# Homework 2
# M1522.001000 Computer Vision (2022 Fall)
### Due: Oct. 19 Wednesday 11:59 PM

This homework covers Edge Detection, the Hough Transform, Active Contours, and Appearance. There are 5 questions in total for this homework. You will need to submit both a **writeup** and the **code** for Problem 5.

Create your **writeup** as a pdf file named "`(studentid)-HW2.pdf`" (*e.g.,* 2022-22222-HW2.pdf). All of your writeup should be written in **English**. Do not attach your code to the writeup directly. For submission, put your **code** together with your **writeup** into a directory named "`(studentid)-HW2`" and tar it into a tar.gz named "`(studentid)-HW2.pdf.tar.gz`".

Your homework should be formatted as follows:

```
(studentid)-HW2
├── (studentid)-HW2.pdf
└── (studentid)-HW2.py
```

Upload your `tar.gz` file to **ETL** before the due date.

If your submission does not follow the format above, 20% of your total score for HW2 will be deducted.

You can discuss the homework problems with others, but you have to provide your own answers in your own words. Copying code is strictly prohibited and all homework will be checked with a plagiarism checker. Refer to the **ETL** page for detailed policies regarding collaboration, due dates, extensions, and late days.

# 1 Edge Detection [10 points]

1. [5] Why is the Laplacian alone not a good edge detector, but the Laplacian of Gaussian is? Explain in a sentence each.
2. [5] What is the general effect of increasing the Gaussian width for Laplacian of Gaussian edge detectors? Explain in a sentence.

# 2 Hough Transform Line Parameterization [10 pts]

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

# 3 Active Contours [10 points]

Imagine you are trying to implement active contours, or the 'snakes' method for finding shapes.

1. [4] You finish implementing the external energy term, but forget to implement the internal energy term. What kinds of issues may arise with your snakes? Explain in a sentence.
2. [3] You realize your mistake, and implement the internal energy term, as well. However, you notice that your snakes are too rounded, and are not fitting the shapes well. How can you address this problem? Explain in a sentence.
3. [3] You notice that in your images, the shape you want to fit to is always a rectangle. How can you encourage your snakes to fit to rectangular shapes? Explain in a sentence.

# 4 Appearance and BRDFs [10 points]

Consider a hypothetical material with a BRDF given by $\rho(\theta_i, \theta_r) = 1\cos(\theta_i)$, where $\theta_i$ is the angle made by the light direction with the surface normal, and $\theta_r$ is the angle made by the viewing direction with the surface normal.

1. [5] Will the appearance of this surface vary with the viewing direction? Explain why.
2. [5] Will it vary with the incoming light direction? Explain why.

# 5 Implementing the Hough Transform for Line Detection [60 pts]

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

For this question, you will be implementing a Hough Transform-based line detector to detect the lines in a given image. We have provided a template (skeleton) file `HW2_HoughTransform.py` for you to write your code in. Feel free modify the file as you please. Make sure that when run, your program will save the modified images to the /results directory. We will be checking the results of how your functions work on the test images provided.

Note that 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 affects performance. You may modify these parameters in the code in order to get better results.

Finally, note that some questions ask you to include images and descriptions in your **writeup**, as well.

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

You should implement the following functions:

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

   Implement a function that convolves an image with a given convolution filter. The function will take as 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 $\boldsymbol{S}$. Note that 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 would be to pad all these locations with a zero value - but, for a better result, we recommend padding using the value of the nearest pixel that lies inside the image.

2. $\boldsymbol{I}_m$, $\boldsymbol{I}_o$, $\boldsymbol{I}_x$, $\boldsymbol{I}_y$ = `EdgeDetection` ($\boldsymbol{I}_{conv}$, $\sigma$) [15 pts] Implement a function that finds edge intensity and orientation in an image. The function will receive as input a grayscale image $\boldsymbol{I}_{conv}$ and $\sigma$ (scalar). $\sigma$ 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 that, 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 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` ($\boldsymbol{I}_m$, $I_{min}$, $r_\rho$, $r_\theta$) [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. $r_\rho$ (scalar) and $r_\theta$ (scalar) are the resolution of the Hough transform accumulator along the $\rho$ and $\theta$ 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 $\theta$ and $\rho$ 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; $r_\rho$ and $r_\theta$ 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 ($\rho$ and $\theta$ respectively) of the lines found in an image. Ideally, you would want this function to return the $\rho$ and $\theta$ 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 $r_\rho$ and $r_\theta$ 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 .tar 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!