Welcome to the labs for EELE 5110 Image Processing Lab. This lab will get you started with using MATLAB and the Image Processing Toolbox (IPT) to accomplish numerous basic image processing tasks. It prepares students for more specialized topics and techniques in the subsequent weeks.

1 Getting Started

1.1 Images as Matrices

In MATLAB, most images are stored as two-dimensional matrices, in which each element of the matrix corresponds to a single pixel in the image. For example, an image composed of 200 rows and 300 columns of pixels would be stored in MATLAB as an M-by-N matrix of 200-by-300 matrix elements.

However we note that unlike the coordinate convention used in many textbooks, the matrix coordinates in MATLAB originate at \((r,c) = (1,1)\) instead of \((x,y) = (0,0)\). Furthermore, some images, such as truecolor RGB images, require a three-dimensional matrix, where the first component matrix in the third dimension represents the red pixel intensities, the second component matrix represents the green pixel intensities, and the third component matrix represents the blue pixel intensities.

1.2 IPT Basics

IPT extends MATLAB with almost 200 functions for processing digital images. These functions include routines for inputting (outputting) images from (to) files such as `imread` and `imwrite`; displaying images on the monitor such as `imshow`; geometric transformation such as `imresize` and `imrotate`; and many other image processing operations.

To start with, let us read an image into a matrix and then have it displayed onto the monitor:

```matlab
>> f = imread('rose.tif');
>> imshow(f)
```

You should see a grayscale or intensity image of a rose pop up in the figure window. By default, `imshow` attempts to display the image at 100% magnification (one screen pixel for each image pixel).
However, if an image is too large to fit in a figure window, imshow scales the image to fit onto the screen and issues a warning message.

```
>> figure, imshow(imrotate(f,180))
>> imwrite(imrotate(f,180), 'rose-upside-down.tif')
```

Now, the rose image should appear upside down in a new figure window and you have also created a new file for the rotated rose image.

To see the information of your new image file:

```
>> imfinfo('rose-upside-down.tif')
```

Next we will use `colormap` to tweak the colormap property of a figure. A colormap is an M by 3 matrix of RGB-tuples, where each tuple specifies a color by the red, green and blue component values in the RGB color model.

False colormap can be used to enhance the presentation of an image. In this example, we will make use of it to investigate the uniformity of the image background. Estimating and subtracting an image background are two preliminary steps often used for object detection processing.

```
>> g = imread('rice.tif');
>> imshow(g), colormap('JET'), figure, imshow(g)
>> pixval on
```

In the rice.tif image, the background illumination is brighter in the center of the image than at the bottom. This variation can be clearly visualized by changing the colormap property as illustrated in Fig.2. Meanwhile, the pixval command installs a bar at the bottom of the figure which interactively displays the pixel coordinates and value for whichever pixel the cursor is currently over.

If you are able to resolve all the hitches so far, Congratulations! You are likely to reward yourself with many more roses by the weekend. Otherwise, contact your tutor.
1.3 Arithmetic Operations

Image arithmetic deals with standard arithmetic operations, such as addition, subtraction, multiplication, and division, on image pixels. Image arithmetic has many uses in image processing both as a preliminary step in complex operations and by itself.

- **Addition:** \( I(x,y) = \min[I_1(x,y) + I_2(x,y), I_{\text{max}}] \)
- **Subtraction:** \( I(x,y) = \max[I_1(x,y) - I_2(x,y), I_{\text{min}}] \)
- **Division:** \( I(x,y) = \frac{I_1(x,y)}{I_2(x,y)} \) where \( I_2(x,y) \neq 0 \)

Sensibly \( I_2(x,y) \) may take the form of a constant:
- **Addition:** \( I(x,y) = \min[I_1(x,y) + C, I_{\text{max}}] \)
- **Subtraction:** \( I(x,y) = \max[I_1(x,y) - C, I_{\text{min}}] \)
- **Division:** \( I(x,y) = \frac{I_1(x,y)}{C} \) where \( C \neq 0 \)

The formulation can also be extended to obtain the negative of an image:
- **Complement:** \( I(x,y) = I_{\text{max}} - I_1(x,y) \)

Note \( I(x,y) \) are in the range of values permitted by its datatype. For example if \( I(x,y) \) is an 8-bit image, the pixel values are mapped only to the range of 0-255. Overflow values i.e. \( I(x,y) > 255 \) and \( I(x,y) < 0 \) will be clipped at 255 and 0 respectively.

IPT offers support for several arithmetic operations. To browse the list of IPT functions, type:

```matlab
>> help images
```

Among others, you should notice the following arithmetic functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imabsdiff</td>
<td>Compute absolute difference of two images.</td>
</tr>
<tr>
<td>imadd</td>
<td>Add two images, or add constant to image.</td>
</tr>
<tr>
<td>imcomplement</td>
<td>Complement image.</td>
</tr>
<tr>
<td>imdivide</td>
<td>Divide two images, or divide image by constant.</td>
</tr>
<tr>
<td>imlincomb</td>
<td>Compute linear combination of images.</td>
</tr>
<tr>
<td>immultiply</td>
<td>Multiply two images, or multiply image by constant.</td>
</tr>
<tr>
<td>imsubtract</td>
<td>Subtract two images, or subtract constant from image.</td>
</tr>
</tbody>
</table>

Note that these functions have been written to observe various range requirements e.g. uint8 and uint16 datatypes frequently used to represent images in MATLAB.

To add two images:

```matlab
>> f = imread('rose.tif');
>> g = imread('rice.tif');
>> figure, imshow(imadd(f,g))
```

Fig.3: [a] the rose.tif image. [b] the rice.tif image. [c] the image obtained by adding [a] and [b].
The result is illustrated in Fig.3. In this example, pixels in \( y \) are computed by adding corresponding pixels in \( f \) and \( g \):

\[
y = f + g
\]

Suppose the output image is of 8-bit unsigned integer datatype, so the following range requirement applies:

\[
y = f + g \\
y = 255 \text{ if } (f + g) > 255 \quad \text{and} \\
y = 0 \quad \text{if } (f + g) < 0.
\]

1.4 Exercises

**Exercise 1.1:** Read the rose.tif and rice.tif images into variables \( f \) and \( g \).

a) Try various false colormaps on the two images.
b) Checking pixel values with \text{pixval} on the two images with and without the false colormaps. Any changes?
c) Using \text{imrotate}, rotate \( f \) by 45° counter clockwise. Save the result as \text{f1lc.tif}.
d) Using \text{imresize}, reduce \( f \) by half, then enlarge the result by double, write the result using \text{imwrite} as \text{f1ld.tif}. Compare the original image with the result obtained, what happened?
e) Compute \text{grayslice(image,n)} where \( n=2,4,16, \text{and } 32 \) for the two images. Comments?
f) Write the two images as jpeg files.

**Exercise 1.2:** Using \( f \) and \( g \) in Exercise 1.1, compare and comment the following:

a) Compute \( a = f-g \) and \( b = g-f \) using \text{imsubtract}.
b) Compute \( c = f+g \) using \text{imadd}, where \( c \) is of \text{uint16}. Compare the result with image illustrated in Fig.3 [c].
c) Compute \( d = (f+g)/2 \) and \( e = \text{imlincomb}(.5,f,.5,g) \).
d) Compute \( i = \text{imcomplement}(f) \).

**Exercise 1.3:** Using \( f \) and \( g \) in Exercise 1.1, create a black and white \{0,1\} mask image of the same size. Crop \( f \) and \( g \) with the mask. (Hint: using AND operator).
2 Creating Image Effects

In Exercise 1.3, we create a mask image and use it along with a Boolean operator to crop the rose image. Suppose a mask of the keyhole shape is chosen, the result image shall somewhat look stylistic. In fact many image editing effects can be created by simple image arithmetic and straightforward matrix manipulations.

2.1 Solarization Effects

Solarization is a photographic effect in which the image appears partly positive and partly negative. A simple solarization effect can be created by taking the complement of pixels in an image whose intensity values are less than 128.

\[ I(x,y) = \begin{cases} I(x,y) & \text{if } I(x,y) > 128, \\ 255 - I(x,y) & \text{if } I(x,y) \leq 128. \end{cases} \]

2.2 Ripple Effects

Ripple effects which simulate the reflection of an object on the surface of a pond or an object seen through wavy glasses can be created by pixilated effects obtained by adding and subtracting moduli.

\[
\begin{align*}
\text{>> } x &= 1:16; \\
\text{>> } x + \text{mod}(x, 4) \\
\text{ans} &= 2 4 6 4 6 8 10 8 10 12 14 12 14 16 18 16
\end{align*}
\]

Fig.4 : Two solarization effects on the rose image.

Fig.5 : [a] The pond ripple effect. [b] The wavy glass effect.
2.3 Oil-Painting Effects

An oil-painting effect can be created by taking the pixel values most frequently occurred in a small pixel neighborhood.

Fig.6: The oil-painting effects.

2.4 Exercises

Exercise 2.1: By using the equations in Section 2.1, write a function that produces the solarization effect in Fig.4.

Exercise 2.2: Try out the M-Files for the oil-paint effect and the ripple effect. Based on the image effects at hands, create your own image effect(s). Explain how it is done.

4 Conclusions

Now everyone should have more than a dozen of roses to take home and hopefully each of you is now more comfortable with processing images with IPT and MATLAB.

5 References