CIS 4930/CIS 5930 Computer Vision
Homework 2, Due: October 31th
100 points

Part A: Circle detection (30 points)

In this problem, you will implement a Hough Transform circle detector that takes an input image and a fixed radius and returns the centers of any detected circles of about that size. You are not allowed to use any built-in Matlab functions for finding circles!

You can use the built-in function in Matlab or Python for edge detection, or you can modify your edge detection function in Homework 1. The following assumes you use Matlab.

- `[BW,threshOut,Gx,Gy] = edge(im,"Sobel",threshold)` -- The Matlab built-in function `edge`. Returns a binary image `BW` containing 1s where the function finds edges in the grayscale or binary image I and 0s elsewhere. `threshOut` returns the threshold value. `Gx` and `Gy` correspond to the horizontal and vertical gradients.
  - Inputs:
    - `im` is the RGB input image of class `uint8`, and
    - `threshold` is a user-set threshold for detecting edges (which can be omitted if you want to use the default threshold-- to use that, only supply the first input to the function).

Include the following in your submission:

1) [5 pts] `function [edges] = convert_variables(BW,Gx,Gy)`
   a) Inputs:
      i) `BW,Gx,Gy` are defined in the Matlab example.
   b) Outputs:
      i) `edges` is an *N*x4 matrix containing 4 numbers for each of *N* detected edge points: *N(i, 1)* is the *x* location of the point, *N(i, 2)* is the *y* location of the point, *N(i, 3)* is the gradient magnitude at the point, and *N(i, 4)* is the gradient orientation (non-quantized) at the point.
2) [15 pts] `function [centers] = detectCircles(im, edges, radius, top_k)` -- A function to find and visualize circles given an edge map.
   a) Inputs:

        i) `im` is defined in the Matlab example, `edges` is the output of from `convert_variables`
        ii) `radius` specifies the size of circle we are looking for (use `impixelinfo` to hover over the image and visually approximate the radius), and
        iii) `top_k` says how many of the top-scoring circle center possibilities to show and output.
    b) Output:
        i) `centers` is an *N*x2 matrix in which each row lists the *x, y* position of a detected circle's center.
    c) Tips:
        i) Since we use `atan` to compute gradient orientation in degrees, make sure you use `cos` and `sin` to detect circles.
        ii) Consider using `ceil(a / quantization_value)` and `ceil(b / quantization_value)` (where, for example, `quantization_value` can be set to 5) to easily figure out quantization/bins in Hough space. Don't forget to multiply by `quantization_value` once you've figured out the Hough parameters with most votes, to find out the actual *x, y* location corresponding to the selected bin.
        iii) You can use this line at the end of your function to visualize circles: `figure; imshow(im); viscircles(centers, radius * ones(size(centers, 1), 1));`.
        iv) You can ignore circles whose centers are outside the image.

3) [10 pts] Demonstrate the function applied to the provided images `jupiter.jpg` and `egg.jpg` in the HW2 folder. Display the images with detected circle(s), label the figure with the radius, save your image outputs, and include them in your submission. You can use `impixelinfo` to estimate the radius of interest manually.

Part B: Image segmentation with K-means (20 points)

For this problem, you will "quantize" a color space by applying k-means clustering to the pixels. You will map each pixel in the input image to its nearest k-means center. That is, you will replace the R,G,B value at each pixel with the *average* R,G,B value in the cluster to which that pixel belongs. This reduces the amount of information carried by the image, since pixels that had different colors now have the same color. It is also a form of segmentation, since there is only a small number of colors, and the *ID of the cluster* to which an image pixel belongs is the same as the pixel's *color*. Note that the RGB values should be treated jointly as the 3-dimensional representation of your pixel.

Include the following in your submission:

1) [15 pts] `function [outputImg, meanColors, clusterIds] = quantizeRGB(origImg, k)` which performs clustering in the 3-dimensional RGB space, and "quantizes" the image. Use the built-in Matlab function `kmeans` (and read the documentation to see how to use it, including what data types it expects). At the