ECEN 642, Fall 2019

Texas A&M University

Electrical and Computer Engineering Department

Dr. Raffaella Righetti

Due: 09/13/2019 (before class)

Assignment #1

Gonzalez and Woods (4th edition), projects: 2.2-2.7 and Gonzalez and Woods (3rd edition), project 02-04.

For the assignments, you will need to use MATLAB. You can access it on campus through the open access lab (OAL) or remotely through the virtual open access lab (VOAL). The link below will guide you into configuring the Horizon client for VOAL remote connection step by step on your PC or mac:

https://tamu.service-now.com/tamu-selfservice/knowledge_detail.do?sysparm_document_key=kb_knowledge,6f7e0c6adbce5f84778ff5961d96199f#

After you connect to the server, you will see the MATLAB icon. If you don't, you can connect to your VOAL desktop (VOAL icon) and start MATLAB from there.

For projects from the 4th edition, photo copies of the project statements are provided. For projects from the 3rd edition, you can access them via the link:
http://www.imageprocessingplace.com/DIP-3E/dip3e_student_projects.htm#02-04

has one or more dark blobs with a diameter of 0.8 mm or greater, as measured on the scale of the screen. The manufacturing manager believes that if she can find a way to fully automate the process, profits will increase by 50%, and success in this project will aid her climb up the corporate ladder. After extensive investigation, the manager decides that the way to solve the problem is to view each inspection screen with a CCD TV camera and feed the output of the camera into an image processing system capable of detecting the blobs, measuring their diameter, and activating the accept/reject button previously operated by a human inspector. She is able to find a suitable system, provided that the smallest defect occupies an area of at least 2×2 pixels in the digital image. The manager hires you to help her specify the camera and lens system to satisfy this requirement, using off-the-shelf components. Available off-the-shelf lenses have focal lengths that are integer multiples of 25 mm or 35 mm, up to 200 mm. Available cameras yield image sizes of 512×512, 1024×1024, or 2048×2048 pixels. The *individual* imaging elements in these cameras are squares measuring 8×8 μm, and the spaces between imaging elements are 2 μm. For this application, the cameras cost much more than the lenses, so you should use the lowest-resolution camera possible, consistent with a suitable lens. As a consultant, you have to provide a written recommendation, showing in reasonable detail the analysis that led to your choice of components. Use the imaging geometry suggested in Problem 2.8.

Projects

MATLAB solutions to the projects marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

2.1* Become familiar with the software you will be using for image processing. In particular, you should be able to:

(a) Read, display, and write (save) images.

(b) Write, edit, and execute code.

(c) Become familiar with how to save and retrieve your work.

2.2* Extracting pixel values from grayscale images.

(a) Write a function **v = pixVal4e(f,r,c)**, where **f** is a grayscale image and **r** and **c** are scalars corresponding respectively to a row and column number in **f**. Output **v** is the pixel value **f(r,c)**.

(b) Test your function by reading the image **girl.tif** and obtaining the pixel values at the origin and at the middle of the image. Display the values on the screen.

(c) Write a function **[r,c,v] = cursorValues4e(f)** that displays image **f**, displays a mouse-controlled cursor over it and, when the mouse left button is clicked, outputs the row/column coordinates **(r, c)** and the value **v** of the pixel at those coordinates. The function should close the display of **f**. The file of image **f** and any previously opened displays should not be closed by this function. (*Hint:* Use the function from (a) to obtain the pixel value and, if you are using MATLAB, you can use function **ginput** to display the cursor and get its coordinates.)

(d) Read the image **girl.tif** and use **cursorValues4e** to obtain the coordinates and value of the center of the girl's right pupil. Display the results.

2.3 Extracting intensity scan lines from grayscale images.

(a) Write a function, **s = scanLine4e(f,l,loc)**, where **f** is a grayscale image, **l** is an integer, and **loc** is a character string with value **'row'** or **'col'** indicating that **l** is either a row or column number in image **f**. The output, **s**, is a vector containing the pixel values along the specified row or column in **f** (see Fig. 2.16 for an example).

(b) Read the image **rose1024.tif**, get a horizontal intensity scan line in the middle of the image, and plot the scan line.

2.4* Intensity scaling of grayscale images.

(a) Write a function **g = intScaling4e(f,mode,type)** where **f** is a grayscale or RGB image. If

the value of **mode** is **'default'** (or **mode** is not included in the argument) the intensities of **f** should be scaled to the range $[0, 1]$, without affecting the spread of the values. For example, if **f** is an 8-bit image with intensities in the range $[64, 200]$, **g** will have values in the range $[0.2510, 0.7843]$ using this option. If any value of **f** exceeds 255, then **g** should be **f** divided by its maximum value. If **f** has negative values, force the **mode** to **'full'**. If **f** has values in the range $[0, 1]$ then **f** should be passed unchanged. If the value of **mode** is **'full'**, **f** is scaled to the full $[0, 1]$ range using Eqs. (2-31) and (2-32). Parameter **type** is a character string with value **'integer'** or **'floating'** (the default), to set the values of **g** to integers in the range $[0,255]$, or to floating point numbers in the range $[0, 1]$. To use **type**, **mode** must be included in the argument.

(b) Read and display the image **spillway-dark.tif**.

(c) Scale the image to the floating point range $[0, 1]$ using the **'full'** option in function **intScaling4e**. Display the result.

(d) Scale the image to the **'full'** and **'integer'** range, i.e., $[0, 255]$, and display the result.

(e) Discuss the reason(s) for any differences between the three images.

2.5 Rectangular binary masks.

(a)* Write a function **m = mask4e(M,N,rUL,cUL,rLR, cLR)** for creating a binary mask of size $M \times N$ with 1's in the rectangular region defined by upper-left row and column coordinates **(rUL,cUL)**, and lower-right coordinates **(rLR,cLR)**, respectively. The rest of the $M \times N$ mask should be 0's. Your function should contain a check to make sure the specified region of 1's does not exceed the dimensions of the $M \times N$ region.

(b)* Read the image **rose1024.tif** and generate a square mask whose sides are one-half the size of the image in both directions. The mask should be centered on the image (see Fig. 2.19).

(c)* Confirm that the number of 1-valued elements in the mask is correct.

(d) Apply your mask to image **rose1024.tif** and display your result.

2.6 Arithmetic operations between grayscale images.

(a)* Write a function **g = imArithmetic4e(f1, f2, op)** for performing arithmetic operations (as defined by **op**) on grayscale images **f1** and **f2** (note that either, or both, of these images could be a scalar: i.e., an image of size 1×1). Parameter **op** is a character string that indicates the following arithmetic operations between **f1** and **f2**: **'add'**, **'subtract'** (**f1** − **f2**), **'multiply'**, and **'divide'** (**f1**/**f2**). These are elementwise operations, as defined in Section 2.6. Output image **g** should be floating point. (*Hint:* Convert the input images to floating point to perform the arithmetic operations.)

(b) Read the image **girl.tif** and use function **mask** and the **'multiply'** option in **imArithmetic4e** to highlight the girl's face from the top of the forehead to the bottom of the chin, and from the left to the right ear. This is an example of how to define a *region of interest* (ROI) in an image. This would be one of the first steps in an application such as face recognition. (*Hint:* Use project function **cursorValues4e** to get the coordinates needed to define the mask image.)

2.7 Mask mode radiography. Use function **intScaling4e** in default mode to display your results.

(a)* As explained in Example 2.6, *mask mode radiography* is based on subtracting a mask image from a live image (note that this is a fundamentally different way of using a mask image from the approach in part (b) of project 2.6). Read the images **angiography-live-image.tif** and **angiography-mask-image.tif** and subtract the latter from the former to duplicate the result in Fig. 2.32(c).

(b) Reverse the order of subtraction in (a) and explain the reason for the difference in the result.

2.8 Affine transformations. If interpolation is needed, use nearest-neighbor interpolation. See Table 2.3 for the definition of the constants and the directions of coordinate axes.

(a)* Write a function **g = imageTranslate4e(f,tx,ty,mode)** for performing image translation, where **f** is a grayscale image and **tx** and **ty** are translation factors (they can be any real number: positive, negative, or zero) in the x (vertical) and y