derivatives and Differences of Gaussians

An important operation in image processing is differentiation. Consider a continuous image $f(\mathbf{s})$, with horizontal and vertical derivatives

$$f_1(\mathbf{s}) = \frac{\partial}{\partial s_1} f(\mathbf{s}) \quad \text{and} \quad f_2(\mathbf{s}) = \frac{\partial}{\partial s_2} f(\mathbf{s})$$

Then the gradient vector for $f(\mathbf{s})$ is written

$$\nabla f(\mathbf{s}) = \begin{pmatrix} f_1(\mathbf{s}) \\ f_2(\mathbf{s}) \end{pmatrix}$$

and takes values at each spatial location.

The magnitude of the gradient vector

$$\|\nabla f(\mathbf{s})\| = \sqrt{\langle \nabla f(\mathbf{s}), \nabla f(\mathbf{s}) \rangle} = \sqrt{f_1^2(\mathbf{s}) + f_2^2(\mathbf{s})}$$

is the maximum rate of change of the image intensity $f(\mathbf{s})$ in any direction, while the orientation of $\nabla f(\mathbf{s})$ is the direction associated with this maximum rate of change. To see this, let $\boldsymbol{\delta}$ denote a small change in spatial position $\mathbf{s}$; then the rate of change of intensity in the direction of $\boldsymbol{\delta}$ is

$$\frac{f(\mathbf{s}+\boldsymbol{\delta}) - f(\mathbf{s})}{\|\boldsymbol{\delta}\|} \approx \frac{\delta_1 \cdot \frac{\partial}{\partial s_1} f(\mathbf{s}) + \delta_2 \cdot \frac{\partial}{\partial s_2} f(\mathbf{s})}{\|\boldsymbol{\delta}\|} = \frac{\langle \nabla f(\mathbf{s}), \boldsymbol{\delta} \rangle}{\|\boldsymbol{\delta}\|} = \|\nabla f(\mathbf{s})\| \cdot \underbrace{\frac{\langle \nabla f(\mathbf{s}), \boldsymbol{\delta} \rangle}{\|\boldsymbol{\delta}\| \cdot \|\nabla f(\mathbf{s})\|}}_{\cos\theta}$$

The last term on the right hand side of the above equation necessarily lies in the range -1 to 1, by the Cauchy-Schwartz inequality, and is in fact just the cosine of the angl θ between $\nabla f(\mathbf{s})$ and $\boldsymbol{\delta}$. So when the direction of movement $\boldsymbol{\delta}$ is the same as the gradient vector $\nabla f(\mathbf{s})$, we observe the largest rate of change in intensity, which is $\|\nabla f(\mathbf{s})\|$.

Differentiation is an LSI operator, so it is equivalent to multiplicatoin in the Fourier domain. In particular, it is easy to show that

$$\hat{f}_1(\boldsymbol{\omega}) = j\omega_1 \hat{f}(\boldsymbol{\omega}) \text{ and } \hat{f}_2(\boldsymbol{\omega}) = j\omega_2 \hat{f}(\boldsymbol{\omega})$$

Therefore, true differentiation of an underlying bandlimited continuous image can be implemented precisely by applying discrete LSI filters $h_1[\mathbf{n}]$ and $h_2[\mathbf{n}]$ to the discrete image $x[\mathbf{n}] = f(\mathbf{s})|_{\mathbf{s}=\mathbf{n}}$, where

$$\hat{h}_1(\boldsymbol{\omega}) = j\omega_1 \text{ and } \hat{h}_1(\boldsymbol{\omega}) = j\omega_2, \text{ over the Nyquist region } \boldsymbol{\omega} \in [-\pi, \pi]^2.$$

This allows us to compute the true gradient vector of the underlying continuous image at each discrete location $\mathbf{n}$, simply by convolving x by h_1 and h_2.

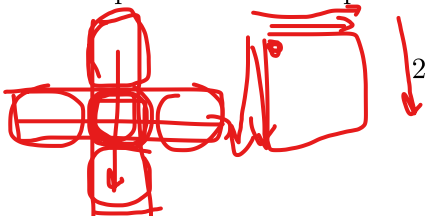
Not suprisingly, the PSFs h_1 and h_2 have infinite support in space, so must be approximated. One simple approach is just to take the inverse DSFT of $\hat{h}_1$ and $\hat{h}_2$ and apply a raised-cosine (Hanning) window to obtain finite support approximations to the derivative operators. In fact, the ideal filters obtained by taking the inverse DSFT of $\hat{h}_1$ and $\hat{h}_2$ are given by

$$h_1[\mathbf{n}] = \delta[n_2] \cdot \left[\frac{\partial}{\partial s} \operatorname{sinc}(s)\right]_{s=n_1} \text{ and } h_2[\mathbf{n}] = \delta[n_1] \cdot \left[\frac{\partial}{\partial s} \operatorname{sinc}(s)\right]_{s=n_2} \tag{1}$$

Very simple approximations to these differentiation filters can be obtained by using

$$h_1^D[n_1] = \frac{1}{2}\delta[n_1+1] - \frac{1}{2}\delta[n_1-1] \text{ and } h_2^D[n_2] = \frac{1}{2}\delta[n_2+1] - \frac{1}{2}\delta[n_2-1] \tag{2}$$

Make sure you understand what these trivial approximations, each with only 2 non-zero coefficients, are actually doing and how to implement them. In particular, use a pen and paper to sketch out what happens



if you perform convolution with these filters, using the output perspective to show that these are just calculating simple "finite differences" on the input image. It is also instructive to work out the Fourier transform of these 1D horizontal and vertical filtering operations; in particular,

$$\hat{h}_1^D(\omega_1) = \sum_{n_1} h_1^D[n]\, e^{-j\omega_1 n_1} = \frac{1}{2}\left(e^{j\omega_1} - e^{-j\omega_1}\right) = j\sin\omega_1,$$

which is approximately equal to $j\omega_1$ for small ω_1.

Notice that the ideal differentiation filters $\hat{h}_1(\boldsymbol{\omega}) = j\omega_1$ and $\hat{h}_1(\boldsymbol{\omega}) = j\omega_2$ are strongly high-pass, with magnitude responses that increase linearly with ω_1 and ω_2, respectively. For this reason, it is common to combine differentiation with low-pass filtering operation, where the amount of low-pass filtering determines the "scale" at which gradients are obtained. In particular, Gaussian low-pass filters are strongly desirable, due to their rotational symmetry and other properties. In this way, we can determine a scale dependent gradient

$$\nabla_\sigma f(\mathbf{s}) = \begin{pmatrix} \frac{\partial}{\partial s_1}(f * G_\sigma)(\mathbf{s}) \\ \frac{\partial}{\partial s_2}(f * G_\sigma)(\mathbf{s}) \end{pmatrix}$$

where G_σ is a circularly symmetric Gaussian low-pass filter with standard deviation σ, which is also understood as the "scale parameter."

In lectures you learned how to perform Gaussian filtering on discrete images while ensuring that the result is equivalent to Gaussian convolution on the underlying continuous image. You will be doing this in this project and combining Gaussian filtering with derivative approximations, in order to discover and display gradient orientation within images.

3 Second derivative operators

Since you now know how to differentiate an image, the approach can readily be extended to a second level of differentiation. Again, write $f(\mathbf{s})$ for the spatially continuous image, with horizontal and vertical second derivatives

$$f_{11}(\mathbf{s}) = \frac{\partial^2}{\partial s_1^2} f(\mathbf{s}) \quad \text{and} \quad f_{22}(\mathbf{s}) = \frac{\partial^2}{\partial s_2^2} f(\mathbf{s})$$

Then the Laplacian operator can be written as

$$f_L(\mathbf{s}) = \nabla^2 f(\mathbf{s}) = \operatorname{div}(\nabla f(\mathbf{s})) = \nabla \cdot \nabla f(\mathbf{s}) = f_{11}(\mathbf{s}) + f_{22}(\mathbf{s})$$

In the Fourier domain, we have

$$\begin{aligned} \hat{f}_{11}(\boldsymbol{\omega}) &= (j\omega_1)^2 \hat{f}(\boldsymbol{\omega}) = -\omega_1^2 \hat{f}(\boldsymbol{\omega}) \\ \hat{f}_{22}(\boldsymbol{\omega}) &= (j\omega_2)^2 \hat{f}(\boldsymbol{\omega}) = -\omega_2^2 \hat{f}(\boldsymbol{\omega}) \\ \hat{f}_L(\boldsymbol{\omega}) &= -\left(\omega_1^2 + \omega_2^2\right) \hat{f}(\boldsymbol{\omega}) = -\|\boldsymbol{\omega}\|^2 \cdot \hat{f}(\boldsymbol{\omega}), \end{aligned}$$

which reveals the fact that the Lapacian operator is a circularly symmetric filter in the continuous domain.

One particularly important property of the Laplacian filtered image is that the precise locations of image edges correspond to zero-crossings of $f_L(\mathbf{s})$ – i.e., the zero-valued contours of the function $f_L(\mathbf{s})$.

As with differention (in fact much more so), second derivative operators are strongly high-pass, so it is again common to combine the Laplacian operator with a Gaussian low-pass filter. This leads to a scale-dependent Laplacian operator, which we can express as

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

4 Boundary and image extension

As you already know, image boundaries cannot simply be ignored. In this project, you will use filters with various regions of support, so from a practical perspective you must ensure that the output image dimensions do not depend on how large the filter support is. Unless otherwise stated, you should use the symmetric boundary extension method.

5 How to go about the tasks

These notes assume that you will demonstrate your work using the laboratory PCs, based on the Windows operating system with Microsoft Visual Studio. You may use your own personal computer to develop code and also to demonstrate your work, but it will be more time efficient for laboratory demonstrators to assess your project on the laboratory PCs, which may result in you getting **more opportunities to correct errors in your solution and be re-assessed for particular tasks**. In general, if you make things easier for the lab demonstrators, they will be able to make things easier for you.

For demonstration purposes, you should 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".

Be sure to arrange for all your executable programs to be placed in a single location (e.g., `c:\elec4642\bin`), which is referenced from the `path` environment variable. Also, please place all the images you are working with into a single directory (e.g., `c:\elec4642\data`). That way, it will be much easier to demonstrate your work in a time effective manner – it will also save you personally a lot of time. In any event, **you might not be marked for your work unless it is organized in this way**.

You are **required to produce hand-drawn sketches** illustrating key features of your design for each task. This is not really any additional work, since it would normally be part of the process of coming up with a design and getting it working. As explained already in laboratories, being able to draw what a diagram of what your program is doing is an essential first step to implementing a solution and a key element in the debugging process. **Do not discard these sketches and diagrams, since lab demonstrators will ask to see them when marking your work, and you are also required to upload them to Moodle as part of an electronic submission process for the project.**

You should rely primarily on **mi_viewer** for viewing images, rather than the Windows previewer. There are lots of reasons for this, but the obvious ones are that you can launch multiple copies to view multiple images at once, and you can zoom in and inspect individual pixel values with ease. Ultimately, it is a great deal faster and less confusing to execute your programs and view your images directly from the DOS prompt, than by using mouse clicks – assuming you can type. Remember that the up and down arrows allow you to scroll through and edit previous commands easily. Remember also that you can use the tab key to expand file names, program names, etc., so you rarely have to type everything in full.

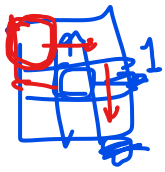
One thing you need to be prepared for is that it may be difficult at first to know whether your program is working correctly. This is a real world dilemma that you need to learn how to address. Just because your program compiles and produces something which looks like an image does not mean it is doing the right thing. So you need to have a good feeling for what the result should look like, based on your understanding of the fundamental concepts, such as shift and rotation invariance.

To debug your program, you may find it useful to process very tiny images with only a few samples, so you can check that the output is as expected; you can use something like `"mi_pipe2 -i image.bmp -o tiny.bmp -form bmp :: crop_n_shuffle -crop 0 0 16 16"` to create a 16x16 image by cropping a much bigger one. You can also read the sample values directly within the "mi_viewer" application by zooming in (z accelerator) and holding the mouse button down. Then you can see exactly what is going on and match up the results with what you expect from the theory.

To verify your work, it may be helpful to collect sequences of command-line statements together into a script that you can run quickly. You can do this with a Windows batch file (i.e., any file with the ".bat" suffix), which can then be executed like any other program from the command-line. This will greatly facilitate the marking for demonstrators, who might insist upon it.

6 Tasks

Task-1: (6 marks) Develop a program that can filter an image either with a Gaussian PSF or with a moving average window.



- Your program should accept either RGB or Greyscale input images in the BMP format, using the image file reading code that you worked with in Lab 1 and Lab 2. Similarly, your program should output images in the BMP format, using the image file writing code that was given to you in those labs. Colour input images should produce colour output images in which each colour channel (red, green and blue) is processed in the same way.
- In addition to image file names, the command line arguments to your program should include a floating-point σ value and an optional switch "-w" – if the switch is found amongst the command line arguments, your program should perform moving average instead of Gaussian filtering.
- For the moving average filter, you need to determine the support of the moving average filter $\mathcal{R}_h = [-H, H]^2$, such that the effective variance of the moving average PSF (interpreted as a probability density funtion) is as close as possible to σ^2, which is the variance of the Gaussian PSF (interpreted as a probability density function).
  - To be clear, your moving average filter $h[\mathbf{n}]$ should have a constant value for $-H \leq n_1, n_2 \leq H$, being zero outside this region, with DC gain 1; then its variance can be modelled as

$$\mathrm{var}(h) = \sum_{\mathbf{n}} h[n_1, n_2] \cdot (n_1^2 + n_2^2)$$

    You can think of the variance as the weighted squared "spread" produced by the PSF, where weights come from the PSF coefficients. Apart from the subtle impact of discretization, your Gaussian PSF also has a weighted squared "spread" of σ^2
- For the Gaussian filter, you are required to choose a sensible region of support, again based on the value of σ, that is a command-line argument to your program.

- Your program must support any non-zero value of σ, up to a maximum of 100 and you should print out a single line of text (just use the "printf" function) that describes the region of support of whichever filter is being implemented. Note that we do not recommend using $\sigma < 1$ in general, but it is useful to have the ability to try out very small values of σ so that you can see what happens – this is especially useful when the extensions of Task 2 are introduced.
- Both the moving average and Gaussian PSFs implemented by your program should have zero phase – i.e., they must be symmetric about $\mathbf{n} = \mathbf{0}$ – and they must be normalized to have a DC gain of exactly 1.

Bonus-1: (2 mark) To score this bonus mark, you should provide an implementation of the moving average filter that uses at most 2 additions, 2 subtractions and one normalizing multiplication per sample value, and which produces essentially the same result as a direct implementation – this can be done within a separate program if you like.

Task-2: (2 marks) Extend your program from Task 1 to find the (approximate) gradient vector field $\nabla_\sigma x[\mathbf{n}]$ from input image $x[\mathbf{n}]$, performing all internal computations in floating-point arithmetic. Use the simple derivative approximation method from equation (2) in combination with the low-pass filtering of Task 1, which determines the scale parameter σ.

- Your program should accept the same σ argument and "-w" option as in Task 1, plus one additional real-valued positive scaling factor α.
- The BMP output image $y[\mathbf{n}]$ should be the scaled magnitude of the gradient vector – i.e.,

$$y[\mathbf{n}] = \alpha \cdot \|\nabla_\sigma x[\mathbf{n}]\| \tag{3}$$

- In implementing equation (3), you need to be careful to use clipping to avoid numerical wrap-around in the output image, in case $y[\mathbf{n}]$ becomes larger than 255.
- As in Task-1, for colour input images, you are expected to process each of the red, green and blue colour planes in the same way.

Bonus-2: (2 marks) An alternate way to combine Gaussian filtering with differentiation is to recognize that

$$\frac{\partial}{\partial s_1}(f * G_\sigma)(\mathbf{s}) = (f * G_{\sigma,1})(\mathbf{s}),$$

where

$$G_{\sigma,1}(\mathbf{s}) = \frac{\partial}{\partial s_1} G_\sigma(\mathbf{s}) = \underbrace{\frac{\partial}{\partial s_1}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}s_1^2/\sigma^2}\right)}_{\text{horizontal derivative of Gaussian}} \cdot \underbrace{\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}s_2^2/\sigma^2}\right)}_{\text{vertical Gaussian}}$$

and

$$G_{\sigma,2}(\mathbf{s}) = \frac{\partial}{\partial s_2} G_\sigma(\mathbf{s}) = \underbrace{\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}s_1^2/\sigma^2}\right)}_{\text{horizontal Gaussian}} \cdot \underbrace{\frac{\partial}{\partial s_2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}s_2^2/\sigma^2}\right)}_{\text{vertical derivative of Gaussian}}$$

Notice that $G_{\sigma,1}$ and $G_{\sigma,2}$ are both separable continuous filters, and that their Fourier transforms are

$$\hat{G}_{\sigma,1}(\boldsymbol{\omega}) = j\omega_1 \cdot e^{-\frac{1}{2}|\boldsymbol{\omega}|^2\sigma^2} \text{ and } \hat{G}_{\sigma,1}(\boldsymbol{\omega}) = j\omega_2 \cdot e^{-\frac{1}{2}|\boldsymbol{\omega}|^2\sigma^2},$$

which are approximately Nyquist bandlimited so long as $\sigma \gtrsim 1$. Therefore, we can implement these so-called "Derivative of Gaussian" (DOG) filters in the discrete domain, simply by using filters whose PSFs are obtained by sampling the above expressions.

For this (optional) bonus task, create a separate program that is identical to the one from Task-2, except that it uses DOG filters instead of separate Gaussian low-pass filters and finite difference derivative approximations, choosing a suitable region of support for the DOG filters so that negligible approximation error is introduced into your solution.

- You should compare the output from your DOG-based solution with the output from Task-2, using Gaussian low-pass filters, focussing on small values of σ (i.e., close to 1) and be prepared to comment on any differences you observe.
- Besides visual observations themselves, you should be prepared to discuss any thoughts you have on how you might expect the use of the DOG approach to be beneficial for an algorithm that detects image edges.

Task-3: (2 marks) Extend your program from Task 1 to find the (approximate) Laplacian image $\nabla_\sigma^2 x[\mathbf{n}]$ from input image $x[\mathbf{n}]$, performing all internal computations in floating-point arithmetic. Use the simple derivative approximation method from equation (2) twice, in combination with the low-pass filtering of Task 1, which determines the scale parameter σ.

- Your program should accept the same 3 command-line arguments as in Task 2
- The BMP output image $y[\mathbf{n}]$ should be a level-shifted, scaled version of the Laplacian image – i.e.,

$$y[\mathbf{n}] = \alpha \cdot \nabla_\sigma^2 x[\mathbf{n}] + 128 \tag{4}$$

- In implementing equation (4), you need to be careful to use clipping to avoid numerical wrap-around in the output image, in case $y[\mathbf{n}]$ becomes larger than 255 or smaller than 0.
- As in Task-1, for colour input images, you are expected to process each of the red, green and blue colour planes in the same way.

Bonus-3a: (2 marks) An alternate way to combine Gaussian filtering with the Laplacian operator is to recognize that since both are LSI operators (i.e., filters), the order of operations can be changed without altering the result, so we get

$$\nabla_\sigma^2 f(\mathbf{s}) = \left(f * \left(\nabla^2 G_\sigma\right)\right)(\mathbf{s})$$

That is, the Lapacian-filtered image at scale σ can be obtained by filtering $f(\mathbf{s})$ with the Laplacian-of-Gaussian filter, whose PSF is

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

It should be obvious that the Fourier transform of this Laplacian-of-Gaussian (LOG) filter is

$$\hat{L}_\sigma(\boldsymbol{\omega}) = -\|\boldsymbol{\omega}\|^2 \hat{G}_\sigma(\boldsymbol{\omega}) = -\|\boldsymbol{\omega}\|^2 e^{-\frac{1}{2}\|\boldsymbol{\omega}\|^2 \sigma^2},$$

which of course is still circularly symmetric. Importantly, the operation is effectively Nyquist bandlimited, so long as $\sigma \gtrsim 1$, due to the strong decay of the exponential term with rising $\|\boldsymbol{\omega}\|$. Thus, the true continuous Laplacian-of-Gaussian operator can be implemented accurately in the discrete domain by convolving a source image with the discrete LOG PSF

$$L_\sigma[\mathbf{n}] = L_\sigma(\mathbf{s})|_{\mathbf{s}=\mathbf{n}}$$

Create a separate program that is identical to the one from Task-3, except that it uses the discrete LOG filter above, instead of separate Gaussian low-pass filters and finite difference derivative approximations, choosing a suitable region of support for the LOG filter so that negligible approximation error is introduced into your solution.

- You should compare the output from your LOG-based solution with the output from Task-3, using Gaussian low-pass filters, focussing on small values of σ (i.e., close to 1) and be prepared to comment on any differences you observe.

Bonus-3b: (2 marks) In this (optional) bonus task you create a program that presents both the magnitude of the gradient image and the laplacian-of-Gaussian filtered image together, within thecolour planes of its output image.

- Your program should accept the same command-line arguments as the ones in Task-2 and Task-3, in addition to a second real-valued positive scaling factor β.
- The BMP colour output image should have red, green and blue colour planes as follows:

$$\begin{aligned} y_B[\mathbf{n}] &= \alpha \cdot \|\nabla_\sigma x[\mathbf{n}]\| \\ y_G[\mathbf{n}] &= \beta \cdot \nabla^2_\sigma x[\mathbf{n}] + 128 \\ y_B[\mathbf{n}] &= x[\mathbf{n}] \end{aligned}$$

- As always, you need to be careful to use clipping to avoid numerical wrap-around in the output image.
- For colour input images, you should process only the green input colour plane.
- If you have successfully completed both the Bonus-2 and Bonus-3 tasks, it is preferable for you to use the DOG and LOG filters developed in those bonus tasks to form the $\nabla_\sigma x[\mathbf{n}]$ and $\nabla^2_\sigma x[\mathbf{n}]$ intermediate images for this bonus task. Otherwise, use the methods from the non-bonus Task-2 and non-bonus Task-3.
- Experimenting with different values of σ, α and β, you should be prepared to comment on the relationship between the gradient magnitude, the Laplacian image, and the original image edges. For this, you should find it very helpful to inspect the output image using the "mi_viewer" utility, which allows you to quickly turn individual colour planes on or off using accelerator key combinations "ctrl-r", "ctrl-g" and "ctrl-b."

7 Assessment

You should not rely upon implementing this project within the scheduled laboratory sessions. Instead, you must be prepared to demonstrate and explain your work in the Week 5 (or perhaps Week 6) laboratory. You

should make sure each task can be run simply from the command-line. Spend some effort organizing your patterns and test images ahead of time, perhaps using separate sub-directories. I suggest creating small batch files (files ending in ".bat" that contain commands you would normally type on the command line) that display the test image and output using "mi_viewer". Doing this will save you and the demonstrator a lot of time, both before and during the lab, since it is easy to compare different tasks using such batch files without having to remember all the conditions you used.

Note carefully: Lab demonstrators will need to see the hand-drawn sketches you have produced as a critical part of the design process, showing how you have organized your image data, how you have computed parameters and dimensions, extended boundaries, and other such things.

7.1 Team work, plagiarism and copying

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 otherwise be your own original work – this is **not a group project**!

You are **required to submit your code** via an Assignment item on the course's Moodle page, **within 2 hours of the end of your scheduled laboratory session**, following the instructions provided there. Your code may be cross-checked for plagiarism, so make sure that you do not copy any other student's actual implementation, or base your solution on one that you obtain from another student.

7.2 Option for re-assessment in Week 6

As noted in the introduction, although this project is due in Week 5, you will be given the opportunity for re-assessment in Week 6. This does not mean that you can simply defer your assessment until Week 6 or that you can think of the entire project as being due in Week 6. **You will only be granted the opportunity to be re-assessed in Week 6 if you are able to present at least a partially working solution to some of the tasks in Week 5.**

7.3 Managing the limited resource of demonstrator time

During your labs, demonstrators will have a major responsibility of marking your project. This is time consuming, and so you cannot expect to be marked only in the last hour of a lab session. To maximize your opportunity to be marked and perhaps to be re-assessed in Week 6, you should come prepared to the lab session in Week 5 with many elements of your project completed or at least partially working, so that you can ask a demonstrator to mark or look over your solution as early as possible within the lab session.