# ENSC 180 – Introduction to Engineering Analysis Tools

# Assignment #5: Basic Image Processing

### Objective

The purpose of this assignment is to introduce basic image processing in MATLAB. You will learn how to load and save images, perform blurring, sharpening, edge detection, and resizing. These are the basic operations used in many image processing applications.

### Background

MATLAB represents images as matrices. A grayscale ("black and white") image is usually represented as a single matrix whose elements are called pixels. Typically, a pixel value is an integer number between 0 and 255, so it can be represented as a single Byte (8 bits). For color images, the most common representation consists of three matrices, sometimes called image "planes" or "channels," which represent Red, Green, and Blue (RGB) values of a pixel.

To get started with the assignment, read the help files of the following MATLAB functions: imread, imwrite, double and uint8. These functions are used for reading an image from a file, writing an image to a file, and for converting between 8-bit integer representation and floating-point representation. Also read the help file of imshow, which is used to display an image.

### Assignment details

The image, *Baboon,* is referenced in this assignment, in TIFF format. This image can be downloaded from the publicly accessible USC-SIPI image database http://sipi.usc.edu/database/database.php?volume=misc

**Task 1:** In this task, you will learn how to access image planes of a color image, extend an image, and implement simple image blurring. First, load the *Baboon* image using

X = imread('baboon.tiff');

Then convert it from uint8 to double using

X = double(X);

Next, read the help file of the MATLAB function wextend. This function can be used to extend an image horizontally and vertically, which is what we will need in order to implement blurring.

Our first goal is to implement simple blurring as pixel averaging within a certain window. We will start with a small window of size 3×3. Each pixel value will be replaced by an average of 9 pixel values: itself, and its 8 nearest neighbors (Top, Bottom, Left, Right, Top-Right, Bottom-Right, Bottom-Left, Top-Left). Such operation is often represented by a 3×3 matrix (in this context, sometimes called *kernel* or *filter*) shown below, where '1/9' in each entry means that each of the neighbors gets equal weight (1/9) when averaging is performed.

|  |  |  |
| --- | --- | --- |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

So, we need to scan all three image planes, and for each pixel, perform this averaging and replace the pixel value by the average value. Note that pixels at the boundary of the image don't have all 9 neighbors - for example, pixels on the left boundary don't have neighbors to the left. That is why we need to extend the image, so that all pixels from the original image get 9 neighbors. For a 3×3 kernel, 1-pixel extension in each direction is sufficient.

Use the MATLAB function wextend to symmetrically extend the image by 1 pixel on each side. The command is as follows:

Y = wextend('2D', 'sym', X, [1,1]);

Make sure you understand all the parameters. Next, we need to obtain the number of rows (r), columns (c) and planes (p) of the image Y:

[r,c,p] = size(Y);

Then, for each plane, and each row between 2 and r-1 and each column between 2 and c-1 (this corresponds to all the rows and columns of the original image prior to extension), replace each pixel by the average of itself and its eight neighbors. To get you started, here is what the sequence of nested for loops could look like:

for k = 1:p

for i = 2:r-1

for j = 2:c-1

You need to fill in the code to accomplish averaging.

Once this is accomplished, Y will be an extended blurred version of image X. Now extract the central portion of Y, which excludes the 1 pixel boundary that was added on all sides, and store the result in X1 - this is the blurred version of the original image. Using imshow, display both images X and Xb. Note that you need to convert images from double back to uint8 in order to display them using imshow. Do you see any difference?

**Task 2:** Repeat the procedure from Task 1, but for the larger kernel size of 5×5. Call the resulting image X2. Keep in mind that extension needs to be adjusted to account for larger kernel size. What other changes need to be made? Do you see any difference in the resulting images?

**Task 3:** I this task you will learn how image blurring can be accomplished more efficiently using MATLAB's built-in functions. Please read the help files of fspecial and imfilter. The first of these functions is used to construct the kernel and the second one is used to perform blurring (or, more generally, filtering).

First, construct a 3×3 averaging kernel using

H = fspecial('average',[3 3]);

How does H compare with the 3×3 matrix shown in Task 1?

Then, perform image blurring using

X3\_1 = imfilter(X, H, 'symmetric', 'same');

Make sure you understand what each argument of imfilter means. Note that there is no need to extend the image this time, imfilter does that automatically based on the kernel size. How does the resulting image compare with that obtained in Task 1?

Repeat fspecial and imfilter for the 5×5 averaging kernel, store the resulting image in X3\_2, and compare the result with the image obtained in Task 2.

Finally, using fspecial, create a Gaussian kernel of size 5×5 with . Then perform blurring using imfilter, store the resulting image in X3\_3 and compare it with the image from Task 2, as well as X3\_2.

**Task 4:** Images are often blurred in practice unintentionally. For example, some parts of the image may be out of focus, or the camera may move during exposition. It would be useful to have a way to sharpen such images to improve their visual quality. In this task you will implement a simple sharpening method called *unsharp masking*. You can read more about it at

http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/unsharpmask/

Unsharp masking works as follows. Let be the blurred image. First, we will blur this image further using a Gaussian kernel, to produce an even more blurred image . Then we will subtract from with some scaling, as follows

With proper choice of the scaling parameter , the resulting image will appear sharper than the original blurred image .

For this task, you should implement unsharp masking as follows. Take image X2 from Task 2 to be the blurred image . Construct the Gaussian filter of size 5×5 with . Create image Xg by blurring image X2 with the Gaussian kernel. Then create the final, sharpened image using the equation above, and call it X4. Test this procedure for various values of between 0.51 and 0.7. Does the resulting image look sharper than X2? Which value of gives the best result visually?