Homework 2
M1522.001000 Computer Vision (2020 Fall)
Due: October 15 Thursday 11:59PM.

There are four parts to this assignment, and 100 points in total. Last part involves **PYTOHN** 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**. Please do **not** attach your code to writeup. 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 Hough Transform Line Parameterization [10 pts]

1. Show that if you use the polar representation of lines $\rho = x\cos\theta + y\sin\theta$, each point (x, y) in the xy-plane results in a sinusoid in the (ρ, θ) plan. Relate the amplitude and phase of the sinusoid to the point (x, y).[5 pts]
2. Assuming that the image points (x, y) are in a image of width W and height H, what is the maximum absolute value of ρ and the range for θ? [5 pts]

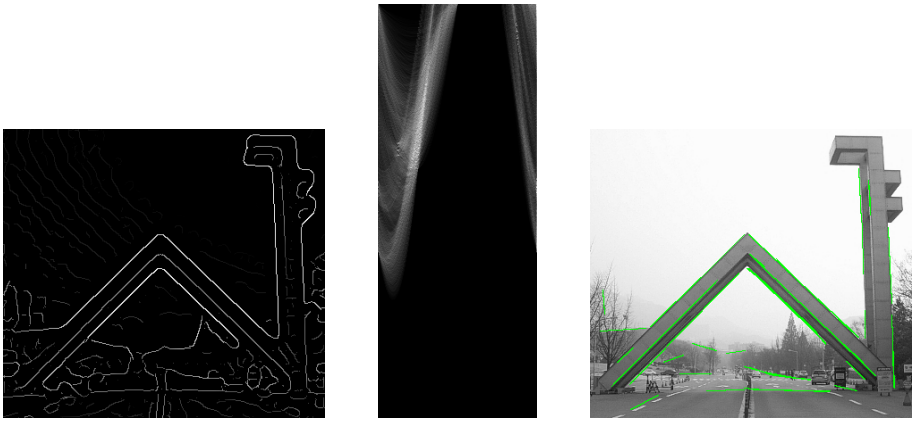
2 Lambertian Reflectance Properties [15 pts]

1. For Lambertian surfaces, the BRDF is constant function of the input and output directions. For such a material, we often describe the reflectance in terms of it's albedo, which is given the symbol . For a Lambertian surface, the BRDF and albedo are related by $f(\theta_i, \phi_i; \theta_r, \phi_r) = \rho/\pi$. Using conservation of energy, prove that $0 \leq \rho \leq 1$. [5 pts]
2. Some surface material cannot be simplified to have Lambertian reflectance properties. The "maria" of the moon, e.g., the dark formation as observed from the earth, is a typical example of a non-Lambertian surface. In this case, we observe that the surface emits radially equally in all directions, or in other words, brightness seems to increase with increasing angle between viewer and surface (hint: remember derivation of the equation for Lambertian surfaces where only the foreshortening of the incoming angle is considered). Derive the equation for the reflectance map R(p,q) in this special case. [10 pts]

3 Camera Calibration [10 pts]

A camera is a mapping between the 3D world $\mathbf{X_i} = \begin{bmatrix} X_i & Y_i & Z_i & 1 \end{bmatrix}^T$ and a 2D image $\mathbf{x_i} = \omega \begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^T$. A camera projection matric $\mathbf{P} = [m_{ij}]$ maps $\mathbf{x_i} = \mathbf{PX_i}$. Given $n \geq 6$ pairs of correspoinding points, we have an over-determined system of equations, $\mathbf{Ap} = \mathbf{0}$,where $\mathbf{p}$ is the vectorized form of $\mathbf{P}$. Consider the SVD of $\mathbf{A}$ as $\mathbf{A} = \mathbf{U\Sigma V^T}$. Show that the solution $\mathbf{p} = \min_p \|Ap\|, \|p\| = 1$ is the last column of $\mathbf{V}$ corresponding to smallest eigenvalue of $\mathbf{A}$.

4 Hough Transform for Line Detection [65 pts]



(a) Edges (b) Hough Transform (c) Hough Line segments

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**.

Your program should solve the following task.

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

   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) [20 pts]

   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.

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

   Implement a function that applies the Hough Transform to an edge magnitude image. $\boldsymbol{I}_m$ 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 $\rho =$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` $(\boldsymbol{H}, r_\rho, r_\theta, n)$ [10 pts]

   $\boldsymbol{H}$ 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` $(\boldsymbol{l}_\rho, \boldsymbol{l}_\theta, \boldsymbol{I}_m, I_{min})$ [15 pts]

   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!