Homework 2
M1522.001000 Computer Vision (2021 Fall)
Due: October 19 Tuesday 3:30PM

There are four parts to this assignment, and 100 points in total. For the first three parts, write your solutions in your `writeup` files. Last part involves **PYTHON** programming to answer. Do not attach your code to the `writeup`. Put your **writeup** and **code** into a directory called "`(studentid)-(yourname)-HW2`" and pack it into a **zip** named "`(studentid)-(yourname)-HW2.zip`" For example, **20201234-gildonghong-HW2.zip**. Upload your **zip** file to **ETL** until due date. Refer to the **ETL** page for the policies regarding collaboration, due dates, extensions, and late days.

1 Image Filters [5 points]

The image pixel array on the left is convolved ($*$) with what operator $\boxed{?}$ to give the result on the right. Specify the operator by numbers within an array, state its relationship to finite difference operators of specific orders, and identify what task this convolution accomplishes in computer vision.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$* \boxed{?} \Rightarrow$

0	0	0	0	0	0	0	0
0	0	**-1**	**-1**	**-1**	**-1**	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	**1**	0	0
0	0	**-1**	**-1**	**-1**	**-1**	0	0
0	0	0	0	0	0	0	0

2 Edge Detection [10 points]

The Laplacian of Gaussian (see figure 1) is often applied to an image $I(x, y)$ in computer vision algorithms, resulting in the function $h(x, y)$:

$$h(x, y) = \int_\alpha \int_\beta \bigtriangledown^2 e^{-((x-\alpha)^2+(y-\beta)^2)/\sigma^2} I(\alpha, \beta)\, d\beta\, d\alpha$$

where

$$\bigtriangledown^2 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$$

1. What image properties should correspond to the zero-crossings of the equation, *i.e.* those isolated points (x, y) in the image $I(x, y)$ where the above result $h(x, y) = 0$? [5 points]

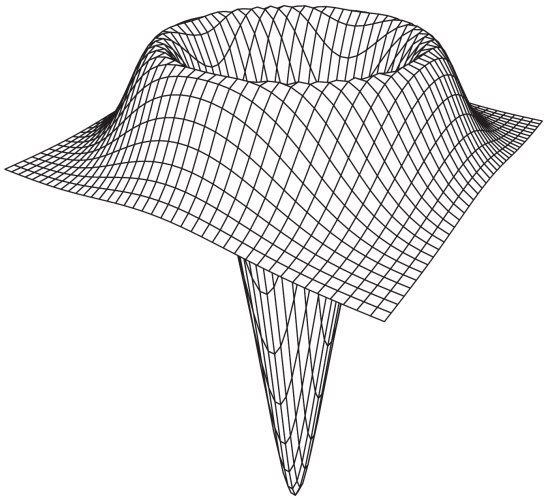


Figure 1: The Laplacian of Gaussian

2. What is the significance of the parameter σ? If you increased its value, would there be more or fewer points (x, y) at which $h(x, y) = 0$? [5 points]

3 Hough Transform Line Parameterization [10 points]

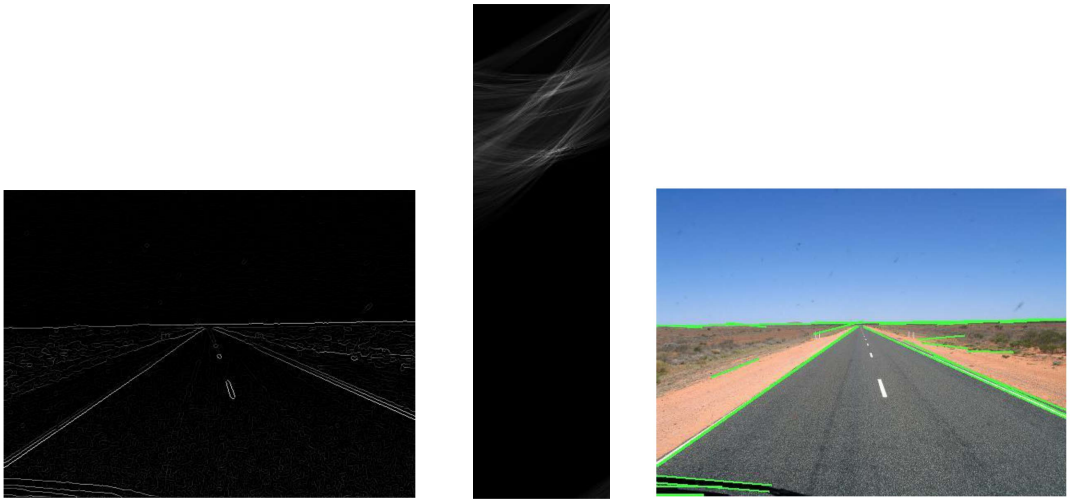
1. Show that if you use the line equation $\rho = x \sin\theta + y \cos\theta$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y). [5 points]
2. Does the period (or frequency) of the sinusoid vary with the image point (x, y)? [5 points]

4 Hough Transform for Line Detection [75 points]

In this question, you are required to detect the lines in a given image. You will implement a Hough Transform based line detector. We provided a skeleton `HW2_HoughTransform.py` to find the start and end points of straight line segments in images. 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.

Like most vision algorithms, the Hough Transform uses a number of parameters whose optimal values are (unfortunately) data dependent. By running your code on the test images you will learn about what these parameters do and how changing their values effects performance.

Usage of **OpenCV** for Hough Transform is **prohibited**.



(a) Edges (b) Hough Transform (cropped) (c) Hough Line Segments

Your program should solve the following task.

1. $\boldsymbol{I}_{conv}$ = `ConvFilter` ($\boldsymbol{I}_{gs}$, $\boldsymbol{S}$) [5 points]

   Implement a function that convolves an images with a given convoluion filter. 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.

2. $\boldsymbol{I}_m$, $\boldsymbol{I}_o$, $\boldsymbol{I}_x$, $\boldsymbol{I}_y$ = `EdgeDetection` ($\boldsymbol{I}_{conv}$, σ, *highThreshold*, *lowThreshold*) [25 points]

   Implement a function that finds edge intensity and orientation in an image. 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 `ConvFilter` to smooth out the image with the specified Gaussian kernel. 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 detection 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.

Lastly, apply the **Double Thresholding** for getting clear image. Unlike standard thresholding, the mechanism uses high and low threshold values. If an edge pixel's gradient magnitude is higher than $highThreshold$, it is marked as a strong edge pixel. If an edge pixel's gradient magnitude is smaller than $highThreshold$ and larger than $lowThreshold$, it is marked as a weak edge pixel. If an edge pixel's gradient magnitude is smaller than $lowThreshold$, 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`.

3. $\boldsymbol{H}$ = `HoughTransform` ($\boldsymbol{I}_m$, r_ρ, r_θ) [15 points]

   Implement a function that applies the Hough Transform to an edge magnitude image, $\boldsymbol{I}_m$. 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.

   Parameterize lines in terms of θ and ρ such that $\rho = x\cos\theta + y\sin\theta$. 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.

4. $\boldsymbol{l}_\rho$, $\boldsymbol{l}_\theta$ = `HoughLines` ($\boldsymbol{H}$, r_ρ, r_θ, n) [15 points]

   $\boldsymbol{H}$ 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 arrays 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.

5. $\boldsymbol{l}$ = `HoughLineSegments` ($\boldsymbol{l}_\rho$, $\boldsymbol{l}_\theta$, $\boldsymbol{I}_m$) [15 points]

   Implement an algorithm that prunes the detected lines into line segments that do not extend beyond the objects they belong to. Your function should output $\boldsymbol{l}$, which is a dictionary structure containing the pixel locations of the start and end points of each line segment in the image. The start location (x_s, y_s) of the i th line segment should be stored as an array of dictionary structure $\boldsymbol{l}$`[i]['start']` and the end location (x_e, y_e) as $\boldsymbol{l}$`[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.)

After implementing it, rename `HW2_HoughTransfom.py` to `(studentid)-(yourname)-HW2.py` and include the file in the .zip file for submission.

Finally, note that we will use a code similarity checker to detect plagiarism. You are expected to work on the assignment individually. For your sake, please do not copy and paste other student's code!