Homework 3
Computer Vision, Fall 2017
Due Date: October 16, 2017
Total Points: 26

This homework contains one written assignment, one programming walkthrough and one programming challenge. All submissions are due at the beginning of class on **October 16, 2017**. The assignment should be submitted according to the instructions in the document titled "**Guidelines for Programming Assignments**" before the beginning of class. Note that there is **no credit for late submissions**. So please start working on the homework early.

Written Assignments

Problem 1: Show that if you use the line equation $xsin(\theta) - ycos(\theta) + \rho = 0$, each point in the (x, y)-image space results in a sinusoid in the (ρ, θ)-Hough space. Describe the amplitude and phase of the sinusoid in terms of (x, y). Does the period (or frequency) of the sinusoid vary with the image point (x, y)? Why or why not? **(4 points)**

Programming Assignments

This programming assignment has one walkthrough and one challenge (each with its own subset of unit tests). Instructions and summary corresponding to these are given below. **runHw3.m** will be your main interface for running your code. Parameters for the different programs or unit tests can also be set in that file. Before submission, make sure you can run all your programs with the command `runHw3("all")` with no errors.

Walkthrough 1: This walkthrough demonstrates some basic image processing tasks: convolution, smoothing and edge detection (Sobel, Canny). Complete **hw3_walkthrough1.m,** and include both the completed script and the generated outputs in your submission. **(2 points)**

Challenge 1: Your task is to develop a vision system that recognizes lines in an image using the Hough Transform. We will call it the "line finder." Test your line finder on these three images: **hough_1.png**, **hough_2.png** and **hough_3.png**.

The line finder pipeline is divided into four sub-parts, each corresponding to a program you need to write and submit.

a. First, you need to find the edge pixels in the image. You may use the built-in `edge` function to generate an edge image from the input gray-level image. Fill in the test case `challenge1a` to generate edge images. **(2 points)**

b. Next, you need to implement the Hough Transform for line detection.
```
hough_accumulator = generateHoughAccumulator(edge_img,
theta_num_bins, rho_number_bins)
```

As discussed in class, the line equation $y = mx + c$ is not suitable, as it requires a huge accumulator array. So, use the equation $xsin(\theta) - ycos(\theta) + \rho = 0$.

Be careful while choosing the range of possible θ and ρ values and the number of bins for the accumulator array. A low resolution will not give you sufficient accuracy in the estimated parameters. On the other hand, a very high resolution will increase computations and reduces the number of votes in each bin. If you get bad results, you may want to vote for small patch of bins rather than a single bin in the accumulator array. After voting, scale the values of the accumulator so that they lie between 0 and 255 and return the resulting accumulator. In the **README**, write down what voting scheme you used (and why). **(6 points)**
Functions not allowed: `hough(), houghlines()`

c. To find "strong" lines in the image, scan through the accumulator array looking for peaks. You can either use a standard threshold to find the peaks or use a smarter method. Briefly explain the method you used to find the peaks in your **README**. You can assign zero to `hough_threshold` if you did not use the standard threshold method. After having detected the peaks that correspond to lines, draw the detected lines on a copy of the original image (using the MATLB `line` function). Make sure you draw the line using a color that is clearly visible in the output image. Use the trick described in `demoMATLABTricksFun` to save the displayed image along with its annotations.
`line_detected_img = lineFinder(orignal_img, hough_accumulator, hough_threshold)`
Function not allowed: `houghpeaks`
(6 points)

d. Note that the above implementation does not detect the end-points of line segments in the image. Implement an algorithm that prunes the detected lines so that they correspond to the line segments from the original image (i.e., not infinite). Briefly explain your algorithm in the **README**. Again, you can assign zero to `hough_threshold` if you did not use the standard threshold method.
`line_segement_detected_img = lineSegmentFinder(orignal_img, hough_accumulator, hough_threshold)`
Function not allowed: `houghpeaks`
(6 points)