Homework 2
Computer Vision CS 4731, Fall 2014
Due Date: October 9, 2014
Total Points: 43

This homework contains two written assignments, two programming walkthroughs and two programming challenges. All submissions are due at the beginning of class on **October 9, 2014**. The written assignments should be turned in as a hard copy at the beginning of class, and the walkthroughs and challenges should be submitted according to the instructions in the document titled **CS4731_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 the extreme values of moment of inertia (E) for a 2D binary object are given by the eigenvalues [1] of the 2 × 2 matrix:

$$\begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

where a, b, c, and E are as defined in the lecture notes. **(3 points)** Argue that E is real and non-negative, and hence prove that $4ac \geq b^2$. **(2 points)** When is E equal to zero? **(1 point)**

Problem 2: 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 two walkthroughs and two challenges (each with its own subset of unit tests). Instructions and summary corresponding to these are given below. **runHw2.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 `runHw2("all")` with no errors.

Walkthrough 1: This walkthrough demonstrates how to convert a gray-level image to a binary image and how to use morphological operations (erosion, dilation) to "clean" a binary image. Complete **hw2_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 two-dimensional objects in images. For example, we might be interested in detecting the two objects shown in **two_objects.png** in another image such as **many_objects_1.png**. The vision system should not only determine whether the objects are present in the image, but also compute their positions and orientations.

The recognition pipeline is divided into four sub-parts, each corresponding to a program you need to write.

a. Write a program named `generateLabeledImage` that converts a gray-level image to a binary image using a threshold value and segments the binary image into several connected regions:
`labeled_img = generateLabeledImage(gray_img, threshold)`
Select any threshold that results in "clean" binary images for the gray-level ones given to you. You should be able to use the same threshold value for all the images. You are allowed to use `bwlabel` to generate the labeled image. In the labeled image, the background should be labeled as 0, and the maximum value of a label should be equal to the total number of objects. **(2 points)**
Functions not allowed: `im*()`, `bwconncomp()`

b. Write a program named `compute2DProperties` that takes a labeled image from the previous step and computes properties for each labeled object in the image. Store these properties in an objects database.
`[obj_db, out_img] = compute2DProperties(gray_img, labeled_img)`
The generated object database `obj_db` should be a 2D matrix, with each column corresponding to an object and each row corresponding to a property. The first six rows should correspond to the following properties:
1. Object label,
2. Row position of the center,
3. Column position of the center,
4. The minimum moment of inertia,

5. The orientation (angle in degrees between the axis of minimum inertia and the horizontal axis, positive = clockwise from the horizontal axis),
6. The roundness.

You can compute any additional properties if you want. However, these should appear after the six properties mentioned above. Describe the additional properties in your **README** file. The computed properties for all the training objects will serve as your object model database. The output image `out_img` should display the positions and orientations of objects on the original image `gray_img`. Use a dot or star to annotate the position and a short line segment originating from the dot for orientation (refer to `demoMATLABTricks` in **hw2_walkthrough1.m** for examples to draw dots, lines, and to save an annotated image). Apply `compute2DProperties` to the labeled image of **two_objects.png**. **(5 points)**
Functions not allowed: `regionprop()`

c. Now you have all the tools needed to develop the object recognition system. Write a program named `recognizeObjects` that recognizes objects from the database:
`output_img = recognizeObjects(gray_img, labeled_img, database)` .
Your program should compare (using your own comparison criteria) the properties of each object in a labeled image file with those from the object model database. It should produce an output image, which would display the positions, and orientations of only the recognized objects on the original image (using dots and line segments, as before). Using the object database generated from **two_objects.png**, test your program on the images **many_objects_1.png** and **many_objects_2.png**. In addition, use **many_objects_1.png** as your object database and find the corresponding objects in the other images. In your **README** file, state the combination criteria and thresholds that you used. **(5 points)**

Walkthrough 2: This walkthrough demonstrates some basic image processing tasks: convolution, smoothing and edge detection (Sobel, Canny). Complete **hw2_walkthrough2.m,** and include both the completed script and the generated outputs in your submission. **(2 points)**

Challenge 2: 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 `challenge2a` to generate edge images. **(1 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 `demoMATLABTricks` (in **runHw2.m**) to save the displayed image along with its annotations.
```
line_detected_img = lineFinder(orignal_img,
hough_accumulator, hough_threshold)
```

Function not allowed: `houghpeaks`
(5 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`
(5 points)

References

[1] E. W. Weisstein, "Eigenvalue," [Online]. Available: http://mathworld.wolfram.com/Eigenvalue.html.