ECE 4984/5554: Computer Vision, Fall 2015
PS2

Instructor: Devi Parikh (parikh@vt.edu)
TA: Jiasen Lu (jiasenlu@vt.edu)

Due: Monday, October 5th, 11:55 pm

Instructions

1. Answer sheets must be submitted on Scholar. 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_mat.zip if using Matlab or FirstName_LastName_PS2_py.zip if using Python. Please do not create subdirectories within the main directory.
4. You may collaborate with other students. However, you need to write and implement your own solutions. Please list the names of students you discussed the assignment with.
5. For the implementation questions, make sure your code 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. Suppose we form a texture description using textons built from a filter bank of multiple anisotropic derivative of Gaussian filters at two scales and six orientations (as displayed below in Figure 1). Is the resulting representation sensitive to orientation, or is it invariant to orientation? Explain why.

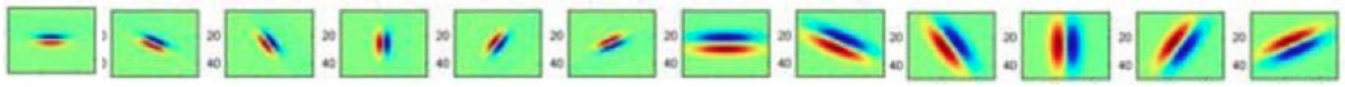

Figure 1: Filter bank

2. Consider Figure 2 below. Each small square 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 square points. What is a likely clustering assignment that would result? Briefly explain your answer.

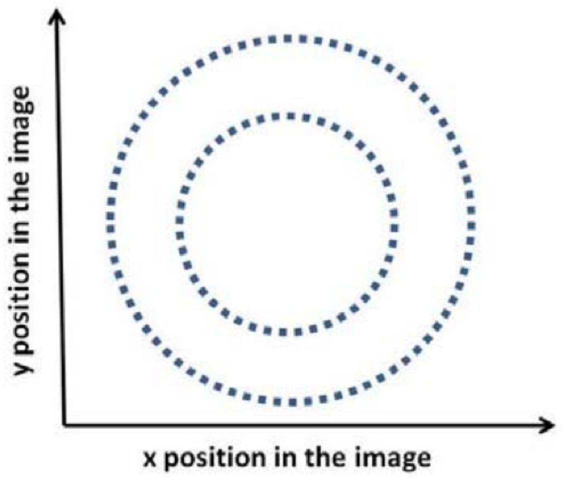

Figure 2: Edge points

3. 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.

4. 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 outer boundary shape, 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(Python) functions as defined below. Save each function in a file called <function-name>.m(py) 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:
   `[outputImg, meanColors] = quantizeRGB(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. (You can use built-in k-means function in Matlab and Python)

   (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:
   `[outputImg, meanHues] = quantizeHSV(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. (You can use built-in k-means function in Matlab and Python)

(c) 5 points. Write a function to compute the SSD error (sum of squared error) between the original RGB pixel values and the quantized values, with the following form:
`function [error] = computeQuantizationError(origImg,quantizedImg)`
where `origImg` and `quantizedImg` are both RGB images, and `error` is a scalar giving the total SSD error across the image.

(d) 5 points. Given an image, compute and display two histograms of its hue values. Let the first histogram use 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] = getHueHists(im, k)`
where `im` is an `MxNx3` matrix represeting an RGB image, and `histEqual` and `histClustered` are the two output histograms.

(e) 5 points. Write a script `colorQuantizeMain.m(py)` 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 and higher 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 writeup, 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?

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:
`[centers] = detectCircles(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 circles' center. Save this function in a file called `detectCircles.m(py)` and submit it.
Then experiment with the basic framework, and in your writeup 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. For Matlab you can use `impixelinfo` to estimate the radius of interest manually. For Python, you can directly read the image pixel information in the plot.

(c) 10 points. For one of the images, display and briefly comment on 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).

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.

Matlab hints:

1. Useful functions: `kmeans, rgb2hsv, hsv2rgb, imshow, im2double, reshape, subplot, title, hist, atan2, hold on; plot, fspecial, conv2, im2double, sin, cos, axis equal; edge impixelinfo`.
2. If the variable `im` is a 3d matrix containing a color image with `numpixels` pixels, `X = reshape(im, numpixels, 3)`; will yield a matrix with the RGB features as its rows
3. `im2double` will rescaling the data as necessary ($0-255 \rightarrow 0-1$). Make sure you rescale the data back.

Python hints:

1. Useful Modules: `numpy, scipy, skimage, matplotlib`.
2. Useful Functions: `scipy.cluster.vq.kmeans2, numpy.gradient, numpy.arctan skimage.color.rgb2hsv, skimage.color.rgb2gray, skimage.feature.canny`.
3. If the variable `im` is a 3d matrix containing a color image with `numpixels` pixels, `numpy.reshape(im, (-1, 3)) will yield a matrix with the RGB features as its rows`;

This assignment is adapted from the PS2 assignment from Kristen Grauman's CS 376: Computer Vision at UT Austin.