Homework 1
M1522.001000 Computer Vision (2017 Spring)

Due: March 29 Wednesday 10:59AM.

Revision v1.0

There are 6 questions on this assignment, and 100 points in total. Last question involves **MATLAB** programming to answer.

Put your **code and writeup** into a directory called "`(studentid)-(yourname)-HW1`" and pack it into a `tar.gz` or `zip` named "`(studentid)-(yourname)-HW1`". For example, `201721234-gildonghong-HW1.tar.gz` or `201721234-gildonghong-HW1.zip`. Your writeup should be **typed** with **English**. Please do **not** attach your code to writeup.

Email your `tar.gz` or `zip` file ONLY to `ta.cv@vision.snu.ac.kr` with title as "`(studentid)-(yourname)-HW1`". Refer to the web page for the policies regarding collaboration, due dates, extensions, and late days.

Your homework should be formatted as following, for example:

```
(studentid)-(yourname)-HW1
├─ writeup.pdf
└─ matlab
   ├─ myImageFilter.m
   ├─ myEdgeFilter.m
   ├─ myHoughTransform.m
   ├─ myHoughLines.m
   ├─ myHoughLineSegments.m
   ├─ houghScript.m          (already provided, submit your modified copy)
   ├─ drawLine.m             (already provided)
   └─ yourHelperFunctions.m  (optional)
```

Please refer to Question 6 for the details of the files in `matlab` folder.

1 Composing Filters [5 pts]

Consider the following three filters $\mathcal{G}$, $\mathcal{E}$ and $\mathcal{M}$. $\mathcal{G}$ is a Gaussian smoothing kernel, $\mathcal{E}$ is one of the linear kernels used by the Sobel edge detector and $\mathcal{M}$ is a median filter. Is applying $\mathcal{G}$ to an image followed by $\mathcal{E}$ equivalent to applying $\mathcal{E}$ to an image followed by $\mathcal{G}$? How about if $\mathcal{M}$ is used in place of $\mathcal{G}$? In both cases, explain your answer.

Hint: Think about the properties of convolution.

2 Decomposing a Steerable Filter [10 pts]

In the continuous domain, a two dimensional Gaussian kernel $\mathcal{G}$ with standard deviation σ is given by $G(x, y) = \frac{1}{2\pi\sigma^2} \exp(-\frac{x^2+y^2}{2\sigma^2})$. Show that convolution with $\mathcal{G}$ is equivalent to convolving with $\mathcal{G}_x$ followed by $\mathcal{G}_y$, where $\mathcal{G}_x$ and $\mathcal{G}_y$ are 1-dimensional Gaussian kernels in the x and y coordinate respectively, with standard deviation σ. From a computational efficiency perspective, explain which is better, convolving with $\mathcal{G}$ in a single step, or the two step $\mathcal{G}_x$-and-$\mathcal{G}_y$ approach.

3 Hough Transform Line Parameterization [5 pts]

Show that if you use the line equation $x \sin\theta - y \cos\theta + p = 0$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y). Does the period (or frequency) of the sinusoid vary with the image point (x, y)?

4 Impulse Response [5 pts]

Show that Direc Delta Function's convolution preserves original function f.
$\int_{-\infty}^{\infty} f(\tau)\delta(x-\tau)d\tau = f(x)$, where

$$\delta(x) = \begin{cases} \lim_{\epsilon \to 0} \frac{1}{2\epsilon} & \textit{if } \; x \geq -\epsilon \;\; \text{or} \;\; x \leq \epsilon \\ 0 & \textit{otherwise} \end{cases}$$

5 Generalized Hough Transform [10 pts]

Consider the equation for a parabola in the 2-D plane, $y = ax^2 + b$. How would you use a Hough transform to detect such parabolae given a set of points?

Hint1: How many parameters did we use for line / circle detection?
Hint2: Assume the center of a parabola is given as (x_{center}, y_{center}).

6 Hough Transform for Line Detection [65 pts]

In this question, you will implement some basic image processing algorithms and putting them together to build a Hough Transform based line detector. Your code will be able to find the start and end points of straight line segments in images. We have included a number of images for you to test your line detector code on. Like most vision algorithms, the Hough Transform uses a number of parameters whose optimal values are (unfortunately) data dependent (*i.e.* a set of parameter values that works really well on one image might not be best for another image). By running your code on the test images you will learn about what these parameters do and how changing their values effects performance.

Many of the algorithms you will implement, as part of this assignment, are functions in the MATLAB image processing toolbox. You are **not** allowed to use calls to functions in this assignment. You may however compare your output to the output generated by the image processing toolboxes to make sure you are on the right track.

We have included a wrapper script named `houghScript.m` that takes care of reading in images from a directory, making function calls to the various steps of the Hough transform (the functions that you will be implementing) and generates images showing the output and some of the intermediate steps. You are free to use and modify the script as you please, but make sure your final submission contains a version of the script that will run your code on all the test images and generate the required output images. Also, please do not miss required materials that are mentioned on each question in your `writeup` file.

1. Convolution [10 pts]

   Write a function that convolves an image with a given convolution filter.

   `function` [$\boldsymbol{I}_{conv}$] = `myImageFilter` ($\boldsymbol{I}_{gs}$, $\boldsymbol{S}$)

   The function will input a grayscale image $\boldsymbol{I}_{gs}$ and a convolution filter stored in matrix $\boldsymbol{S}$. The function will output an image $\boldsymbol{I}_{conv}$ of the same size as $\boldsymbol{I}_{gs}$ which results from convolving $\boldsymbol{I}_{gs}$ with $\boldsymbol{S}$. You will need to handle boundary cases on the edges of the image. For example, when you place a convolution mask on the top left corner of the image, most of the filter mask will lie outside the image. One solution is to pad a zero value at all these locations. But, for the better result, we will pad the value of nearest pixel that lies inside the image. In the interests of running

time, you might want your function to treat kernel $\boldsymbol{S}$ that are just row or column vectors and not full matrices separately, but this is optional. Your code should **not** include MATLAB's `imfilter, conv2, convn, filter2` functions or other similar pre-defined functions. You may compare your output to these functions for comparison and debugging.

2. Edge Detection [20 pts]

   Write a function that finds edge intensity and orientation in an image.

   `function` $[\boldsymbol{I}_m\ \boldsymbol{I}_o\ \boldsymbol{I}_x\ \boldsymbol{I}_y]$ = `myEdgeFilter` $(\boldsymbol{I}_{conv}, \sigma)$

   The function will input a grayscale image $\boldsymbol{I}_{conv}$ and σ (scalar). σ is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. The function will output $\boldsymbol{I}_m$, the edge magnitude image; $\boldsymbol{I}_o$ the edge orientation image and $\boldsymbol{I}_x$ and $\boldsymbol{I}_y$ which are the edge filter responses in the x and y directions respectively.

   First, use your convolution function to smooth out the image with the specified Gaussian kernel. This helps reduce noise and spurious fine edges in the image. To find the image gradient in the x direction $\boldsymbol{I}_x$, convolve the smoothed image with the x oriented Sobel filter. Similarly, find $\boldsymbol{I}_y$ by convolving the smoothed image with the y oriented Sobel filter. After then, the edge magnitude image $\boldsymbol{I}_m$ and the edge orientation image $\boldsymbol{I}_o$ can be calculated from $\boldsymbol{I}_x$ and $\boldsymbol{I}_y$.

   In many cases, the high gradient magnitude region along an edge will be quite thick. For finding lines, it is most preferred to have edges that are a single pixel wide. Towards this end, make your edge filter implement **non maximal suppression**, that is for each pixel look at the two neighboring pixels along the gradient direction and if either of those pixels has a larger gradient magnitude then set the edge magnitude at the center pixel to zero. Please attach explanation and an example of your non maximal suppression in your `writeup` file.

   Your submitted code can not call on **MATLAB**'s `edge` function or other similar functions. However, it is okay to use `edge` for comparison and debugging.

3. The Hough Transform [15 pts]

   Write a function that applies the Hough Transform to an edge magnitude image.

   `function` $[\boldsymbol{H}]$ = `myHoughTransform` $(\boldsymbol{I}_m, I_{min}, r_\rho, r_\theta)$

   $\boldsymbol{I}_m$ is the edge magnitude image, I_{min} (scalar) is a edge strength threshold used to ignore pixels with a low edge filter response. r_ρ (scalar) and r_θ (scalar) are the resolution of the Hough transform accumulator along the ρ and θ axes respectively. $\boldsymbol{H}$ is the Hough transform accumulator that contains the number of 'votes' for all the possible lines passing through the image.

   First, threshold the edge image. Each pixel (x, y) above the threshold is a possible point on a line and votes in the Hough transform for all the lines it could be a part

of. Parameterize lines in terms of θ and ρ such that $x \sin\theta - y \cos\theta + \rho = 0$, where θ lies between 0 and 2π and the range of ρ is large enough to accommodate all lines that could lie in an image. The accumulator resolution needs to be selected carefully. If the resolution is set too low, the estimated line parameters might be inaccurate. If resolution is too high, run time will increase and votes for one line might get split into multiple cells in the array. In your `writeup`, please attach the grayscale image of $\boldsymbol{I}_m$, $\boldsymbol{H}$, and specify your parameters used in this question.

For debugging purpose, you may use `hough` function. However, your final code can not call on **MATLAB**'s `hough` function or other similar functions.

4. Finding Lines [10 pts]

   `function` $[\boldsymbol{l}_\rho\ \boldsymbol{l}_\theta]$ = `myHoughLines` $(\boldsymbol{H}, r_\rho, r_\theta, n)$

   $\boldsymbol{H}$ is the Hough transform accumulator; r_ρ and r_θ are the accumulator resolution parameters and n is the number of lines to return. Outputs $\boldsymbol{l}_\rho$ and $\boldsymbol{l}_\theta$ are both n x 1 vectors that contain the parameters (ρ and θ respectively) of the lines found in an image. Ideally, you would want this function to return the ρ and θ values for the n highest scoring cells in the Hough accumulator. But for every cell in the accumulator corresponding to a real line (likely to be a locally maximal value), there will probably be a number of cells in the neighborhood that also scored highly but shouldn't be selected. These non maximal neighbors can be removed using non maximal suppression. Note that this non maximal suppression step is different to the one performed earlier. Here you will consider all neighbors of a pixel, not just the pixels lying along the gradient direction. You can either implement your own non maximal suppression code or find a suitable function on the Internet (you must acknowledge / cite the source). If you used your own implementation, then please briefly describe your own method in your `writeup` file.

   Once you have suppressed the non maximal cells in the Hough accumulator, return the line parameters corresponding to the strongest peaks in the accumulator. Then, please plot the lines to the original image and then attach the result of it on your `writeup`. Also, please describe why r_ρ and r_θ are needed in here.

   Your code can not call on **MATLAB**'s `houghpeaks` function or other similar functions. However, we recommend you to use `houghpeaks` for comparison and debugging.

5. Fitting Line Segments [10 pts]

   Implement an algorithm that prunes the detected lines into line segments that do not extend beyond the objects they belong to.

   `function` $[\boldsymbol{l}]$ = `myHoughLineSegments` $(\boldsymbol{l}_\rho, \boldsymbol{l}_\theta, \boldsymbol{I}_m, I_{min})$

   Your function should output $\boldsymbol{l}$, which is a MATLAB array of structures containing the pixel locations of the start and end points of each line segment in the image. The start location of the i th line segment should be stored as a 2 x 1 vector $\boldsymbol{l}$`(i).start`

and the end location as a 2 x 1 vector in $\boldsymbol{l}$`(i).end`. After you compute $\boldsymbol{l}$, please plot the lines to the original image and attach the result of it on your `writeup` file. (*p.s.* If multiple line segments can be drawn on one peak point, then please draw the longest one.)

You are allowed to use `houghlines` for comparison and debugging purpose. However, your cannot include **MATLAB**'s `houghlines` function or other similar functions in your final implementation.

Revision History

Revision	Date	Author(s)	Description
1.0	2017/03/15	TA	1. HW1 out