Lab 3 Information

- The material in this section is informational. Please read through the section as it helps you work on the lab exercises in the next section. There may be code examples in this informational section. You are welcome to copy-and-paste them to MATLAB to run the code, but no submission is needed on any test run.

Images in MATLAB

For our purpose here, we will divide images into two categories: grayscale ("black-and-white") and color images:

- A greyscale image can be thought of as a two-dimensional discrete-"time" signal with the two "time" dimensions being the vertical and horizontal pixel indices (or simply the pixel *location*) and the value of the signal being the intensity of the pixel at that pixel location.
- A color image can be thought of as a collection of three "grayscale" images, with each set of three pixels with the same location indicating the intensities of the three primary colors (red, green, and blue) at that location of the image.

In MATLAB, a greyscale image can be represented by a matrix. The row (first) and column (second) indices of the matrix respectively give the vertical and horizontal pixel indices while the value of an element in the matrix stores the intensity of the image at that location. On the other hand, a color image can be represented by 3-dimensional array (a collection of three matrices) with the third index ranging from 1 to 3 corresponding to the primary colors of red, green, and blue.

One may use the `imagesc` function to display grayscale and color images in MATLAB. For a grayscale image, `imagesc` uses the current *color map* to display the image. The color map can be set using the `colormap` function. To display a true grayscale image, we need to set the current color map to grayscale

```
>> colormap(gray);
```

Otherwise, `imagesc` will display the grayscale image using pseudo colors specified by the current color map. You should do

```
>> help imagesc
>> help colormap
```

to learn more about the two functions.

Lab 3 Exercises

- Unless stated otherwise, you must submit your solutions to all the lab exercises in this section.
- Your laboratory solutions should be submitted on Canvas as a single PDF. The simplest way is to put your codes and answers for all the lab exercises in a single MATLAB Publisher script and use `%%` to separate the codes and answers for different exercises into different sections as described in the information section of Lab 1.

Exercise 3.1:

The piece of MATLAB code below provides some examples of how to generate, display, and manipulate grayscale images. Copy the code to a MATLAB script and replace each corresponding comment with the appropriate description. This piece of code is designed to show you how to work with images in MATLAB.

Note: You may use the MATLAB function `help` to learn more about the MATLAB functions used in the code. You should also run the code to help you understand how it works and help you write your comments. **You may use and/or modify elements of this MATLAB code to answer questions asked in Lab Exercise 3.2 below.**

```
% USER DEFINED VARIABLES
w = 15;          % Width
x = 1:160;       % Horiztonal Axis
y = 1:80;        % Vertical Axis

% ==> Comment about the next line here <==
z = round(127*exp(-1/w.^2*((y.'-40).^2+(x-80).^2)));

% ==> Comment about the next line here <==
colormap(gray);

% ==> Comment about the next three lines here <==
[xs,ys,zs] = image_system1(z,2,2);
za         = image_system2(zs,-10,35);
zb         = image_system3(za,-30,35);


% PLOT RESULT WITH SUBPLOT
figure(1);
subplot(2,2,1);          % ==> Add comment about this command <==
imagesc(x, y, z);        % ==> Add comment about this command <==
axis image;              % ==> Add comment about this command <==
title('Original')
subplot(2,2,2);          % ==> Add comment about this command <==
imagesc(xs, ys, zs);     % ==> Add comment about this command <==
axis image;              % ==> Add comment about this command <==
title('After System 1')
```

```
subplot(2,2,3);          % ==> Add comment about this command <==
imagesc(xs, ys, za);     % ==> Add comment about this command <==
axis image;              % ==> Add comment about this command <==
title('After System 2')
subplot(2,2,4);          % ==> Add comment about this command <==
imagesc(xs, ys, zb);     % ==> Add comment about this command <==
axis image;              % ==> Add comment about this command <==
title('After System 3')

function [xs, ys, zs] = image_system1(z,Dx,Uy)
%IMAGE_SYSTEM1   ===> Describe function here <===

  % ==> Comment about the next line here <==
  zs = zeros(ceil(Uy*size(z,1)),ceil(size(z,2)/Dx));

  % ==> Comment about the next two lines here <==
  ys = 1:ceil(Uy*size(z,1));
  xs = 1:ceil(size(z,2)/Dx);

  % ==> Comment about the next line here <==
  zs(1:Uy:end,1:end) = z(1:end,1:Dx:end);
end

function [za] = image_system2(z,Sx,Sy)
%IMAGE_SYSTEM2   ===> Describe function here <===

  % ====> Comment about the next line here <====
  za = zeros(size(z,1), size(z,2));

  for nn = 1:size(z,1)
    for mm = 1:size(z,2)
      % ====> Comment about next line here <====
      if nn>Sy && nn-Sy<size(z,1) && mm>Sx && mm-Sx<size(z,2)
        % ====> Comment about this line here <====
        za(nn,mm) = 1/2*z(nn-Sy,mm-Sx);
      end
    end
  end
end

function [zb] = image_system3(za,Sx,Sy)
%IMAGE_SYSTEM3   ===> Describe function here <===

  % ====> Comment about next two lines here <====
  x = 0:1:size(za,2)-1;
  y = 0:1:size(za,1)-1;

  % ====> Comment about next two lines here <====
```

```
  xs = mod(x-Sx, size(za,2));
  ys = mod(y-Sy, size(za,1));

  % ====> Comment about next line here <====
  zb = za(ys+1,xs+1);
end
```

Exercise 3.2:

This exercise shows you how to use MATLAB to implement three important operations in image processing: (1) sampling (converting a large image to a small image), (2) anti-aliasing (reducing distortions from aliasing), and (3) interpolation (converting a small image to a large image).

(a) *(Sampling)* Write a MATLAB function `image_sample` implementing the following usage example:

```
>> [xs, ys, zs] = image_sample(z, D);
```

which inputs a grayscale image `z` and samples every `D` pixels in both the horizontal and vertical direction. It outputs the sampled image `zs` and the new axes `xs` and `ys`.

(b) *(Sampling)* The file `lighthouse.mat` stores the MATLAB matrix `lighthouse`. Use the MATLAB command

```
>> load('lighthouse.mat')
```

to load the matrix `lighthouse` from the MAT file into MATLAB. The matrix `lighthouse` represents a grayscale image. Apply `image_sample` to the image `lighthouse` by sampling every 2 pixels in the horizontal and vertical directions to obtain the sampled image `lighthouse_sampled`. Use `subplot` to show side-by-side images before and after sampling. Compare the two images and describe how aliasing manifests in sampling the `lighthouse` image (hint: aliasing distorts your signal). Relate this to your understanding of aliasing from class.

(c) *(Anti-aliasing)* Write a MATLAB function `image_antialias` implementing the following usage example:

```
  zaa = image_antialias(z);
```

which inputs a grayscale `z` and outputs the anti-aliased image `zaa`. Design the anti-aliasing function to compute each point of $z_{aa}[x, y]$ (i.e., the mathematical notation for `zaa`) according to the two-dimensional difference equation (i.e., a discrete-time system/filter)

$$z_{aa}[x, y] = \frac{1}{2} z[x, y] + \frac{1}{8} \left(z[x-1, y] + z[x+1, y] + z[x, y-1] + z[x, y+1] \right)$$

Hint: You may use the function `image_system2` from Lab Exercise 3.1 as a guide.

(d) *(Anti-aliasing)* Use a for-loop to apply the anti-aliasing function `image_antialias` to the high-resolution image `lighthouse` SIX times to obtain the anti-aliased image `lighthouse_aax6`, and then apply the sampling function `image_sample` on `lighthouse_aax6` to obtain a new sampled image `lighthouse_aax6_sampled`. Use `subplot` to show side-by-side the original image `lighthouse`, the anti-aliased image `lighthouse_aax6`, the sampled image `lighthouse_sampled`

in (b), and the anti-aliased-and-then-sampled image `lighthouse_aax6_sampled`. What does the anti-aliasing filter do to the image? How does it reduce aliasing? Why is this useful in real-world applications?

(e) *(Interpolation)* Write a MATLAB function `image_insertzeros` implementing the following usage example:

```
[xz, yz, zz] = image_insertzeros(zaas, U);
```

which inputs an anti-aliased and sampled (grayscale) image `zaas`, and outputs the image `zz`. The output image contains `U-1` zero-valued pixels inserted between each pixel from `zaas`, both horizontally and vertically. The function also outputs the new axes `xz` and `yz`. Hint: You may use the function `image_system1` from Lab Exercise 3.1 as a guide.

(f) *(Interpolation)* Apply `image_insertzeros` to your anti-aliased and sampled image `lighthouse_aax6_sampled` in (d) to obtain the image `lighthouse_zeros` by add one zero (i.e. `U=2`) zeros between each pixel. Your function `image_antialias` can be used to implement anti-aliasing or interpolation! Therefore, apply `image_antialias` to `lighthouse_zeros` SIX times to do interpolation to obtain the interpolated image `lighthouse_interpolated`. What are the dimensions of `lighthouse_interpolated`? Use `subplot` to show side-by-side the images `lighthouse`, `lighthouse_zeros`, and `lighthouse_interpolated`. What does the interpolation filter do to the image? Why is this useful in real-world applications?

Exercise 3.3 (Extra credits: +20 points):

Extend your functions `image_sample`, `image_antialias` and `image_insertzeros` to handle grayscale as well as color images. Repeat parts (b), (d), and (e) of Lab Exercise 3.2 using the `zebra` image in the MAT file `zebra.mat`. You may need to increase the sampling factor D in (b) if you want to see more severe aliasing after sampling. You may also need to change the number of applications of the anti-aliasing and interpolation filter in (d) and (e) to get good interpolation results. If you have the image processing toolbox, you may use the function `imread` to read in a GIF or JPEG image for testing in place of `zebra`.