SD 575 Image Processing
Fall 2016

Lab 0: Introduction
Lab Date: Sep. 14, 2016
No Lab Report, Not Graded

The Image Processing Labs

The lab component of SD 575 is designed to complement the material being covered in the lecture portion of the course. Labs are intended to be done in groups of two or three. The labs will illustrate and reinforce the concepts covered in the lectures by providing an opportunity to observe the effects of the algorithms. The labs will allow you to experiment with a variety of image processing algorithms, and to design and implement your own.

The lab instructions assume that you are using Matlab. The image and signal processing toolboxes of Matlab have made it powerful scientific software for digital image processing. Matlab is particularly well-suited for prototyping and experimenting with algorithms with minimal overhead.

Students who are new to Matlab can find many tutorials and introductions online:

http://www.saw.uwaterloo.ca/matlab/
UW IST's page on Matlab has details on the availability of Matlab, examples, and links to tutorials.

http://www.mathworks.com/access/helpdesk/help/helpdesk.html
The official documentation for Matlab contains detailed examples illustrating the use of practically every function.

Matlab

The goal of this lab is to provide an introduction using the Image Processing Toolbox of Matlab.

Matlab supports most common image formats, including JPEG, GIF, BMP, PNG, and TIFF. Typing "`help imread`" at the Matlab prompt will give a complete list.

To find out what commands are available in the Image Processing Toolbox, type "`help images`". Matlab commands are generally quite well documented, and typing "`help` [*commandname*]" will give you detailed information on what a function does and how to use it. You will also be able to find an often more detailed explanation online by searching "Matlab [*commandname*]" on Google.

Some Useful Matlab Image Processing Commands

`imread`	Load an image
`imwrite`	Save an image
`imshow`	Display an image
`imfinfo`	Display image information
`mean2`	Compute global mean value of an image
`std2`	Compute the global standard deviation of an image
`improfile`	Select a line along which to take an intensity profile
`imhist`	Compute and display the image histogram
`fspecial`	Generate a predefined filter mask
`filter2`	Perform a 2-D convolution
`rgb2gray`	Convert RGB to grayscale

Useful Matlab Information

Numeric variables in Matlab are all implicitly treated as matrices, and most operators and functions are designed to behave appropriately for matrices. Matlab is highly optimized for operations on vectors and matrices, and users encouraged to design their programs to exploit this: a "vectorized" function is generally at least an order of magnitude faster than a function that manually iterates through the matrix.

Matrix vs. Element-by-Element Operations

When working with matrices, there are two ways that operations can be performed. They can be performed element-by-element, or using the entire matrix. Operators `*`, `/`, and `^` are defined as the matrix operation. So, for example, if A and B are matrices

```
C = A*B;    D = A^2;
```

would multiply matrix A with B and store the result in C, and multiply matrix A by itself and store the result in D. (Note that Matlab will give an error if the dimensions of A and B are not suited to these operations.)

Adding a "**.**" modifier in front of these operators causes them to be performed on respective, individual elements. Thus, the command

```
C = A.*B;  D = A.^2;
```

would instead assign to C a matrix where each element is the product of the respective element in A multiplied by the respective element of B. D is a matrix equal in size to A where each individual element has been squared. By using the modifier, the operation is performed on each element of the matrix individually rather than on the matrix as a global structure. Notice that when both operands are scalars, both forms yield the same result.

One side note: if you do not terminate a command with a semicolon (`;`), Matlab will output the value generated by the command.

uint8 vs. double Format

Typically, images are stored in a format where the pixel values span the integer range [0, 255]. Images of this format are loaded by Matlab and stored in a type known internally as *uint8*, or an 8-bit unsigned integer. Some operations in Matlab may not be defined for this data type and will need to be converted to the *double* floating-point type which spans the range [0, 1]. This can easily be done by dividing the values by 255, or by using the `im2double` function.

Loading and Saving Images in Matlab

The previously mentioned `imread` and `imwrite` commands are used to load and save images in Matlab.

To load an image, use the `imread` command as follows.

```
i = imread('filename');
```

The image information is now stored in the matrix `i`. The file name must be enclosed in the single quotation marks (`'`). If the file is not found, it may not be in the Matlab path. Either use the path tool (Set Path) found under the File menu, specify the entire path of the file, or change directory in the Matlab prompt to the location of the file. For example, to load an image called `vacation.jpg` from `D:\Pictures`, you would use the command

```
i = imread('D:\Pictures\vacation.jpg');
```

To save an image, use the imwrite command as follows.

```
imwrite(i,'filename');
```

or

```
imwrite(i,'filename','fmt');
```

If the string `fmt` is not specified, `imwrite` attempts to determine the format based on the filename extension you use.

To incorporate images into your reports, Microsoft Word is able to import images from many formats, including JPEG, TIFF, GIF, PNG, and BMP.

You are also able to export to file from the figure window when you display your images. With this method, you can create Encapsulated Postscript (eps) versions of your images for use with mark-up languages such as LaTeX or for use in Word.

Sample Session

Command	Result
`A = imread('cameraman.tif');`	Read in the cameraman image
`imshow(A)`	Display the image
`imfinfo('cameraman.tif')`	Display the information associated with that image

`Figure`
`Agray = rgb2gray(A);`
`imhist(Agray)`
	Open a new figure window
	Display that image's histogram in the new window

`P = impixel(A)`	Examine the pixel values in the image
`m = mean2(A)` `s = std2(A)`	Find global statistics (mean, and standard deviation) on all the pixels in the image

`B = imadjust(A,[],[],gamma);`
with gamma < 1
or gamma > 1
`figure, imshow(B)`
	Adjust image intensity
	Stretch dark regions
	Stretch light regions
	View the result in a new window

`B = histeq(A);` `imshow(B)`	Equalize the histogram
`h = fspecial('average')` `C = filter2(h,A)./255;`	Generate a local average mask Smooth the image with the local average mask, and normalize the result from [0,255] to [0,1]
`D = (double(A)./255) - C;` `imshow(D + 0.5)`	Find edges using (image - smoothed image) NOTE: the image when loaded is stored in 'uint8' format, which has range [0, 255]. Most operations must be done using 'double' format which has range [0, 1] so the image must be rescaled.
`E = (double(A)./255) + D;` `imshow(E)`	Enhance the edges
`h = fspecial('sobel')` `F = filter2(h,A)./255;`	Find horizontal edges with sobel operator
`h = fspecial('sobel')'` `G = filter2(h,A)./255;`	Find vertical edges with sobel operator
`I = sqrt(f.^2 + g.^2);` `imshow(I)`	Create an edge image with the gradient magnitude

Programming in Matlab

Matlab provides an easy means for users to create their own programs or functions. User defined scripts and functions are provided through what are called ".m" files. These are a collection of Matlab commands organized in a particular means. All valid Matlab commands can be used within functions, including other user defined functions, provided they are all in the same directory, or are locatable within the Matlab path. To set the Matlab path in Windows, select the "Set Path" option under the "File" menu and add your own directory to the path.

The ".m" files can be created in your favourite text editor (vi, emacs, Notepad++, NOT Microsoft Word or WordPerfect) or using the Matlab editor. To use the Matlab editor, simply type "edit" once Matlab is running.

An ".m" file is normally organized as follows:

function [*return_value_1, return_value_2, …*] = *function_name*(*input_value_1, input_value_2, …*)

Matlab code

The word function must appear as the first uncommented word in the file. If there is only one return value, it does not need to be contained in "[]". The ".m" file must have the same name as the *function_name* or it will not work. The return values and input values can be scalars, matrices, images, etc. There is no need to use a return statement, as in C or C++, and multiple values can be returned. The returned value is simply the last value stored in that variable name when the function completes execution.

Comments can be added using the "%" character. Anything after the "%" will be considered a comment and ignored.

Matlab supports all programming constructs including decisions (if - else if - else) and loops (for, while). For more details, see the help files or printed documentation.

For example, the following function will create the edge image with the gradient magnitude using the Sobel edge detector (edge detection will soon be discussed in the course lectures).

```
function Y = sobel_grad(X)
% sobel_grad creates and returns the magnitude
% image of the sobel edge detector

Y = sqrt(filter2(fspecial('sobel'),X).^2 + ...
      filter2(fspecial('sobel')',X).^2);
```

The "**…**" indicates that the command is continued on the next line.

Note that you can also create .m files that do not begin with a function declaration. These files are treated as scripts, and operate on the user's workspace variables.