An introductory tutorial on MATLAB in Image Processing
ECE 178 (1-2pm Discussion batch, TA: Srivatsan Pallavaram)

I. GETTING STARTED

MATLAB is a data-analysis and visualization tool widely used by electrical engineers and stands for "Mathematics Laboratory." The most important difference between MATLAB and C (or C++) is that functions in MATLAB are specifically written with a focus on large array-based operations in mind. Image processing is heavy on the memory usage as well as the run-time of programs.

You would need the Image Processing toolbox installed in your MATLAB to work with the commands that are listed later in this worksheet. You can check if the Image Processing toolbox is installed using the *ver* command. Using "help images" one can see all the commands that are supported in this toolbox. If you find the help document scrolling past very fast you can switch the 'more' command using >> more on or for a more formal help use >> doc 'command' A MATLAB *function* is a keyword that accepts various parameters, and produces output of some sort. At times in this course, we may need to write our own functions (and this is very easy as you will see). A *command* is a particular usage of a function. *Variables* are elements used to store values. If you are using a windowed version of MATLAB, you may find a Workspace item in the View menu. This lists all your currently defined variables, their data types, sizes etc.

Exercise 1: Explore the *whos* and *who* commands that are used to list the variables used in the workspace.

At this point, I would like to stress that learning/ understanding MATLAB is impossible by just reading this handout. The exercises in this handout are motivated primarily by this reason. Using standard MATLAB routines "efficiently" Could mean the difference between a program that runs for a day and one that runs in an hour. Use of *for* loops is known to increase the run-time by a significant amount.

Exercise 2: This exercise describes how 'slow' the *for* command can be. We can measure the time that elapses between the executions of a command using the *tic toc timer* of MATLAB. tic starts a stopwatch timer and toc stops it and prints out the elapsed time in seconds. Execute the following code and compare the elapsed time differences between the two.
>> tic; for i = 1:10^6, sin(i);end; toc;
>> tic; i=1:10^6; sin(i); toc; Understand how the *for* loop can slow down the process by quite a bit.

II. BASIC MATRIX OPERATIONS IN MATLAB

The standard data type of MATLAB operations is the matrix. Images, of course, are matrices whose elements are the gray values (or possibly the RGB values) of its pixels. I am assuming that most of you are aware of the basics of MATLAB. More information on these commands can be obtained using the *help* and *lookfor* commands. A standard 2 × 3 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \qquad (1)$$

is defined as >> A = [a11 a12 a13; a21 a22 a23;]; Matrix elements can be obtained using the standard row, column indexing scheme >> a(2,3) The *reshape* function produces a matrix with elements taken column by column from the given matrix. MATLAB also allows matrix elements to be obtained using a single number; this number being the position when the matrix is written as a single column. For e.g. a matrix with r rows and c columns, element $a(i, j)$ corresponds to $a(i + r \cdot (j - 1))$. We would rarely be using this notation. Addition of matrices is done as >> c = a+b; The *inv*, *det* and *'* commands do the inverse, determinant and transpose operations respectively.

Exercise 3: Matrices can be flipped up or down using *flipud*, left or right using *fliplr* and rotated by 90 degrees by *rot90*. Explore them.

To obtain a row of values, or a block of values, we use the colon (:) operator. Pointwise multiplication (Hadamard product of two matrices) is done with >> c = a.*b; We also have dot division and dot powers. The command >> a.^2 produces a matrix, each element of which is the square of corresponding elements of *a*.

Exercise 4: Using the colon and dot operators alone, generate the first 15 cubes.

Sometimes, we would need to generate some special matrices and random matrices. An

identity matrix of order *N* is generated using the command *eye*. An all zero matrix is generated using *zeros*. An all-1 matrix is generated using *ones*. *Randn(N)* generates an $N \times N$ matrix with entries from a standard (real) normal distribution whereas *rand(N)* generates a random matrix with entries from a uniform distribution. We can generate a complex random matrix with Gaussian entries either using the *randn* function or the *wgn* function. The *floor* operator produces the integer part of a result.

Exercise 5: Unlike other programming languages that require a nested loop to generate a matrix A with $A(i, j) = i + j - 1$, understand this MATLAB code that generates A.

```
>> rows = (1:10)'*ones(1,10);
>> cols = ones(10,1)*(1:10);
>> A = rows + cols - 1;
```

The construction of rows and cols can be done automatically with the *meshgrid* function.

```
>> [cols, rows] = meshgrid(1:10,1:10);
```

Exercise 6: Write a function issquare that will determine whether a given integer is a square number. (Hint: Use the floor function.)

Exercise 7: Learn the different features of the *plot* and *axis* commands. I am assuming that you are familiar with the basic MATLAB operations and this would just be a quick review of it.

III. STORING IMAGES IN MATLAB

Images are usually heavy on memory. More pixels, (number of elements needed to store the pictures) implies more memory. The pictures that you usually download from the web are either in the jpeg/jpg (Joint Photographic Experts Group), gif (Graphics Interchange Format), tif/tiff (Tagged Image File Format), bmp (Windows Bitmap). Other not-so-common formats include png, hdf, pcx, xwd, cur and ico. There are different types of images that can be stored in MATLAB. Binary images store an image as a 3-D matrix with two dimensions representing the image dimensions and the third storing values only in 0 and 1 (thus binary). Since each pixel needs only 1 bit for storage, this image needs very less memory to store. You also have the gray scale images (black and white) which is similar to the binary image, but with more than a bit to store the pixel value. This memory is a byte for an image of the *uint8* class and much more for the *double* class. The *double* class stores the pixel value in a continuum between 0 and 1, whereas the *uint8* class quantizes the pixel value between 0 and 255. 0 corresponds to a completely black pixel and 255 (respectively 1) corresponds to a completely white pixel. The *uint8* class occupies approximately 8 times less memory than the *double* class. Also some image processing commands work only on the *double* class and some only on the *uint8* class. It is very easy to convert between these two classes. There are also other classes like *uint16* which are more common with tiff/png/bmp images, but *uint8* is the default for the more commonly used jpg images. An irreversible conversion from the gray-scaled image to the binary image is possible (check later for conversion between different formats). Usually image files on the web are color images. Storing color images can be done in two ways, the RGB format or the indexed format. The RGB format stores the red, green and blue (primary color contents) of the corresponding pixel. MATLAB then internally combines these corresponding pixel values to paint the pixel with the appropriate color when we display the image. It is a simple exercise to note that MATLAB can represent 256^3 colors in the RGB format. MATLAB by default stores (check later for the 'imread' command) an image in the RGB format unless you tell it otherwise. An RGB image is also called a 24-bit image. An image in the RGB format is a 3-D matrix (size1 × size2 × 3). The red, green and blue pixel values are stored always according to the *uint8* class. The other storage format for color images is the indexed format. Not all the 16 million color combinations possible with the RGB format are needed for representing an image. Thus sometimes it is more efficient to use a reduced set of colors to represent the image. From this arises the notion of "colormap." Unlike the RGB format, which stores the pixel value in the third dimension, the indexed format stores a reference number. We then use another matrix called the colormap matrix as a lookup table to see what color the reference number actually corresponds to. In reality this scheme is much more memory efficient than the RGB format. You can check if a color image on the web is an indexed image or an RGB image using the *imfinfo* command. The pixel values are always stored in the *double* class. The image read operations (check the following section) also read the colormap.

IV. I/O OPERATIONS WITH AN IMAGE

Reading a jpg/gif image is done using the 'imread' command. If the file is stored as filename.fileextension, >> I = imread('filename','fileextension'); reads the

image as a matrix I. Be sure to put the ';' or you will be staring at a set of numbers running endlessly on the screen. If you store the image as
>> I = imread('filename.fileextension');
you let MATLAB decide by what it sees, the file format etc. Indexed images can be identified using the *imfinfo* command. For indexed images, one also has to read the colormap as follows.
>>[I,cmap]=imread('filename.fileextension');
After you have done the necessary image processing, storing an image back in the jpg (or similar) formats is done with the 'imwrite' command. The standard procedure is to
>>imwrite(I,'filename.fileextension');
Check for corresponding commands with indexed images. You may sometimes want to save the image matrix so that you can work with it later. This is done with the *load* and *save* commands (which are not image processing toolbox specific). *save I* saves the image matrix I in the file 'I.mat' and *load I* loads the saved contents to the variable 'I'. There are three different file formats in MATLAB, the m files, the mat files and the mex files. Mat files are stored files and you would not be dealing with the mex files in this course. You can check the variables in your workspace with the *whos* command and list all the files with the *ls* command. Displaying an image can be done with the *imshow* or *imagesc* commands. We only describe the syntax for indexed images (For rest, check the help files.). >> imshow(I,cmap); The images are displayed in the Figure window.

Exercise 8: Pick a gray-scale image, say cameraman.tif or any other file that you can get hold of, and using the imwrite function write it to files of types JPEG, BMG and GIF. What are the sizes of those files?

V. COMMANDS THAT YOU WILL BE USING LATER

The following commands maybe useful later in this course.

- *pixval on/off*: You can see the values of the pixels using this command in the figure window.
- *impixel(·, ·, ·)*: You can obtain the values of the pixels using this command.
- *brighten(·)*: This command brightens an image if the parameter inside is positive. Check the help files for syntax.
- *isgray(·)*: Answers if the image is a gray-scaled image.
- *imcrop(·)*: Produces a copy of the image which can then be cropped. Should store the image then in a different file for post-processing.
- *image(·)*: Displays image in a Figure window.
- *imfinfo(·)*: Returns information about the image file stored (in the jpg/gif/other formats).
- *plot(·,·,·)*: Plots the vectors against each other. Sizes should match.
- *zoom on/off*: Zooms into or out of the figure.
- *colormap*: You can get to see the colormap associated with an indexed figure using this command. Consult the help file for syntax.
- *fft2(·)*: This does the 2-D Fourier transform of the image as a matrix.
- *dct2(·)*: The DCT-II of the input matrix is done with this command.
- *conv2(A,B)*: This does the 2-D convolution of A by B. Note that the resultant convolved matrix is of increased size. More information can be obtained using the MATLAB help files or a standard MATLAB book or consulting the Image processing document pages in the Mathworks site.

VI. COMMON PITFALLS

- Be sure to use .* instead of * when you're multiplying two images together point-by-point. Otherwise, Matlab will do a matrix multiplication of the two, which will take forever and result in total nonsense.
- Be sure not to forget the semicolon at the end of a command. Otherwise, you may sit for a while watching all the pixel values from the resulting image scroll by on the screen!
- Do not use for loops unless absolutely necessarily. The use of for loops will make your programs take **much** longer.