**CSSE463 Image Recognition Lab 1: Intro to MATLAB and Image Processing**

**Outcomes:**

1. Familiarity with MATLAB IDE.

2. Familiarity with arrays.

3. Ability to read, save, and view images.

3. Create a personal QuickReference guide to MATLAB that you can use in the future.

**Grading checklist of deliverables:**

1. Answers to the questions in red:
2. \_\_\_\_/5 pts: 3D arrays
3. \_\_\_\_/5 pts Matlab for the first column of a 2D-array.
4. \_\_\_\_/5 pts Matlab for upper-right-hand corner of a 7x10 array.
5. \_\_\_\_/5 pts: replace values in range with -1 using *find*
6. \_\_\_\_/10 pts: paragraph about Matlab subscripts
7. \_\_\_\_/10 pts: paragraph about converting color to grayscale images.
8. Images you were told to save:
9. \_\_\_\_/5 pts Orig img
10. \_\_\_\_/5 pts Grayscale img
11. Your Matlab QuickReference guide
12. \_\_\_\_/10 pts
13. Following directions:
14. \_\_\_\_/8 pts: All answers and images included inline in a single document

labeled as *yourname*- Lab 1.docx or *yourname*-Lab1.pdf.

1. \_\_\_\_/2 pts: submitted to the dropbox for Lab 1.
2. \_\_\_\_/10 pts: writing, formatting, and professionalism

**Total: \_\_\_\_/80 pts**

**Step-by-step Directions:**

**I. Install MATLAB**

*Note: you need to be on campus or connected to the VPN to use MATLAB, since it requires a license key.*

Install MATLAB if you don’t have it. It is available on \\tibia\public\Course Software\MATLAB\MATLAB\_R2008a (grab the newest, but we don’t technically need it). You’ll want to run the Full Install.bat batch file, since it handles entering the license key as well. It takes awhile since it’s a huge install, so plan accordingly. (If you don’t want to do a full install because disk space is an issue, you will need at least the following toolboxes: **image processing toolbox Version 5.0.2 or later, image acquisition, neural nets, optimization, and statistics**.) You can check the version of the toolbox by typing “help images” at the MATLAB prompt once it’s installed. **Note**: if you have an old version of MATLAB and it doesn’t work, you may need to do a re-install: Uninstall the old version of MATLAB (and blow away the MATLAB folder with any remaining files that are left when the uninstaller completes), reboot, and do then reinstall as above.

**II. Familiarity with MATLAB IDE (5 min)**

Start MATLAB on your laptop. Experiment with the environment for a few minutes to get comfortable with it. MATLAB includes a command-line interpreter, a help system, a scripting system, and a debugger (and more), but for now, we want to get familiar with the interpreter.

Note: you can change the startup directory by following the comments below (taken from MATLAB help, searching for “default directory”).

**III. Change the Startup Folder**

You should start MATLAB in a folder other than the default, like somewhere in My Documents, or wherever you are storing your work for this course.

*Windows only*: Right-click the MATLAB shortcut icon and select Properties. The Properties dialog box for MATLAB.exe opens to the Shortcut page. Enter the new startup directory in the Start in field and click OK. It will update the next time you start MATLAB using that shortcut icon. You can make multiple shortcuts to start MATLAB, each with its own startup folder, and with each startup folder having different startup options.

*Alternative that works on all platforms*: In the current startup folder (e.g., C:\Program Files\MATLAB\R2012a\bin), create a startup.m file which includes the cd function to change to the new directory (like cd /Users/boutell/Documents/Courses/CSSE463/MyMatlabProjects).

**IV. Prepare to create a Quick Reference Guide**

Create an empty text or other file. As you work through this tutorial, copy into it anything that you want to remember for later. For example, it could be a list of functions, with sample usage as you deem appropriate for yourself.

For the questions you are to answer and images you are to save, please create a Word document named *yourname*- Lab 1.docx. Please save all images inline in the text, so they are easy to find. (Requiring Word may seem picky, but it’s the easiest way for me to have access to your original images as well, since I can just unzip the Word doc to grab them.) [Alternatively, if you don’t want to use Word, you could create a pdf using another tool, but still have images inline. You’ll also need to submit images separately.]

**V. 1D Arrays (5 min)**

*I will use >> to mean something you should type.*

*Everything after the % character is a comment.*

>>A = [3 6 8 9 10 ]; %to create a 1x5 matrix called A.

>>A (with no semicolon) to display the matrix. Omitting the semicolon on any expression displays the value of the expression.

>>B = 2\*A %to create a matrix B, with values twice as big as A’s

>>C = A + B %to find the sum.

>> D = 1:10 %What does this do? Then guess what E=1:2:10 does.

Note how easily MATLAB does array and matrix calculations.

Type A(1), A(1:3), and A([1,3,5]) to extract parts of array A. These can be saved as other variables, e.g., X = A(4). Note three things:

**MATLAB uses 1-based subscripts.** You have to forget your Java and C training in this course.

**MATLAB has flexible subscripting.** You can even use another array as a subscript (can you come up with an example to use here?)

**MATLAB uses (), not [] to contain subscripts. It uses [] to initialize arrays.**

**VI. 2D Arrays (5 min)**

>>A = [1 2 3; 4 5 6; 7 8 9] %to create a 3x3 matrix called A.

>>B = A %to create a copy of A,

>>C = A + B %to sum them

>>A(2,3) %**What does this do?**

>>A(1:2, 2) %**What does this do?**

>>A(3,:) %**What does this do?**

**VII. 3D arrays (5 min)**

3D arrays are a pain to type by hand. We use multiple statements in each of which one 2D “slice” of the array is assigned.

>> clear A; % to remove its contents

>> A(:,:,1) = [1 2; 3 4]

>> A(:,:,2) = [5 6; 7 8]

>> A(:,:,3) = [9 10; 11 12]

>> B = A(:,:,2)

**Why will dealing with 3D arrays be important in this class?**

We will be dealing with pixels in an image a lot. These pixels can be represented as a 3D array. In color it will be x, y, color, and in greyscale it will be x, y, intensity. The color itself will be another array representing the RGB value.

**Write a MATLAB expression to refer to the first column of a 2D-array.**

A(:, 1)

**Write a MATLAB expression to refer to the upper-right-hand pixel of a 7 x 10 array. Note we always give dimensions of 2D arrays as (#rows) x (#cols).**

A(1, cols)

**VIII. MATLAB syntax for old and new things**

MATLAB can operate on whole arrays at a time, so you should avoid loops whenever you can. You can even apply conditions to a whole image in one step. Say we wanted to replace all negative values in an array with 0.

A = [5 -4 7 -2 10];

idx = find(A < 0) % What’s the significance of the values in idx?

A(idx) = 0 % Wow!

Note: We can combine these:

A(find(A < 0)) = 0

In MATLAB, we use &, |, ~ for **and**, **or**, and **not,** respectively.

**Write a MATLAB expression to replace all values in a matrix A between 10 and 20 with the value -1.**

A(find(A > 10 & A < 20)) = -1

You can create a matrix of all zeroes:

B = zeros(4,2) % This will be helpful in the future

**In your own words, summarize these MATLAB concepts: find, zeros, slicing, flexible subscripting.**

**Find – returns indices of a matrix where the given condition is true for the value at an indice**

**zeros – creates a matrix of all zeroes with a given row and column size**

**slicing – extracting part of a matrix or array**

**flexible subscripting – you can give it a number of things depending on what you would like to get, a single subscript for a single value, colons for entire rows or columns or ranges, an array of the elements to get [1 3 4] or the first, third, and fourth.**

Of course, MATLAB lets you use decisions and loops when needed:

if x > 10

x = x + 1; % no ++ or even += in MATLAB

else

x = x – 1;

end

j=1;

for i = 1:10

j = j\*i; % What’d we just compute?

end

However, nested loops are slow in MATLAB. *Use find and matrix ops whenever possible when operating on images*.

**IX. Loading and saving images (5 min)**

There are two ways to move around in the file system.

1. Use the browser (the top-left window).

2. Use the command-line interpreter. It is just like Linux: **pwd** gives you the present working directory, **ls** displays the files and folders there, **cd** moves you around (for example, cd ../ to move up one level).

Create an image folder in your current directory.

Move a photo of your choosing into the image directory. One with a few bright colors will be more fun to work with. Please **cite the source** (if online, the URL where you found it; if a personal photo, just say so) in your report.

http://wallimgs.com/landscape-photography-trees.html/landscape-photography-trees

Load the image by using, for example:

>> img = imread(‘image\foo.jpg’);

**Note you can get help for any MATLAB function by typing help <function> or using the help browser (F1).**

Filenames are enclosed in **single** quotes.

Folders are specified using **back**slashes.

>>img %to see the pixel values.

>> whos %to see its size. What are the 3 dimensions of your image?

>>imshow(img) %to see the image.

**Save the image using**

>> imwrite(img, ‘<filename>.png’);

Tiff is uncompressed, and so would lead to big files. Jpg is lossy compression, so we get artifacts. Png is a happy medium: its compression is less aggressive than jpg, but is *lossless.*

**XI Exploring Images (5 min)**

There are two tools for exploring images and pixel values: imtool and imshow. Try them both:

>>imtool(img)

Experiment with some of the menu options here. Try Tools/PixelRegion.

Notice the info displayed at the bottom of the image

>>imshow(img) And play around with the settings.

**XII Manipulating Image Colors (15 min)**

We’ll now experiment with looking at single bands of the image.

>> red = img(:,:,1);

>> imtool(red);

Now isolate the blue and green bands of the image as well. **Which looks the most like how you’d expect the corresponding grayscale (“black-and-white”) photo to look?**

A quick and dirty approximation of grayscale is found by averaging the R, G, and B values.

Note: you need to convert from **byte** (the default for images) to **double** to “do arithmetic” on the images.

**So don’t use** >> avg = (red + green +blue)/3;

Instead, **use** >> avg = uint8((double(red) + double(green) + double(blue))/3);

**Notes about uint8:**

bar = uint8(foo); %converts foo to a byte for display or to save.

%This also “clips” the values to [0,255] (mapping anything too

% big, like 372, down to 255, and anything negative to 0)

**% If you don’t account for this, you can get in big trouble!**

>>gray = rgb2gray(img) creates a 2D (grayscale) image from the color image. Compare it visually to each of the single bands and the average grayscale conversion formula above. rgb2gray uses a more realistic approximation of grayscale is given by the formula

gray = 0.3R + 0.59G + 0.11B.

(Note: this formula gives a more realistic approximation, but for some images, it may yield results that are less visually appealing than one of the other formulas. For example, consider an image with a lot of a single color and little of the others.)

>> imtool(gray)

In imtool, try Tools/Adjust contrast and Tools/Colormap

**Save any grayscale image you created.**

**Summarize what you have learned about converting color to grayscale images.**

**I learned that a picture can be broken into a red, green, and blue band of it. The averaging of these three bands seems to be what matlab uses for its rgb2gray function.**

**Now go back to the grading checklist and make sure you have done everything listed there, then submit it following those directions.**