**In the Name of God, the Compassionate the Merciful**

| | | |
|---|---|---|
| Digital Image Processing<br>Prof. Kasaei<br>Due 24 Khordad | **Homework 5**<br>Image Analysis | |

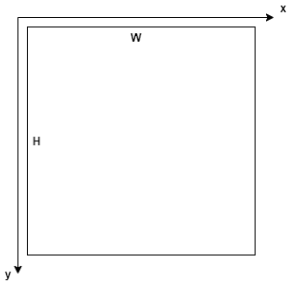
Notes:

- Problem 1 and 2 are theoretical and the rest of the problems are practical.
- If a question asks you to run the code on images, please save your results in a folder. Or you can use Jupiter notebooks and leave the results in the notebook. please try to as clean as possible, one Jupiter notebook with all the results shown in it will do and is also preferred.
- You are free to use any libraries unless said otherwise but obviously you can only use Python.
- all the images for the practical part are located in the `resources` folder.

If you have any questions, please contact me at ms.mehabadi@gmail.com.

## Problem 1: Hough Transform (25 points)

In this problem, we want to investigate Hough Transform. Hough Transform is used to detect known shapes such as lines, circles. we will focus on detecting lines. For this problem, assume that you are given an image of size $H \times W$, with an $(x, y)$ coordinate system where $x, y \in \mathbb{R}$, as follows:




Also assume that we have ran some edge detector on this image, and we are given N set of points $S = \{(x_1, y_1), \dots, (x_N, y_N)\}$ extracted from the binary edge map outputted by the edge detector such that $(x_i, y_i) \neq (x_j, y_j)$ for $i \neq j$. The problem is to find the subsets of points that lie on a straight line. Though we will simplify the problem by only trying to find the equation of lines associated with these subsets of points.

(a) A possible solution to this problem is to find all lines associated to each pair of points. Then for each line, finding how many points are close to it. compute the runtime complexity of such algorithm.

(b) In the following, we will show step by step that there is a better algorithm to find these lines. first show that any line L visible on the image, can be written as the following equation: $x\cos(\theta) + y\sin(\theta) = \rho$, such that $\theta$ and $\rho$ are the angle of the line with respect to x axis and length of perpendicular line from origin to line L, respectively.

(c) Find the min and max values of $\theta$ and $\rho$ for all the visible lines on the image.

(d) Now consider the coordinate system $(\theta, \rho)$. For each point $(x_i, y_i)$, we can plot a curve in the $(\theta, \rho)$ plane using the equation: $x_i\cos(\theta) + y_i\sin(\theta) = \rho$. what does this curve represent?

(e) Show that for any two points $(x_i, y_i)$ and $(x_j, y_j)$ where $1 \leq i, j \leq N$ and $i \neq j$. the corresponding curves in the $(\theta, \rho)$ plane of these points intersect at exactly one point.

(f) Show that for any subset of S such that its points are on the same line in the image plane, their corresponding curves in the $(\theta, \rho)$ plane will intersect at exactly one point.

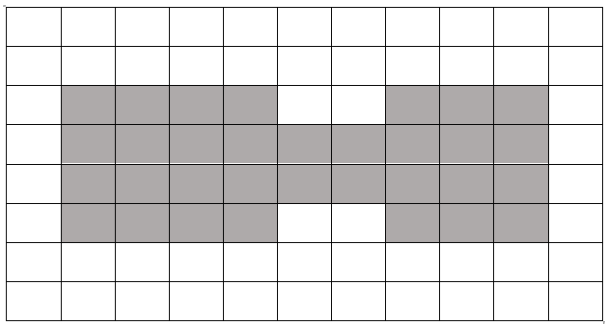
(g) The main idea behind Hough Transform is that given a set of points, S, for each point plot its corresponding curve in $(\theta, \rho)$ plane and find the intersect points with maximal curves passing through it. this gives you a set of $(\theta, \rho)'s$ that each of them corresponds to a line in the image plane. Suggest an algorithm to do this in practice and then analyze its runtime complexity.

## Problem 2: Morphological Analysis (15 points)

In this problem, we will investigate different aspects of the morphological analysis.

(a) For a structuring element B such that it only contains a single point, valued 1 and any set of points, A. What happens if we dilate A by B?

(b) For the following shape, suggest a structuring element and an operation such that it breaks the bridge between two blobs. report the final result too.



(c) Explain shortly in what scenarios we would use opening and closing.

(d) let A be a set of points. define complement of A as $A^c = \{w | w \notin A\}$. show that the following equation holds: $(A \ominus B)^c = A^c \oplus \hat{B}$.

### Problem 3: Hough Transform [Practical] (25 points)

In this problem, we want to implement Hough Transform and then compare our results with OpenCV implementation.

(a) Load and show the `coins.jpg` image.
(b) Implement Hough Transform and show the detected lines using it on the `coins.jpg` image.
(c) Do the same as in (b) only using OpenCV built-in function for Hough Transform. and finally compare your results with this part side-by-side.

### Problem 4: Morphological Analysis [Practical] (35 points)

In this problem, we will perform different morphological operations on different images. you are free to use any library you want.

(a) Load and show the `hole.jpg` image.
- perform dilation and erosion on this image using two different structuring elements and report the results. the size of the structuring element is of your choice.
- perform open and close operations on this image again using the same structuring elements. the size of the structuring element is of your choice. report and explain the results.
- using morphological operation, try to fill the hole as best as possible. explain and report your result.

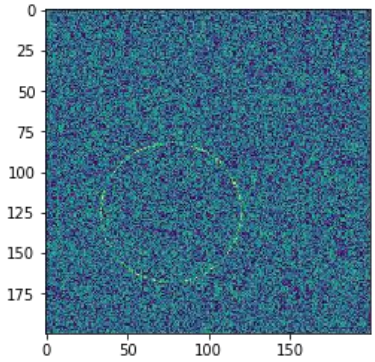
(b) Load and show the `noisycameraman.jpg` image and `cameraman.jpg` image side-by-side.
- as you can see, we have added salt noise to the `cameraman.jpg` image. now using morphological operations try to reduce the noise as much as possible without distorting the image. report and explain your result.

(c) Load and show the `circles.jpg` image.
- plot the histogram of this image.
- in this image, you see a number of dark circles in a light background. first do a thresholding and remove the background, leaving only the circles as white and background as black in a binary image. show this result.
- now use a morphological operation to separate circles from each other such that they won't touch each other anymore. explain and report your result.

(d) Load and show the `lines_circle.jpg` image.
- in this image, you see a bunch of lines and circles together. separate the lines from circles as best as possible using morphological operations. explain what operations you used and report two images, one that only contains circles and another that only contains lines. show these images side-by-side.

## Problem 5: Circle Detection [Practical] (20 extra points)

In this problem, you are going to detect a circle inside an image under presence of noise. detecting a circle means, to write a function that outputs the parameters of the circle, namely (row, column, radius) which the first two are the center coordinates of the circle and that last one is the radius of the circle inside the image. you are free to use any library and/or method that you desire to solve this problem. a noisy circle image looks like the following:




Look at the `circle_detection` folder, you will find two files:

- **`requirements.txt`:** in which, the libraries that you need to install to be able to use `main.py` is listed.
- **`main.py`:** in which some core functionalities reside.

In the `main.py` file, you will the following functions:

- **`draw_circle`:** this function receives an `img` variable, which is a numpy `ndarray`, along with the circle parameters and draws a circle of that specification inside the `img` variable. the function does this in-place, so it does not return anything.
- **`noisy_circle`:** this function receives an integer `size` variable, an integer `radius` variable and finally a `noise` variable and returns a noisy image with a circle of that `radius` in a random place. it also returns the position and radius of the circle so you are able to know the true parameters of the circle.
- **`find_circle`:** you must fill this function such that it receives an `img` variable which is a numpy `ndarray` and returns the parameters, (row, column, radius), of the circle inside the image.
- **`iou`:** receives two triples (tuples of size 3), each corresponding to a circle's parameters and returns the intersection over union (iou) value. this function is used to evaluate the returned parameters from the `find_circle` function compared to the real parameters of the circle.

- **`main`:** this function will be used to evaluate your `find_circle` implementation. it returns the mean intersection over union (mIoU). you can use this function to evaluate the performance of your `find_circle` too.

Finally, here are some notes for you:

- **I will be using images of size 200 by 200 with circles of radius 50 randomly distributed inside the image with noise level 2, to evaluate the performance of your `find_circle`.**
- you can train a deep neural network to detect the circle parameters. a simple CNN would give you a moderately good performance on the iou.

What you must deliver is as follows:

(a) simply plot a noisy circle.
(b) a fully operating `find_circle` function with mIoU>0.7. you can use `main` function to evaluate your `find_circle` before submitting your code.
(c) get a mIoU>0.9. if you have a `find_circle` that evaluates to a mIoU>0.9, this will count for the answer to (a) too.
(d) handle the corner cases! circles do not have to fully reside in the image, and rather some part of it may be outside of the image. these are corner cases. many algorithms fail to recognize the parameters of the circle when given circles in these conditions.

**Good Luck!**