Homework 2: Line Detection

CS 639, Fall 2020

Due on October 13

Total points: 16

Please follow the homework guidelines in your submission.

`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 that you can run all your programs with the command `runHw2("all")` with no errors.

Walkthrough 1: Edge detection (1 point)

This walkthrough demonstrates some basic image processing tasks: convolution, smoothing, and edge detection (Sobel, Canny). Complete `hw2_walkthrough1.m`, and include both the completed script and the generated output in your submission.

Challenge 1: LineFinder (15 points)

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) **Edge detection** (1 point):

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.

b) **Hough Transform** (5 points):

Next, you need to implement the Hough Transform for line detection.

```
hough_accumulator = generateHoughAccumulator(edge_img, theta_num_bins,
                                             rho_num_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 $x\,sin(\theta) - y\,cos(\theta) + \rho = 0$.

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 a 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).

Functions not allowed: `hough()`, `houghlines()`

c) **Find peaks** (4 points):

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 use 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 MATLAB `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(original_img, hough_accumulator,
                               hough_threshold)
```

Function not allowed: `houghpeaks`

d) **Line segments** (5 points):

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_segment_detected_img = lineSegmentFinder(original_img,
                                              hough_accumulator,
                                              hough_threshold)
```

Function not allowed: `houghpeaks`