CS1674: Homework 5

Due: 3/9/2021, 11:59pm

This assignment is worth 50 points.

Part I: 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 edges or circles!

The following function is provided on Canvas:

- `function [edges] = detectEdges(im, threshold)` -- A function to compute edges in an image. Converts to grayscale first, computes gradients, then computes the gradient magnitude and orientation at each pixel, and only returns those *(x, y)* locations with magnitude that is higher than the threshold. The function also displays the final thresholded edge image.

  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).

  Output:
  - `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.

Include the following in your submission:

a. [20 pts] `function [centers] = detectCircles(im, edges, radius, top_k)` -- A function to find and visualize circles given an edge map.

   Inputs:
   - `im, edges` are defined as above,
   - `radius` specifies the size of circle we are looking for (use `impixelinfo` to hover over the image and visually approximate the radius), and
   - `top_k` says how many of the top-scoring circle center possibilities to show and output.

Output:

- `centers` is an *N*x2 matrix in which each row lists the *x, y* position of a detected circle's center.

Tips:

- Since we use `atan`**`d`** to compute gradient orientation in **d**egrees, make sure you use `cos`**`d`** and `sin`**`d`** to detect circles.
- 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.
- 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));`.
- You can ignore circles whose centers are outside the image.

b. [10 pts] Demonstrate the function applied to the provided images `jupiter.jpg` and `egg.jpg`. 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. Here is an example.

Part II: 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:

a. [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). At the end of your function, show the (1) original and (2) quantized image, in a 1x2 subplot. Make sure to label the figure with the value of *k* that was used.

Inputs:

- `origImg` is an RGB image of class `uint8`, and
- `k` specifies the number of colors to quantize to (i.e. the number of clusters).

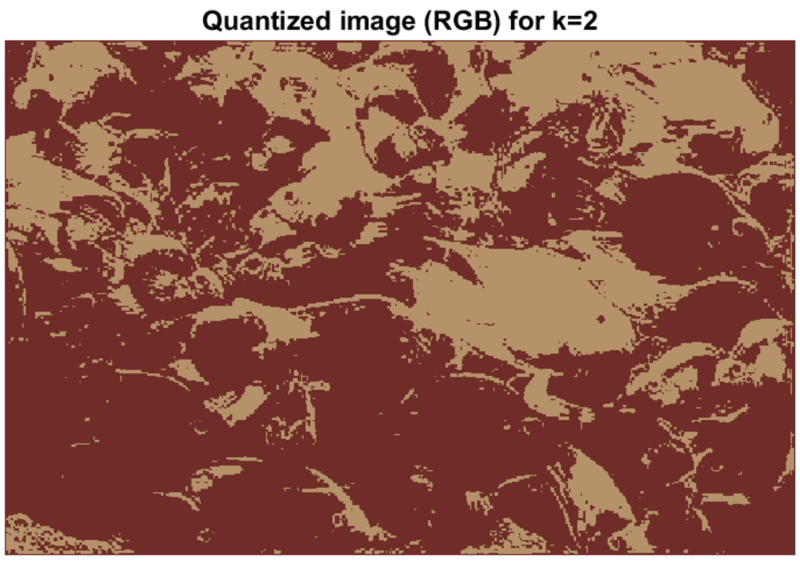
Outputs:

- `outputImg` is an RGB image of class `uint8` and the same size as the original image,
- `meanColors` is a *k*x3 array of the *k* centers (one value for each cluster and each color channel), and
- `clusterIds` is a *numpixels*x1 matrix (with *numpixels = numrows * numcolumns*) that says to which cluster each pixel belongs.

Tip:

1. If the variable `origImg` is a 3d matrix (*numrows*x*numcols*x3) containing a color image with *numpixels* pixels (first two dimensions) in each color channel (third dimension), then `X = reshape(origImg, [numpixels, 3]);` will yield a matrix with the RGB features as its rows. It is in this space (samples = rows, features = columns) that you want to apply k-means.
2. Be careful with the variable types. `kmeans` takes double. `imshow` needs either (i) an image of type `uint8` or (ii) a `double` image with rescaled intensities between 0 and 1 (e.g., divide all intensities by 255).
3. You may get a warning `Warning: Failed to converge in 100 iterations.` Do not worry about it. This just warns you that the kmeans could still be improved slightly if we let it run for longer by increasing `MaxIter` which the default is 100.

[5 pts] Run your function on the provided image `fish.jpg`. Use three different values of *k*; display and submit the results. For example, if you set *k*=2, you might get:



Submission:

- `detectCircles.m`
- demonstrations of your circle detection on the provided images `jupiter.jpg` and `egg.jpg`, for one radius (can be different for the two images), named `jupiter_circles.png` and `egg_circles.png`, respectively
- `quantizeRGB.m`
- 3 different images with different *k* applied on `fish.jpg`, named `k?.png`, `k??.png`, `k???.png`, where the question marks are replaced with the values of *k* you used

Acknowledgement: This assignment was adapted from Kristen Grauman's original assignment and Adriana Kovashka.