Due date: **Oct. 22, 2024, 11:59 PM** (Indy time). The penalty for late submission will be applied goes as follows for each subsequent 24 hour period: {10%; 25%; 50%}. That is, if you submit this assignment on a Monday at 4:00 AM in the morning (if it were due on Friday at 11:59 PM), your submission is three days late.

What to turn in:

1. Your submission should include your complete code base in an archive file (zip, tar.gz) and q1/, q2/, q3/, and **a very very clear README describing how to run it**.

2. **A brief report (typed up in LaTex, submit as a PDF file, NO handwritten scanned copies) describing what you solved, implemented and known failure cases if any. Report will be the first thing that we will look at from your submission.**

3. Submit your entire code and report to Canvas

Notes from instructor:

- Start early!
- You may ask the TA or instructor for suggestions and discuss the problem with others (minimally). But **all parts of the submitted code must be your own**.
- Use either Matlab or Python. Problem descriptions are based on Matlab codes, but you may convert them to equivalent functions in Python.
- Do not use pre-defined functions. For Python users, you may use libraries minimally to load data or perform basic operations such as matrix multiplications.
- Make sure that the TA can easily run the code by plugging in our test data.
- Do not try to estimate what the instructor is expecting for an answer. As long as the result looks 'not too bad' and you have proper explanation, you will get credits.

Problem 1

1. (**10pts**) Write a gradient based edge detector. Your code should load in a gray-scale image (use imread and convert to a double array using `im2double`). You can display the image using functionality like `imagesc`. Once you have loaded in the image, you should smooth the image with a Gaussian filter and then compute horizontal and vertical derivatives using a derivative filter. The amount of smoothing is determined by the parameter σ of the Gaussian (which should be a parameter in your code). You can use `conv2` to perform the required convolutions, or use your own implementation from HW1. Once you have computed the derivatives in the x and y directions, compute the gradient magnitude and orientation. Display the gradient magnitude as an image and the orientation using quiver functionality.

2. (**20pts**) We want to extract the outer boundary of each object contained in one image. First, use `edge()` function (pre-defined or your own edge detector) to obtain the edge map. Then use `imview()` function to manually locate one pixel located on the outer boundary of each object. Next, apply any technique you can come up with to trace the boundary edge pixels. Try to get a boundary as complete and accurate as possible: it is OK if your results are not perfect (see below). Plot the final boundary you are able to detect and trace for each object. Discuss the success, failure, and possible ways for improvement in the report.
   Note that boundary tracing is a non-trivial problem, involving many complicated issues such as noise, broken boundaries, ambiguous locations and directions, among others. Thus, do not worry if you are not able to get a 'perfect' result | we just want to see how hard you tried.
   Do not use built-in functions such as '`bwtraceboundary()`' provided by Matlab image processing library. The idea is for you to gain familiarity with the algorithm through the experiments in this homework.

Problem 2

(Corner Point Detection, **20pts**) Implement your own Harris corner detector to automatically identify corner points in an image. Your implementation should take an 2D image I as an input and compute corner responses for each pixel. Take the points of local maxima (within a radius r) of the response.

Problem 3

(Hough Transform, **50pts**) The purpose of this component is to get familiar with the Hough transform for shape detection. You will need to implement the Hough transform on your own. The built-in functions such as `hough()` in Matlab are not allowed.

1. (**20pts**) Write a program to automatically identify straight lines in an image: `lines = myHoughLine(imBW, n)`, where `imBW` is a binary image and n is the desired number of lines (in order of voting importance). If there are not enough n lines in an image, return as many as your implementation detected. Test with n = 5, and plot your lines along with a few images in your report.
2. (**30pts**) If you haven't already, read \Generalizing the Hough transform to detect arbitrary shapes (Ballard, 1981)" and review \Lecture 6-1" for Hough transform from class. Write a program to find (roughly) circular objects of a specific radius in a binary image. You should write two functions, `yourcellvar = myHoughCircleTrain(imBW, c, ptlist)`, and `myHoughCircleTest(imBWnew, yourcellvar)`. The binary image `imBW` supplied to the first function will have a single circular object. Further, `c` will be the (given) reference point, and for convenience we will also provide you an ordered list of boundary

points in ptlist. Use this information to construct whatever data structure you want (and save all tabular data necessary for the subsequent step), and return it as a cell array variable, yourcellvar. That is, yourcellvar is the object myHoughCircleTrain should return. Next, this will be passed directly to myHoughCircleTest where you will identify circular objects in a novel image, imBWnew. Your function should report the reference points for the top two circles identified. You will receive full credit if your reference point is close enough.