ECS 174: Intro to Computer Vision, Spring 2018
Problem Set 2

Instructor: Yong Jae Lee (yongjaelee@ucdavis.edu)
TA: Chongruo Wu (crwu@ucdavis.edu)
TA: Maheen Rashid (mhnrashid@ucdavis.edu)
TA: Yash Bhartia (yvbhartia@ucdavis.edu)

Due: Monday, May 21st, 11:59 PM

Instructions

1. Answer sheets must be submitted on Canvas. Hard copies will not be accepted.

2. Please submit your answer sheet containing the written answers in a file named: FirstName_LastName_PS2.pdf.

3. Please submit your code and input/output images in a zip file named: FirstName_LastName_PS2.zip. Please do not create subdirectories within the main directory.

4. You may complete the assignment individually or with a partner (i.e., maximum group of 2 people). If you worked with a partner, provide the name of your partner. **We will be using MOSS to check instances of plagiarism/cheating.**

5. For the implementation questions, make sure your code is documented, is bug-free, and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.

6. If plots are required, you must include them in your answer sheet (pdf) and your code must display them when run. Points will be deducted for not following this protocol.

1 Short answer problems [20 points]

1. When using the Hough Transform, we often discretize the parameter space to collect votes in an accumulator array. Alternatively, suppose we maintain a continuous vote space. Which grouping algorithm (among k-means, mean-shift, or graph-cuts) would be appropriate to recover the model parameter hypotheses from the continuous vote space? Briefly describe and explain.

2. Consider Figure 2 below. Each small dot denotes an edge point extracted from an image. Say we are going to use k-means to cluster these points' positions into k=2 groups. That is, we will run k-means where the feature inputs are the (x,y) coordinates of all the small dots. What is a likely clustering assignment that would result? Briefly explain your answer.

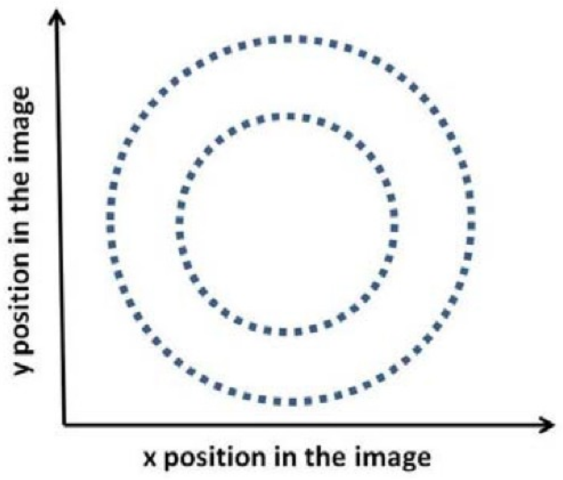

Figure 2: Edge points

3. Suppose we have run the connected components algorithm on a binary image, and now have access to the multiple foreground 'blobs' within it. Write pseudocode showing how to group the blobs according to the similarity of their area (# of pixels) and aspect ratio (width/height), into some specified number of groups. Define clearly any variables you introduce.

2 Programming [80 points]

1. Color quantization with k-means [40 points]

For this problem you will write code to quantize a color space by applying k-means clustering to the pixels in a given input image, and experiment with two different color spaces—RGB and HSV. Write Matlab functions as defined below. Save each function in a file called <function-name>.m and submit all of them.

(a) [5 points] Given an RGB image, quantize the 3-dimensional RGB space, and map each pixel in the input image to its nearest k-means center. That is, replace the RGB value at each pixel with its nearest cluster's average RGB value. Use the following form:

```
Function [outputImg, meanColors] = quantize_RGB(origImg, k)
```

where `origImg` and `outputImg` are MxNx3 matrices of type uint8, `k` specifies the number of colors to quantize to, and `meanColors` is a k x 3 array of the k centers. **Matlab tip:** if the variable `origImg` is a 3d matrix containing a color image with `numpixels` pixels, `X = reshape(origImg, numpixels, 3);` will yield a matrix with the RGB features as its rows.

(b) [5 points] Given an RGB image, convert to HSV, and quantize the 1-dimensional Hue space. Map each pixel in the input image to its nearest quantized Hue value, while keeping its Saturation and Value channels the same as the input. Convert the quantized output back to RGB color space. Use the following form:

```
Function [outputImg, meanHues] = quantize_HSV(origImg, k)
```

where `origImg` and `outputImg` are MxNx3 matrices of type uint8, `k` specifies the number of clusters, and `meanHues` is a k x 1 vector of the hue centers.

(c) [5 points] Write a function to compute the SSD error (sum of squared distances error) between the original RGB pixel values and the quantized values, with the following form:

```
function [error] =
compute_quantization_error(origImg,quantizedImg)
```

where `origImg` and `quantizedImg` are both RGB images, and error is a scalar giving the total SSD error across all pixels in the image.

(d) [5 points] Given an image, compute and display two histograms of its hue values. Let the first histogram use k equally-spaced bins (uniformly dividing up the hue values), and let the second histogram use bins defined by the k cluster center memberships (i.e., all pixels belonging to hue cluster i go to the i-th bin, for i=1,...k). Use the following form:

```
function [histEqual, histClustered] = get_hue_hists(im, k)
```

where `im` is an MxNx3 matrix representing an RGB image, and `histEqual` and `histClustered` are the two output histograms.

(e) [5 points] Write a script `color_quantize_main.m` that calls all the above functions appropriately using the provided image fish.jpg, and displays the results. Calculate the SSD error for the image quantized in both RGB and HSV space. Write down the SSD errors in your answer sheet. Illustrate the quantization with a lower (k=5) and higher (k=25) value of k. Be sure to convert an HSV image back to RGB before displaying with `imshow`. Label all plots clearly with titles.

(f) [15 points] In your write-up, explain all the results. How do the two forms of histogram differ? How and why do results vary depending on the color space? The value of k? Across different runs of k-means (with the same value of k)?

Matlab useful functions: `kmeans, rgb2hsv, hsv2rgb, imshow, im2double, reshape, subplot, title, hist.`

2. Circle detection with the Hough Transform [40 points]

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. Include a function with the following form:

```
Function [centers] = detect_circles(im, radius, useGradient)
```

where `im` is the input image, `radius` specifies the size of circle we are looking for, and `useGradient` is a flag that allows the user to optionally exploit the gradient direction measured at the edge points. The output `centers` is an N x 2 matrix in which each row lists the (x,y) position of a detected circle's center. Save this function in a file called `detect_circles.m` and submit it.

Then experiment with the basic framework, and in your write-up analyze the following:

(a) [10 points] Explain your implementation in concise steps (English, not code).

(b) [10 points] Demonstrate the function applied to the provided images jupiter.jpg and egg.jpg (and an image of your choosing if you like). Display the accumulator arrays obtained by setting `useGradient` to 0 and 1. In each case, display the images with detected circle(s), labeling the figure with the radius. You can use `impixelinfo` to estimate the radius of interest manually.

(c) [10 points] For one of the images, display and briefly comment on the appearance of the Hough space accumulator array.

(d) [5 points] Experiment with ways to determine how many circles are present by post-processing the accumulator array.

(e) [5 points] For one of the images, demonstrate the impact of the vote space quantization (bin size).

Matlab useful functions: `atan2, hold on, plot, fspecial, conv2, im2double, sin, cos, axis equal, edge, impixelinfo.`

Matlab tip: Note that the row number ("y" value) for an image position is flipped from Cartesian coordinates, increasing as we move down.

3 [OPTIONAL] Extra credit [up to 10 points]

Extend your Hough circle detector implementation to detect circles of any radius. Demonstrate the method applied to the test images.