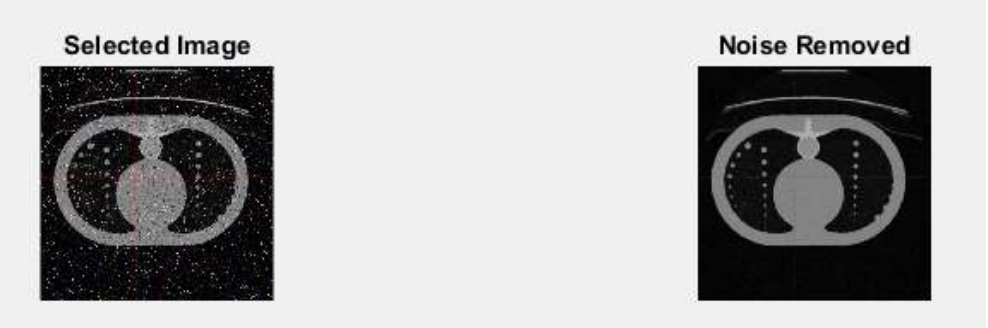
Digital image analysis of chest phantom images in CT



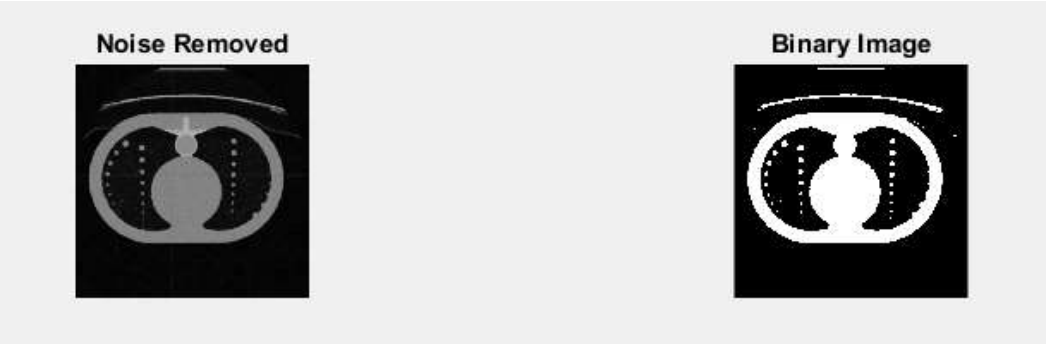
Task One

Identify type of noise?
The noise throughout most of these images is commonly referred to as salt and pepper noise this type of noise consists of certain amounts of the pixels in the image either being black or white, the most common method of removing such noise is known to be through a medium filter " filtering is less sensitive than linear techniques to extreme changes in pixel values, it can remove salt and pepper noise without significantly reducing the sharpness of an image" Therefore the approach of using a medium filter within this assignment was applied

Input, Grayscale and Noise removal
The problem with the initial image is the amount of noise present is very high which needs to be removed before any other processes can be applied to find the circles within the image, noise can be seen in the screenshot above and causes distortion within the image in terms of brightness and darkness "Noise which varies randomly above and below a nominal brightness value" (Bovik, 2009) Therefore the first step to providing the solution to this problem is the removal of any noise within the images inputted into the program. The first step to this task was to read the image into the program "Selected_Image = imread (image_jpg);" This allows for the desired input images to be read, after the image is read it is then converted from RGB to grayscale using the rgb2gray function, this allows for the removal of hue and saturated information while allowing for the ability to keep the images desired luminance.

Once the image has been read and converted the noise then can be removed. The noise is removed using the medium filtering, this is done through a separate function called "customfilter" The function applies a medium filter to the grayscale image, which then allows for each outputted pixel to contain a medium value in the 3-by-3 region around the corresponding pixels within the image. The medium filter follows a similar approach to that of smoothing techniques, the technique of using corresponding pixels allows for the better ability of the removal of noise without reducing the sharpness of an image therefore making this the appropriate function to reduce the salt and pepper noise within the images for this assignment.

After the noise has removed the image is then implemented into a subplot to output the image with its corresponding input image, this is done throughout the project to all later images implemented within the project, an example of the noise removal and subplot is shown above.



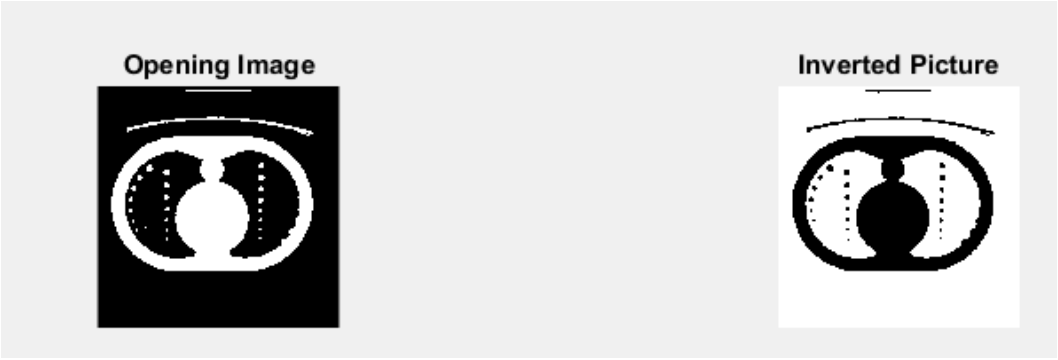
Task Two

Converting the Greyscale Image to a Binary Image

The next key step to solving the problem is to convert the current greyscale image into a binary image. The reason behind converting the image into a binary image is it allows for the better ability to detect objects within an image (circles) the reason behind this is due to the fact now only two different colours exist within the image (black, white).The image is converted into a binary image by using the im2bw function “binary_picture = im2bw(median_filtering_Image, 0.2);” This function uses the filtered image as an input and then converts the pixels within the image dependent upon the luminance level of each pixel to either the value 1 (white) or the value 0 (black).

Create a morphological structuring element (STREL)

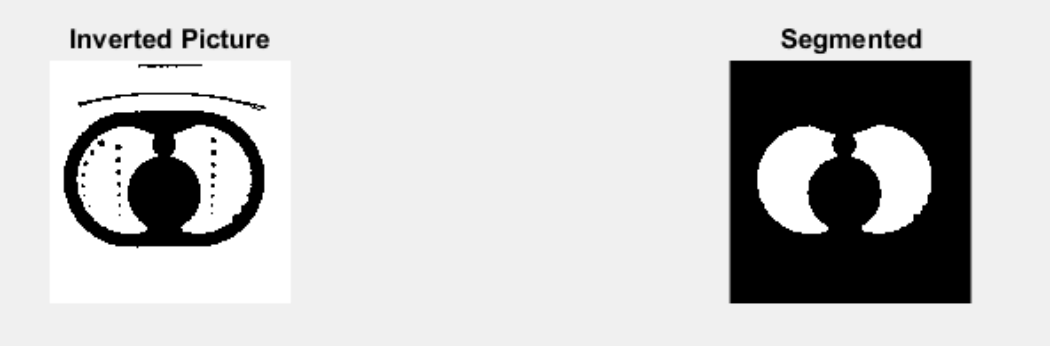
This step involves creating a structuring element in the form of a rectangle “se1 = strel('disk', 2);” This specific function creates a flat disk shape structure where “2” specifies the radius. This is done twice to create two separate structures which are applied to the initial binary image, creating two separate images “postOpenImage_1” and “postOpenImage_2” Once these structures have been applied the first image is applied into the subplot and shown to the user while the other is saved for later image processing to help solve to problem in later stages.



Inversion of the Opened Image

This step involves the inversion of the binary image which is completed by simply using a ones function, this function basically creates an array of all ones, this is implemented to the binary image “inverted = ones(size(binary_picture));” Once this is done the following is implemented “invertedImage_1 = inverted - postOpenImage_1;” This line basically takes the inverted image with

the array of ones and takes the Open image from this which in turn inverts the image by swapping all previous 0 pixels to 1 pixels and 1 pixels to 0, creating an inverted image as shown above. This is completed to allow for segmentation to be completed in the following step to allow for the program to better find the circles within the image.

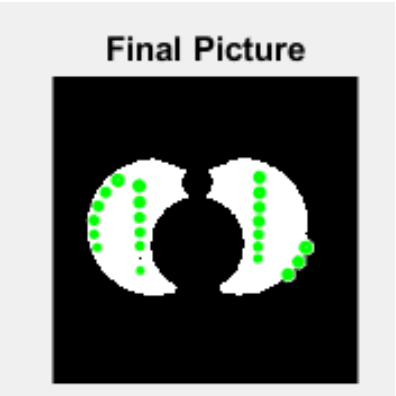


Creating an initial Contour and implementing segmentation

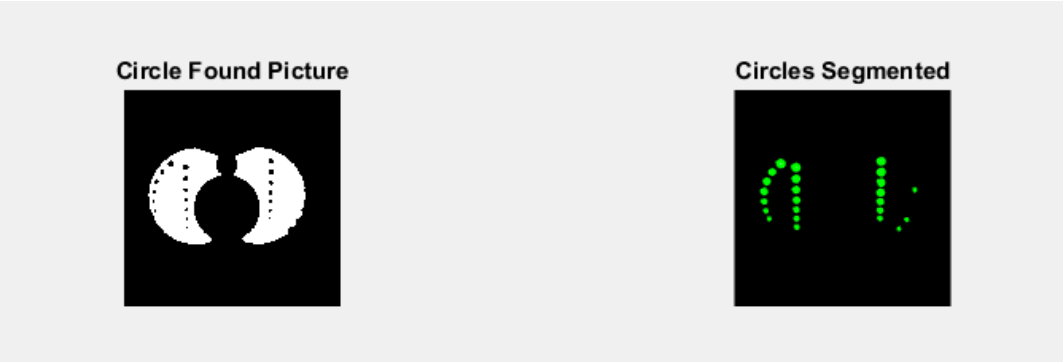
This step initially involves creating and specifying the initial contour “mask = zeros(size(invertedImage_1)); mask(50:end-50,50:end-50) = 1;” The mask is a binary image that specifies the initial state of the active contour, this involves the boundaries of the images regions in this case the white background and black regions outside the main part of the Image, this all allows for contour evolution to occur to allow for segmentation of the image to occur. Once the image is segmented it will produce a more clear way of finding the circles within the lung as this means the foreground and background no longer cause any issues when attempting to find the circles as seen above in the screenshot.

Task Three

Combination Image and Find Circles



The next process required is to find the circles within the lung, within this program the image is created using a combination of the inverted image and contour segmentation “mix_Image_1 = invertedImage_1 + bw_1;” This creates an initial image that the circles can be found upon, from this the image is then converted into black and white using “im2bw” then the final image uses the medium filter “medfilt2” to further filter the image to allow for the best possible ability to find any circles within the lung. The main process within this step is now to find the circles this is done using the imfindcircles function “[centers,radii] = imfindcircles(final_1,[1 9],'ObjectPolarity','dark','Sensitivity',0.88); This function looks for dark sensitivity within the above image using the parameters of centres and radius with a sensitivity specification, from this the circles are then displayed using green circles “viscircles(centers,radii,'EdgeColor','g'); “ After this is completed the image is then displayed as seen above in the screenshot with the circles all being highlighted in green. The circles are also counted “display(size(centers, 1), ' Numbers of Circles');” this allows for all circles found to have a centre to be counted and outputted into the command window.



Task 4

Segment Circles

After the circles have been counted and found the circles need to be Segmented and then detected. This is done by firstly creating a segmented picture “segment_pic = final_2 - final_1;” This is done by taking a picture without the circles located to a picture with the circles located, therefore leading to an image with the circles totally segmented away from the picture allowing for a precise way to find the boundaries of each circle “[B] = bwboundaries(pre_colour_pic,'holes');“ This code basically finds all circles within the image and the boundaries and then colours each circle with a fill of green to identify they have been found as seen in the screenshot above. Therefore solving all problems indicated within the brief as the circles have been counted found and all noise has been removed.

References

Bovik, A. (2009) The essential guide to image processing. [Online] Available from: http://links.uwaterloo.ca/amath391w13docs/bovik_image_processing.pdf [Accessed 23/11/2015]

Matlab (2015) Remove Salt and Pepper Noise from Images. [Online] Available from: http://uk.mathworks.com/help/vision/ug/remove-salt-and-pepper-noise-from-images.html?refresh=true [Accessed 23/11/2015]