CS 1112 Spring 2020 Project 4 Part B **due Monday, April 13, 11pm EDT**

Objectives

Completing Part B of this project will solidify your skills in reading image files into arrays and working with the `uint8` data type.

2 Enhance!

You will **write a function** `multiZoom()` that allows a user to display an image file and select multiple areas to "ehnahce" by zooming in. The function should take a single argument: the name of the image file. When invoked, it will first show the image in a figure window, and then the user will be asked to select an area of interest. This region will be magnified and plotted in a new figure window. The user may repeat this interaction by selecting multiple regions on the original figure until they indicate that they are done (by selecting a small region). At this point the function should return an array containing the magnified image data for the last enhanced region.

More detailed requirements, along with some implementation tips, are outlined below:

- Your function should work for color (8 bits per channel) PNG and JPEG images, both of which are supported by MATLAB's `imread()` function. The returned data should be a `uint8` array.
- Images and enhancements should be displayed "right-side-up;" this is done automatically when using the `imshow()` function. But note that doing so flips the y-axis from what we're used to: the top of the figure will have a y-coordinate of 1, while the bottom will have a larger positive y-coordinate. This is because the y-coordinate corresponds to the row index of the image array, and we number rows from top to bottom.

  You should also keep in mind that the conventional order of coordinates differs between plots and arrays: a point (x, y) specifies the horizontal (x) position before the vertical one, but a matrix index `(r,c)` specifies the vertical (row) position first. Make good choices of variable names to avoid mixing these conventions up!
- The user will select a region of interest by clicking on two different points in the image, which should be interpreted as two opposite corners of a rectangle. The statement `[xvec, yvec] = ginput(n)` accepts `n` user mouse clicks in the current figure window and returns the x and y coordinates of the clicks in the vectors `xvec` and `yvec` (both of which have length `n`). So `(xvec(1),yvec(1))` are the coordinates of the first click, etc. Again, x corresponds to the column number of the corresponding location in the array, while y corresponds to the row.
- The coordinates of the clicked points need to be processed in order to derive a rectangular region of pixels in the array. In particular,
  - Coordinates might not be integers (this happens for various technical reasons depending on the resolution of the user's monitor). They should therefore be rounded to integers before trying to interpret them as row and column indices.
  - Coordinates outside of the image bounds should be "clamped" to the boundary of the image (i.e., if the user clicks at an x-coordinate of -2, the closest pixel in the image would have a column number of `1`).
  - The user may select the two corners of their rectangle in any order (e.g. top-left, then bottom-right, or perhaps bottom-left, then top-right). You should be able to construct a valid subarray regardless of their choice.

- The selected region should be magnified $\sim 2\times$ in width and height using the 2D interpolation described in Exercise 9: add one pixel in between each pair of neighboring pixels and set its value equal to the average of the neighboring values. Perform the interpolation on the red, green, and blue layers separately. Be careful with *types*! Exercise 9 used `double`, but image data is input and output as `uint8` (potentially making your interpolation susceptible to saturation error).
- Since we want to return to the original image after displaying a zoom detail, we need to be able to identify the window that shows the original image. Use the `figure` command: `figure(`k`)` creates a figure window and numbers it k where k is a positive integer, or if a figure window k already exists then that window is made active—shown on top of other windows. The command `figure` on its own starts a new figure window and steps up the figure window counter. Therefore, before displaying the original image, use the command `figure(1)`. Then the subsequent display of a zoom detail can be coded like this:

```
figure          % start new figure window
imshow(newIm)   % newIm is the uint8 array containing the data of the zoom detail
title('Detail of selected area')
pause(2)
figure(1)       % bring figure 1 forward, i.e., make it the active figure
```

  The `pause` command is used so that the user can see the zoom detail for a moment before the original image (in figure window 1) is brought forward. Note that you don't lose the window holding the zoom detail—it is simply behind the active figure window. Start function `multiZoom()` with the command `close all` which closes all figure windows.
- If the selected region is fewer than 4 pixels tall or fewer than 4 pixels wide, that indicates that the user is done interacting with the program. This small region should not be enhanced; instead, the previous enhanced data should be returned. If no data was previously enhanced (this was the user's first selection), then an empty `uint8` array (constructed with `uint8([])`) should be returned.
- Use the `title` command appropriately to give instructions to the user on the figure window (e.g. "Make two clicks to select a rectangular area", "Goodbye", etc.)
- There are a number of opportunities for breaking down this problem into subtasks. Try to design and use at least one subfunction.

Note that you will have to write your own function header for `multiZoom()`; be sure to clearly document its inputs, arguments, and behavior. *Test* your function by running it on the provided image ("zoomMe.jpg"), or on one of your own. You should confirm that both the side effects (the interactive behavior) and the return value of the function are consistent with the specifications.

Submission

Submit your file `multiZoom.m` on CMS, along with the files for Part A.