BMI/CS 567 Medical Image Analysis
University of Wisconsin-Madison
Assignment #3

Instructor: Jeanette Mumford
Due March 15, 2018 *before* class begins

For this assignment you should turn in a single pdf file containing your code and the specific output requested for each problem. It shouldn't be necessary to run your code to grade the assignment. Name it `hwk3_yourname.pdf` (html is fine too). It should be submitted via canvas. The late homework will not be graded. Use sections and comments to explain which question and part your code refers to and you may also use comments in the few places I have requested written responses. Provide enough detailed justification and reasoning. For instance, a simple *yes* or *no* answer without justification or code will result in 0 points.

Important!!! You **may** use `imagesc()` on this assignment!

(1) Use the data in `medfilt_problem_dat.mat` (variable will be called `dat` when you read it in) for this exercise and it looks like Figure 1.

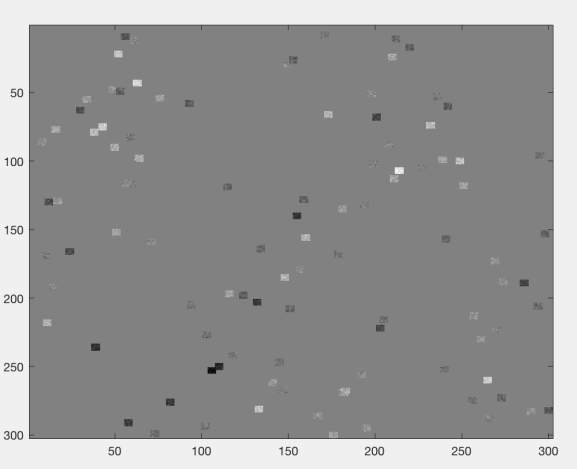

Figure 1: Image you'll be using for median filter problem.

(a) (2 points) Write the code for a median filter using a 3×3 window, using zero padding. Compare your result to what you get using `medfilt2` in MATLAB. To check this, take the difference of the two filtered images and inspect the values of the difference. Viewing the images may not be sufficient, so check the values carefully. Are the images the same? Why or why not?

(b) (1 point) You may use `medfilt2` for this part. What is the minimum filter size that removes all of the "salt and pepper" in the image?

(c) (2 points) I generated the noise in this image by randomly selecting pixels and adding noise to a 5×5 window around that pixel. For an isolated piece of this "salt" or "pepper" (meaning it isn't overlapping or near another chunk of noise), what is the minimum window size that guarantees it will be removed? You should derive this analytically. Did your solution match what you found in the previous part? Why or why not?

(2) (5 points) Linear Hough transform. For this problem you'll be working with the image shown in Figure 2, contained in `dat_hough.mat`. The goal is to apply the linear Hough transform to bring out the edges of the black shape. It is up to you to process the image to remove as much of the artifact as possible, estimate the edges and generate a binary image (edges have a value of 1 and everything else is 0). Then you apply the Hough transform as we did in class and you may use the matlab function we used in class. You may not use the built in MATLAB Hough transform function.

Please display the following images: Image after you removed the artifact (as much as possible), image after edge detection (not yet binary), the binarized edge image and the result from applying the Hough transform. You must use approaches learned in this class for each step and explain why you chose the approaches. Display the first three images in a 3×1 panel plot and the Hough transform result in a single panel plot that uses the `imfuse()` MATLAB function to combine the Hough lines and original image (we did this in class).

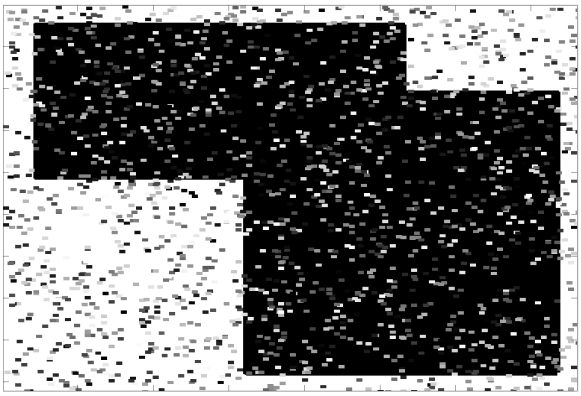

Figure 2: Image you'll be using for linear Hough Transform problem.

(3) (5 points) The Circle Hough transform. The circle Hough is very similar to the linear Hough, but instead of constructing lines through each edge point, we construct circles centered at those points. If the proper radius is chosen, the constructed circles will overlap at the center of the circles you are trying to identify. See class notes for more details. Circles require three parameters: x-coordinate of the center, y-coordinate of the center and the radius. So, unlike the linear Hough, there are 3 parameters to search over instead of 2. You will create a 3D accumulator matrix (x-coordinate, y-coordinate and radius) that counts how often a constructed circle passed through an (x,y) coordinate for a given radius. I will give you three radius values to try (48, 50 and 52). Your original image is 540×720, so your accumulator matrix is $540 \times 720 \times 3$. Use the `img_circle.mat`

file for this problem, which is also shown in Figure 3. There are two circle sizes in the image, but your goal is to find the smaller circles. I have already binarized it, so no preprocessing is required and you may simply start with the Hough transform.

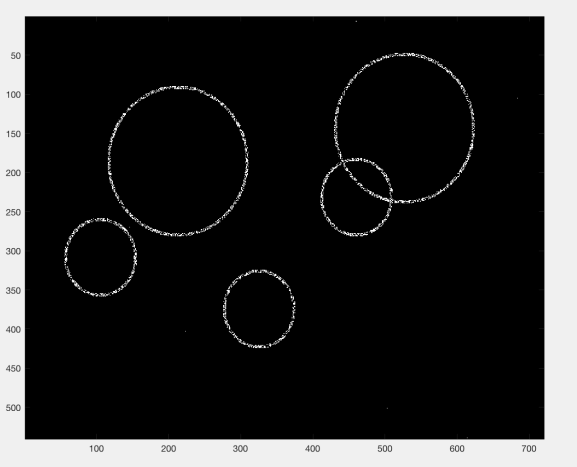

Figure 3: Image you'll be using for the circle Hough Transform problem.

No need to write a separate function, as I did for the Linear Hough in class. Here are the steps that you will need to code. For each pixel that lands on an edge with coordinates, (i, j):

1. Construct a circle centered at that point. The following will build this circle
   (a) For each radius, r, (use r=48, 50 and 52) do the following
   (b) For each angle, t from t=0-359 do the following
      i. $x = i - r \times cos(t * pi/180)$ is the x-coordinate of a point on a circle centered at your edge point
      ii. $y = j - r \times sin(t * pi/180)$ is the y-coordinate of a point on a circle centered at your edge point
   (c) Record this circle point in your accumulator matrix. If you named the accumulator matrix `accum`, you would add a count to the `accum(round(x), round(y), k)` position, where k refers to 1, 2 or 3, which is the radius you're currently on. $k = 1$ would refer to the radius of length $r = 48$, etc.
2. To identify the best radius, find the one that has the largest accumulation value. For example, `max(max(accum(:,:,1))` will be the max bin for the radius of 48, assuming you stored your accumulation values for the 48 voxel radius in this part of the array. The largest max tells you which radius is best. What was the best radius?
3. Now that you've found the best radius, you can see if it did a good job finding the smaller circle centers. As with the linear Hough, you need to threshold the accumulator matrix. Unlike the linear Hough, you don't need to do anything to transform it back to the original image, you simply need to view the two images together. Display the accumulator matrix for the best radius, unthresholded, and also use `imfuse` to fuse the original image with your estimated centers (thresholded accumulator matrix) and display using `imagesc`. All the smaller circles should have a point in the center and you shouldn't have points elsewhere if all went well! If you didn't get the centers or if you have many extra points that aren't in the centers, try a different threshold.