Homework 1
M1522.001000 Computer Vision (2019 Spring)

Due: March 28 Thursday 01:59PM.

There are 5 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, `201921234-gildonghong-HW1.tar.gz` or `201921234-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
   ├─ myGaussianFilter.m
   ├─ myLaplacianFilter.m
   ├─ mySobelFilter.m
   ├─ myNonMaximumSuppression.m
   ├─ myDoubleThresholding.m
   ├─ edgeScript.m            (already provided, submit your modified copy)
   └─ yourHelperFunctions.m   (optional)
```

Please refer to Question 5 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 Identity for Convolution [10 pts]

What kind of function g convolves with any other function f to give f back? i.e. $f(x) * g(x) = f(x)$. Explain how the function act with proper equations.

3 Decomposing a Steerable Filter [10 pts]

In the continuous domain, a two dimensional Gaussian kernel $\mathcal{G}$ with standard deviation σ is given by $G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$. 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.

4 Convolution theorem [10 pts]

In class, we show that convolution operator in spatial domain is equivalent to multiplication in frequency domain. i.e. $g(x) = f(x) * h(x) \iff G(w) = F(w)H(w)$. Also, it is known that multiplication in spatial domain is equivalent to convolution operator in frequency domain. Show, $G(w) = F(w) * H(w) \Rightarrow g(x) = f(x)h(x)$. Use Inverse Fourier Transform as $f(x) = \int_{-\infty}^{\infty} F(w)\exp(2\pi i x w)dw$.

5 Edge Detection [65 pts]

In this question, you will implement some basic edge detecting algoritms. Your code will be able to find the edges in images. We have included a number of images for you to test your edge detector. Like most vision algorithms, edge detectors 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 `edgeScript.m` that takes care of reading in images from a directory, making function calls to the various steps of the edge detectors (the functions that you will be implementing) and generates images showing the output. 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. Laplacian Filter [10 pts]

   Write a function that convolves a given image by the Laplacian kernel.

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

   The function will input a grayscale image $\boldsymbol{I}_{gs}$. Build Laplacian kernel and pass it into `myImageFilter` along with $\boldsymbol{I}_{gs}$ for convolution. You will find some noise in your results. Explain how to denoise it in your `writeup`.

3. Differential of Gaussian [10 pts]

   Write a function that convolves a given image by the Gaussian kernel with σ.

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

The function will input a grayscale image $\boldsymbol{I}_{gs}$ and a standard deviation of the Gaussian kernel σ. You will need to build a Gaussian kernel fits to σ and pass it into `myImageFilter` along with $\boldsymbol{I}_{gs}$ for convolution. Note that the calculating differential part is already implemented in `edgeScript.m`.

4. Sobel Filter [15 pts]

   Write a function that finds edge intensity and orientation in an image using the Sobel kernel.

   `function` [$\boldsymbol{I}_m$ $\boldsymbol{I}_o$] = `mySobelFilter` ($\boldsymbol{I}_{conv}$)

   From now on, you will implement Canny Edge Detector which detect edges from gray scale image. There is three parameters σ, *highThreshold*, *lowThreshold*. Where, σ is the standard deviation of the Gaussian smoothing kernel to be used before edge detection. And, *highThreshold*, *lowThreshold* are the threshold value using in Double Thresholding mechanism.

   First, use your Gaussian filter function to smooth out the image with the specified Gaussian kernel. This helps reduce noise and spurious fine edges in the image. Fortunately you can use `myGaussianFilter` for that. To find the edge magnitude image and edge orientation image, implement **Sobel filter**. In function, find 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 $\boldsymbol{I}_y$.

5. The Non-maximum Suppression [15 pts]

   Write a function that applies the Non-maxium Suppression.

   `function` [$\boldsymbol{I}_n$] = `myNonMaximalSuppression` ($\boldsymbol{I}_m$, $\boldsymbol{I}_o$)

   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, implement **Non-maximum 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`.

6. The Double Thresholding [5 pts]

   Writhe a function that applies the Double Thresholding.

   `function` [$\boldsymbol{I}_{edge}$] = `myDoubleThresholding` ($\boldsymbol{I}_n$, *highThreshold*, *lowThreshold*)

   Finally, implement **Double Thresholding** function for getting clear image. Unlike standard thresholding, the mechanism uses high and low threshold values. If an

edge pixels gradient value is higher than the high threshold value, it is marked as a strong edge pixel. If an edge pixels gradient value is smaller than the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel. If an edge pixel's value is smaller than the low threshold value, it will be suppressed. So far, the strong edge pixels should be involved in the final edge image. But, there will be some debate on the weak edge pixels. Here, you need to pick weak edge pixels if at least one of it's neighboring pixel is a strong edge. Explain how the results change as the parameters change using examples in your `writeup`.

Revision History

Revision	Date	Author(s)	Description
1.0	2019/03/14	TA	HW1 out
1.1	2019/03/18	TA	Remove σ in 5.4 `mySobelFilter`