Digital Image Processing Laboratory: Image Filtering

January 26, 2021

Introduction

In this laboratory, you will filter full color images using both FIR and IIR filters, and you also will learn to use Python to read, show, and write your images.

In this laboratory assignment, the filter algorithms should be implemented in the C programming language, unless otherwise stated. Low level programming languages are necessary for implementing many complex image processing operations, so one objective of this laboratory is to give you some experience in using C to solve image processing problems. Simple example programs are provided, so you should be able to write the required programs even if you have very little previous experience with C.

When processing images on a computer, there are usually special cases that must be properly handled. In particular, all filters will be implemented using **free** boundary conditions along edges, and pixel values will be clipped to a range of $[0, 255]$.

- Free boundary conditions - In order to understand a free boundary condition, consider an $M \times N$ image denoted by $img(i, j)$ where $0 \leq i < M$ and $0 \leq j < N$. Some filtering operations require that you index outside of this valid range. Pixels indexed outside this range should be considered equal to zero.

- Clipping - Image processing operations will sometimes cause a pixel to exceed the value 255 or go below the value 0. In these cases, you should clip the pixel's value before displaying or exporting the image.

$$y(m,n) = \begin{cases} 0 & \text{if } x(m,n) < 0 \\ 255 & \text{if } x(m,n) > 255 \\ x(m,n) & \text{if } 0 \leq x(m,n) \leq 255 \end{cases}$$

Questions or comments concerning this laboratory should be directed to Prof. Charles A. Bouman, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907; (765) 494-0340; bouman@ecn.purdue.edu

1 C Programing

An important goal of these laboratories is for you to learn good C programming skills. You do not need much prior experience in C programming, because the laboratories will lead you through the use of the language with examples and program templates. However, to learn good C programming skills, you should strictly follow the following set of rules at all times.

- **Use strict ANSI C constructs** - All C compilers allow for enforcement of strict ANSI C. You should use these options at all times. With the gcc compiler under the Linux operating system, the compiler flags are
  gcc -ansi -Wall -pedantic
  These flag settings will insure that you receive warnings for any poor programming constructs in your code.

- **Never use global variables** - Global variables should never be used. There are exceptions to this rule, but you will never face a situation where they are truly needed. Global variable will lead to "spaghetti code", that can not be understood or debugged.

- **Fix all compiler warnings** - Any warning from the compiler means that you have an error in you program or are using poor programming style. Either problem is serious and should be corrected. Typically, warnings are generated when variables are not properly retyped. The only exception to this rule is when you dynamically allocate memory. In this case, you may receive a warning about the byte location.

- **Use prototypes for all subroutines** - Every subroutine you write should have a prototype that defines its argument types. This prototype should generally be contained in a single header file with a name such as *defs.h*.

2 Displaying and Exporting Images in Python

Most of the lab exercises in this class will require you to produce one or more output images. These will need to be incorporated into your pdf report. Since many exercises will use Python, we describe here some basic I/O and display commands in the Python environment.

- **Setup Conda Environment** - It is recommended that you create a conda environment for this course using the following procedure:

  1. Install Anaconda through its website.
  2. An ASCII file named "environment.yml" is provided that can be used to configure your Anaconda environment. The file, shown below, installs the packages called "numpy", "matplotlib", and "pillow" that will be used in the class.

     ```
     name: ECE637
     dependencies:
       - numpy
       - matplotlib
       - pillow
     ```

  3. Create the conda environment by running the following command with the file "environment.yml" in your working directory.

     ```
     conda env create -f environment.yml
     ```

  4. Activate conda environment with the following command.

     ```
     conda activate ECE637
     ```

- **Reading Images** - You can read an image file, *img.tif*, into python using pillow using the following commands

  ```
  from PIL import Image
  im = Image.open('img.tif')
  ```

  This will produce an image matrix x of data type *uint8*.

- **Displaying Images** - You can display the image array x with the following command,

  ```
  im.show()
  ```

- **Transfer Image to numpy array** - You can transfer an image object to a numpy array using the following commands,

  ```
  import numpy as np
  x = np.array(im)
  ```

- **Writing Images** - When producing a lab report document, you should strive to present the best representation of your output images. Therefore it is best to export your images to a *lossless* file format, such as TIFF or BMP, which can then be imported into your lab report document. You can write the image array x to a file using the *save* function:

  ```
  img_out = Image.fromarray(x)
  img_out.save('img_out.tif')
  ```

  Note that if x is of type *uint8*, the imwrite function assumes a dynamic range of [0,255], and will clip any values outside that range. If x is of type *double*, then imwrite assumes a dynamic range of [0,1], and will linearly scale to the range [0,255], clipping values outside that range, before writing out the image to a file. To convert the image to type uint8 before writing, you can use

  ```
  img_out = Image.fromarray(x.astype(np.uint8))
  img_out.save('img_out.tif')
  ```

In most of the following exercises, your C routines will output TIFF files which can be imported into your lab document. Your lab document should be submitted in pdf (portable document format), but it can be prepared using an application such as Latex or MS Word. When you prepare your lab document, you may need to convert the image to an intermediate format such as jpeg . If you do this, try to set the image quality parameters to the best possible setting in order to preserve image detail and quality.

For your own viewing, you may display the output images using Python as described above, or using any other standard image viewing software. Regardless, you are encouraged to experiment with the effect of each of the above commands using one of the images provided on the lab web page.

3 FIR Low Pass Filter

In this problem, you will analyze and implement a simple low pass filter given by the 9×9 point spread function

$$h(m,n) = \begin{cases} 1/81 & \text{for } |m| \leq 4 \text{ and } |n| \leq 4 \\ 0 & \text{otherwise} \end{cases} .$$

1. Calculate an analytical expression for $H(e^{j\mu}, e^{j\nu})$, the DSFT of $h(m,n)$, and use Python's matplotlib to plot the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$. Make sure to label the axes properly and plot on over the region $[-\pi, \pi] \times [-\pi, \pi]$.

2. Download the bundle *C-code.zip* containing the C source code for reading and writing tagged image file format (TIFF) images. Compile the application *Example* using an ANSI C compiler.

3. Run the program *Example* on the image *img03.tif* by using the command
   ```
   Example img03.tif
   ```
   This will produce the gray scale image *green.tif*, which is derived from the green component of *img03.tif*.

4. Modify the program *Example* so that it filters the red, green and blue components of *img03.tif* with the filter $h(m,n)$ and generates a full color output image.

   Warning: Make sure to rename the directory containing your modified program so that it can not be overwritten with the original *C-code* program directory.

   Note: In the example code provided, noise is added to the image before writing it out. This is for illustration only–you are **not** asked to add noise to your image in this exercise.

5. Include the filtered image in your report.

Section 3 Report:
Hand in:

1. A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$.

2. A plot of $|H(e^{j\mu}, e^{j\nu})|$.

3. The color image in *img03.tif*.

4. The filtered color image.

5. A listing of your C code.

4 FIR Sharpening Filter

In this problem, you will analyze the effect of a sharpening filter known as an unsharp mask. The terminology comes from the fact that an unsharp mask filter removes the unsharp (low frequency) component of the image, and therefore produces an image with a sharper appearance.

Let $h(m, n)$ be a low pass filter. For our purposes use

$$h(m,n) = \begin{cases} 1/25 & \text{for } |m| \leq 2 \text{ and } |n| \leq 2 \\ 0 & \text{otherwise} \end{cases} .$$

The unsharp mask filter is then given by

$$g(m,n) = \delta(m,n) + \lambda(\delta(m,n) - h(m,n))$$

where λ is a constant greater than zero.

1. Calculate an analytical expression for $H(e^{j\mu}, e^{j\nu})$, the DSFT of $h(m, n)$, and use Python's matplotlib to plot the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$. Make sure to label the axes properly and plot on over the region $[-\pi, \pi] \times [-\pi, \pi]$.
2. Calculate an analytical expression for $G(e^{j\mu}, e^{j\nu})$, the DSFT of $g(m, n)$. Use Python to plot $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$. Make sure to label the axes properly and plot it over the region $[-\pi, \pi] \times [-\pi, \pi]$.
3. Modify the program *Example* so that it filters the red, green and blue components of *img03.tif* with the filter $g(m, n)$ and generates a full color output image. Write your modified code so that it accepts a command line argument specifying the value of λ.
4. Apply your sharpening filter to *imgblur.tif* for $\lambda = 0.2, 0.8, 1.5$. Use Python, or some other image display tool, to view and compare each result to the original image.
5. In your report, include the output image for $\lambda = 1.5$, and also include your C code used for the filtering.

Section 4 Report:
Hand in:

1. A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$.
2. A derivation of the analytical expression for $G(e^{j\mu}, e^{j\nu})$.
3. A plot of $|H(e^{j\mu}, e^{j\nu})|$.
4. A plot of $|G(e^{j\mu}, e^{j\nu})|$ for $\lambda = 1.5$.
5. The input color image *imgblur.tif*.
6. The output sharpened color image for $\lambda = 1.5$.
7. A listing of your C code.

5 IIR Filter

In this problem, you will analyze the effect of an IIR filter specified by a 2-D difference equation. Let $h(m,n)$ be the impulse response of an IIR filter with corresponding difference equation

$$y(m,n) = 0.01x(m,n) + 0.9(y(m-1,n) + y(m,n-1)) - 0.81y(m-1,n-1)$$

where $x(m,n)$ is the input and $y(m,n)$ is the output.

1. Calculate an analytical expression for $H(e^{j\mu}, e^{j\nu})$, the DSFT of $h(m,n)$, and use Python's matplotlib to plot the magnitude of the frequency response $|H(e^{j\mu}, e^{j\nu})|$. Make sure to label the axes properly and plot over the region $[-\pi, \pi] \times [-\pi, \pi]$.
2. Use Python to compute the point spread function of the filter, $h(m,n)$, by applying the above difference equation to a 256×256 image of the form $x(m,n) = \delta(m-127, n-127)$. Export the result to a TIFF file using the following scaling,

```
im_save = Image.fromarray((255*100*h).astype(np.uint8))
im_save.save('h_out.tif')
```

3. Modify the program *Example* so that it filters the red, green and blue components of *img03.tif* with the filter $h(m,n)$ and generates a full color output image.
4. Include the output image in your report.

Section 5 Report:
Hand in:

1. A derivation of the analytical expression for $H(e^{j\mu}, e^{j\nu})$.
2. A plot of $|H(e^{j\mu}, e^{j\nu})|$.
3. An image of the point spread function.
4. The filtered output color image.
5. A listing of your C code.