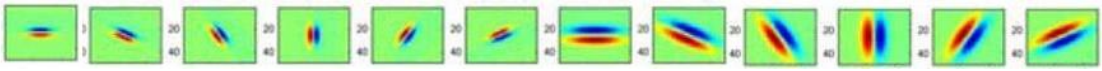
CS 378H Machine Learning and Vision
Fall 2015

Assignment 2
Out: Thursday Sept 24
Due: Friday Oct 9 (11:59 PM)

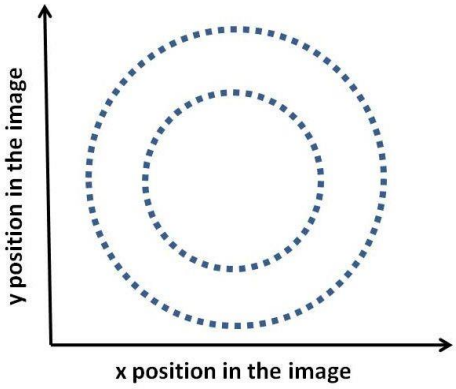
See the end of this document for submission instructions. Please see Piazza for questions and discussion from the class.

I. Short Answer [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). Is the resulting representation sensitive to orientation, or is it invariant to orientation? Explain why.



2. Consider the figure on the right. 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.



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.

II. Programming [80 points total]

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. Specifically, include each of the following components:

a) [5pts] 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] = quantizeRGB(origImg, k)
```

where `origImg` and `outputImg` are RGB images, `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 `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.

b) [5 pts] Hue-Saturation-Value (HSV) is an alternative color space to RGB. Like RGB, each pixel is represented by 3 values, only now those 3 values refer to its hue, saturation, and brightness. This gives certain advantages, including the ability to focus solely on the color dimension hue independent of the saturations or lightnesses of that color. (Read more at p. 79-80 in the textbook.) For this question, 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] = quantizeHSV(origImg, k)
```

where `origImg` and `outputImg` are RGB images, `k` specifies the number of clusters, and `meanHues` is a k x 1 vector of the hue centers.

c) [5 pts] Write a function to compute the SSD 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 pts] 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*).

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

where `im` is the input, and `histEqual` and `histClustered` are the two output histograms. When demonstrating this function call, select a single *k* value that you think illustrates the distribution adequately for the *k* cluster centers.

e) [5 pts] Write a script `colorQuantizeMain.m` that calls all the above functions appropriately using the provided image `fish.jpg`, and displays the results. 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 pts] 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?

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

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

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 circle's center. Write whatever helper functions are useful.

Then experiment with the basic framework, and in your writeup analyze the following:

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

b) [10 pts] Demonstrate the function applied to the provided images `jupiter.jpg` and `egg.jpg`, and an image of your choosing. 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 pts] For one of the images, display and briefly comment on the Hough space accumulator array.

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

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

Useful Matlab 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.

III. [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.

Submission instructions:

Submit all items electronically via Canvas as a single zip file that includes:

- A pdf "main.pdf" that contains both your answers (numbered) for Part I and your explanations and figure results for Part II. Embed the figures by the associated description and label things clearly.
- Well documented code for Part II.

Please label everything clearly so we know where to look for each piece.