Chau, Vincent
998947424
ECS 189G

Problem Set 2

Part I Short Answers Problems:

1) Figure 1 is sensitive to orientation, it is missing two orientations and one scale each to be invariant to rotation. To achieve rotational invariance the filter should be 8 (3 scales for 2 filters, and 2 isotropic).

2) First there will be two centroids placed randomly on the x and y axis, the closest ones to each centroid would be labeled a certain color. For example, the points that are closest to centroid C1 would be labeled yellow and points closest to centroid C2 would be labeled red. When this process is done, next calculate the new centroids, Cj and Ci which will be the mean of all points Xi is assigned to cluster Cj in previous. This process will repeat until the points are clustered into two groups. The k-means would first split the entire graph into two sections, later the points would be rearranged clustered in two directions. Lastly, the clusters would be grouped apart from each other. Possibly one close to the top right of the graph and one cluster close to the bottom left.

3) K-Means would be appropriate to recover the model parameter parameter from a continuous vote space because it works well with a continuous vote space. The similar points would be clustered together despite having a large amount of points. This will require a search for maxima in the voting space, and once it is found, extract the segmentation mask based on stored masks for the refined hypothesis.

4)

Assuming the binary image and connected component algorithm already run.
Assuming the connected components should now be labeled.

Pseudocode:

```
var colorLabel := %set each blob to a different color to make them distinct

var blobProperties := regionprops() % get properties of blobs

var n := size(blobs) % get number of blobs

var allBlobIntensity := get blob mean pixel intensities with regionprops()

var allBlobArea := get blob areas % comparing area for similar width and height

%group by pixel intensities through return properties of regionprops()

%find indices in between threshold that is similar in pixel intensity
var blobByIntesityIndices = allBlobIntensity  > threshold1 and allBlobIntesity < threshold2

% find all blobs with similar width and height if they have similar area
var blobByAreaIndices =  allBlobAreas > threshold3 and allBlobAreas < threshold4
```

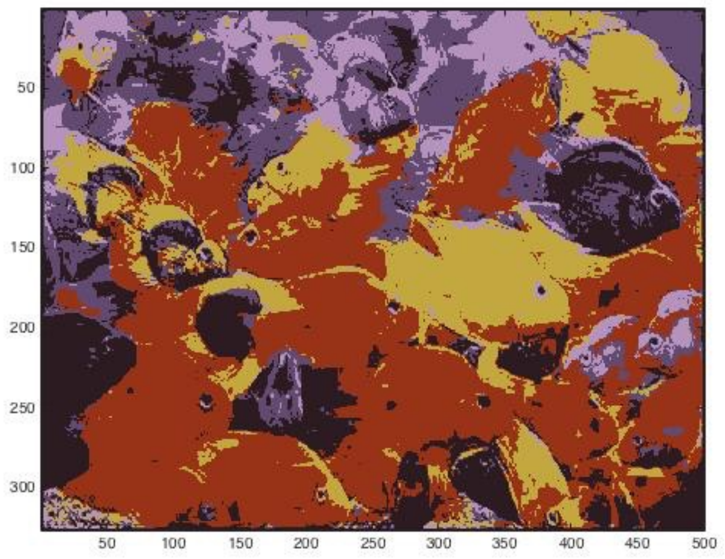
Chau, Vincent
998947424
ECS 189G

Part II Programming

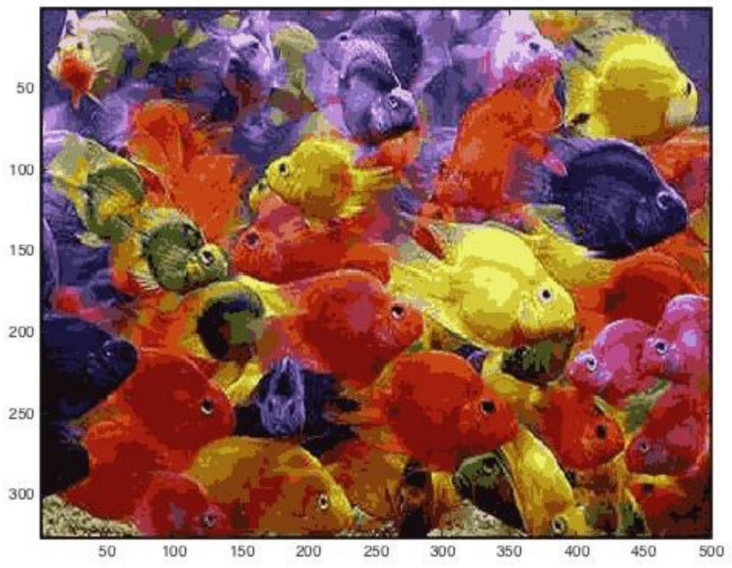
1. Color Quantization with K-Means

A.

quantizeRGB K = 5:



quantizeRGB K = 25:

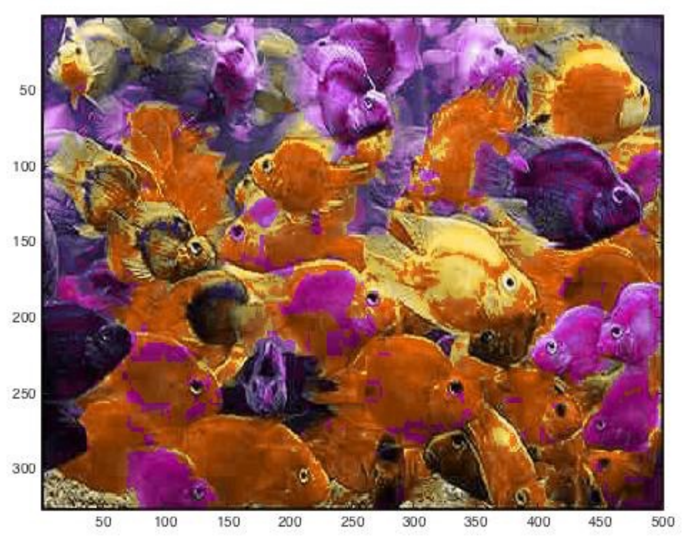


Chau, Vincent
998947424
ECS 189G

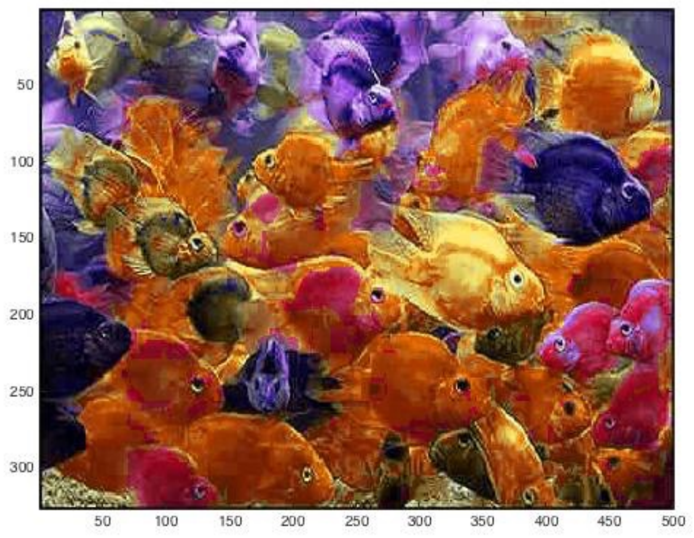
The results were as expected. All the RGB channels for K=5 are clustered into 5 bins of intensities, as seen in the image. As for K=25, there are 25 different bins and 25 different intensities throughout each channel, as shown in the image. The quantization took out the intensities in between each bin which made the original image blend, thats why for K=5, the objects in the images stand out from the rest of the objects since they were clustered to the bin.

B)

quantize HSV K = 5 (Converted Back to RGB)



quantize HSV K = 25 (Converted Back to HSV)

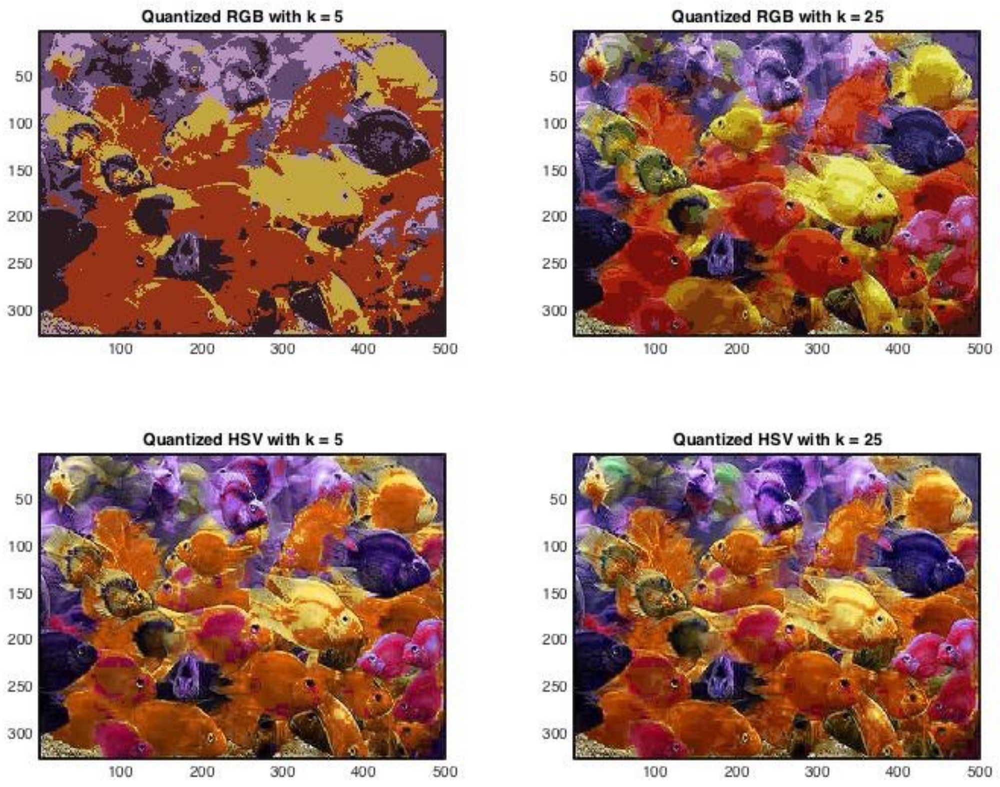


Chau, Vincent
998947424
ECS 189G

The quantization of the Hues are also as expected. There are only 5 bin of Hues when K=5 which explains the hue intensities seen in the picture. I altered the Hue channels into different bins and kept the Saturation and Value the same from the original image. When K=25, the hue resembles more of the original image as the shades of hue make the image more consistent in variety. The main noticeable difference is the fish's scale color on the bottom left of the image. It closely resembles the original image opposed to when K=5 where there is a big difference in the same fish's scale. The clusters when K=5 got moved to the Hue group of purple since the original Hue value in the original image was closer to a reddish-purple.

As for HSV vs RGB with the same k value, the HSV image resembles more of the original image as expected since altering the hue would only cluster the shade of intensities of the image and not the colors like in RGB. Altering RGB would make objects blend in if the background object has similar RGB intensities due to the cluster, this is why it is hard to differentiate the objects like when K=5.

Sub-Plotted Image of Part (A & B):



Chau, Vincent
998947424
ECS 189G

C)

computeQuantizedError:

SSD Error RGB k = 5:
R Error: 14979613
G Error: 14237969
B Error: 13399581
SSD Error RGB k = 25:
R Error: 11519078
G Error: 10705642
B Error: 11029535
SSD Error HSV k = 5:
H Error: 2856986
S Error: 9266568
V Error: 1426765
SSD Error HSV k = 25:
H Error: 2521700
S Error: 9202598
V Error: 1840459

Note: the errors were calculated with the HSV image converted back to RGB, vs the original HSV image.

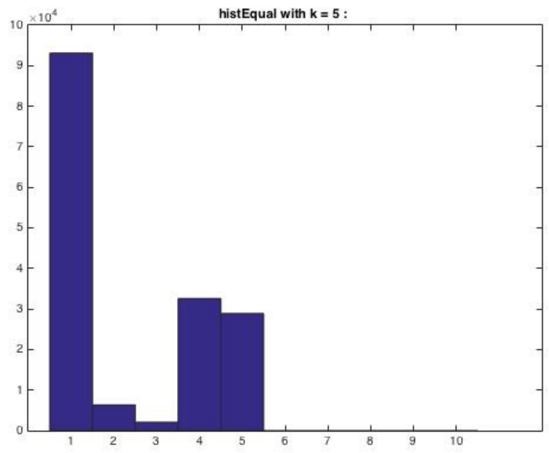
I computed each of the SSD Error by channels as opposed to computing the error as a total. I did so because seeing the different errors in channels helps give a better insight on the changes. For the RGB channel I compared each individual channels with the channels from the original image. As for the HSV, I did the same comparison.

The errors calculated when K = 5 was consistently greater than the channels of when K=25 which is accurate because the bins of intensities differs more when there are less bins to represent the range of intensities from the original image. As for the HSV values, when K = 5 the errors were also greater than when K = 25 for the same reasons as RGB. The Saturation remained somewhat consistent around 9202598, but the Value error was higher. This may be due to the transition from converting back to an RGB image after quantizing the HSV and the differences of changes in the values due to the change in Hue may have caused a change in the corresponding RGB channels.

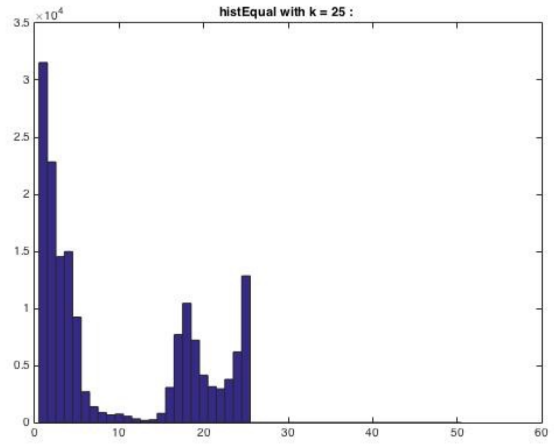
Chau, Vincent
998947424
ECS 189G

D)

HistEqual k = 5

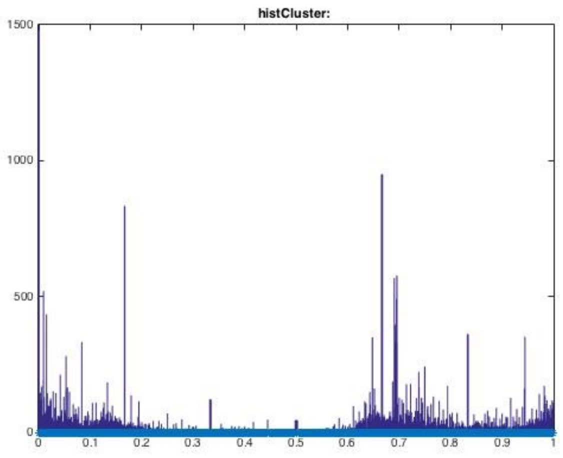


HistEqual k = 25

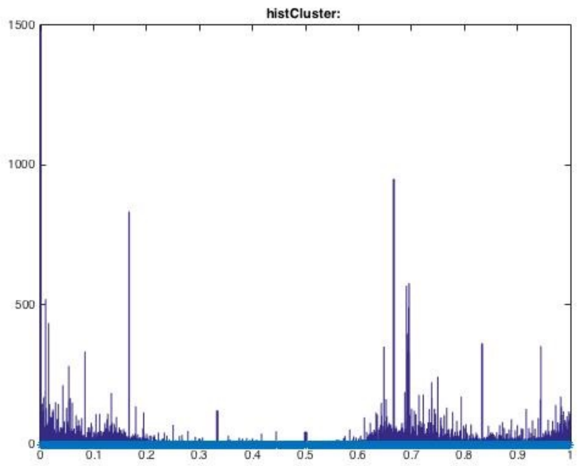


Above are the outputs of getHueHist with k=5 and k=25. As expected there are 5 bins for K=5 and the frequencies of each on the y-axis. The same goes for k=25.

HistClustered k = 5



HistClustered k = 25



Above are the outputs for HistClustered. Based on piazza the outputs are incorrect, and I have not finished debugging it. I am having trouble vectorizing the outputs. I attempted to extract the hue channel and plot it with the centers as they should be the the same values. The histogram should look like HistEqual except the x axis should be the value of the centers. I have not fixed this issue.

Chau, Vincent
998947424
ECS 189G

2. Circle Detection with Hough Transform

A.

In my implementation, I followed the lecture slide's general hough transform algorithm. I first used the canny edge detector on the grayscale image then extracted the edges with the edge function. Like the algorithm in the slide, I used two nested for loops which iterated through the edge pixels, the possible values of the radius and the direction theta. I implemented:

```
Lecture slide algorithm:
For every edge pixel(x,y)
    For each possible radius value r
        For each possible direction theta
        // or estimate at (x,y)
        a = x - rcos(theta)
        b = y + rsin(theta)
        H[a,b,r] += 1
    end
end
```

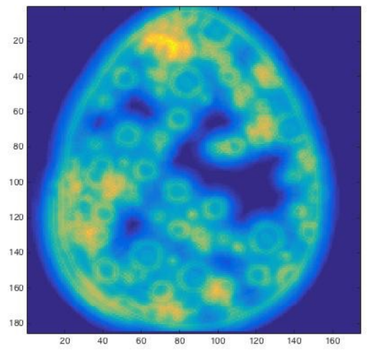
Like the algorithm, I stored scanned the possible radius from 1 to the radius of the user input. The only thing the algorithm didn't include was to check for the values going out of bounds which as a result I added checks before incrementing the accumulator. Unlike most people on piazza, I followed the algorithm and stored the radius as well in the accumulator so it is not 2-Dimensional. So my accumulator has the hough votes for different planes for the radiuses. I also made another accumulator H2 to store all the votes on the y and x coordinate despite the radius, which I did not use later for detecting the circles.

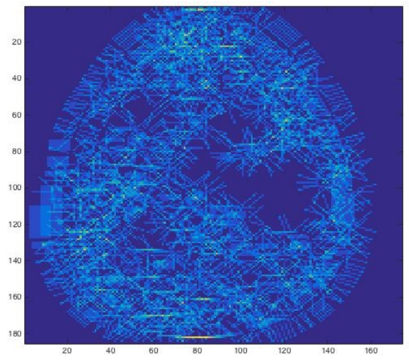
As for the useGradient, The only difference between the angle of using the gradient direction or not is that I loop over from 0 to 360 and covert it to radians before passing into cos or sin function. For the gradient, I had trouble using atan2(dY/dX), because I would get errors regarding "Operands to the || and && operators must be convertible to logical scalar values. gradient". Which I fixed by changing the && to & however my program would be in an infinite loop. My other failed attempts of using atan2() would end up having too many centers in the image where the object is, or the angle would only vote on a specific point only. My method for debugging constantly was by plotting the centers using the markers parameter in Matlab's plot function. I ended up only taking the gradient direction of the imgradient() function and using it's direction for my theta when useGradient was set to true. The results were similar and it detect more circles but there were more irrelevant points after thresholding compared to scanning from 0:360.

Furthermore, after the accumulator has finished taking votes on different planes, I thresholded the points based on hough votes. I eliminated the points that were less than 70% of the max value which got rid of many irrelevant circles. Next, I thresholded again by first sorting the votes in descending order and only keeping the first 80% of the points because at this point, the circles found were very accurate but there were some unnecessary circles still. Lastly, I stored the X and Y points into the centers which is the return value.

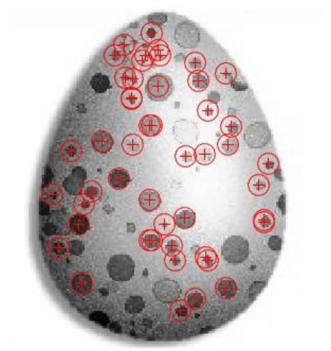
Chau, Vincent
998947424
ECS 189G
B.

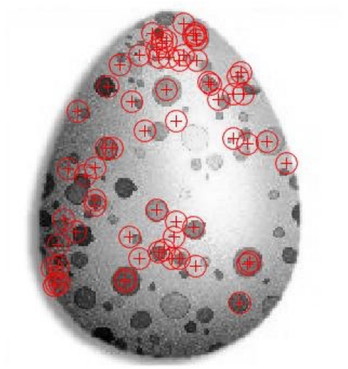
Accum: r = 10, useGradient = false r = 10, useGradient = true





Output: r = 10, useGradient = false r = 10, useGradient = true



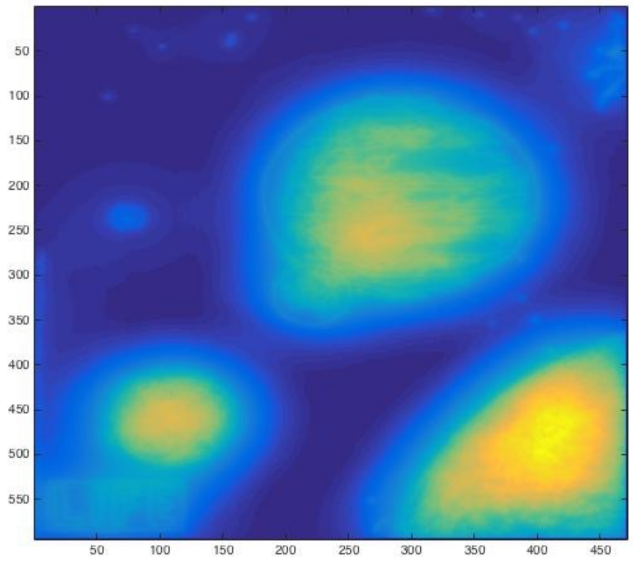


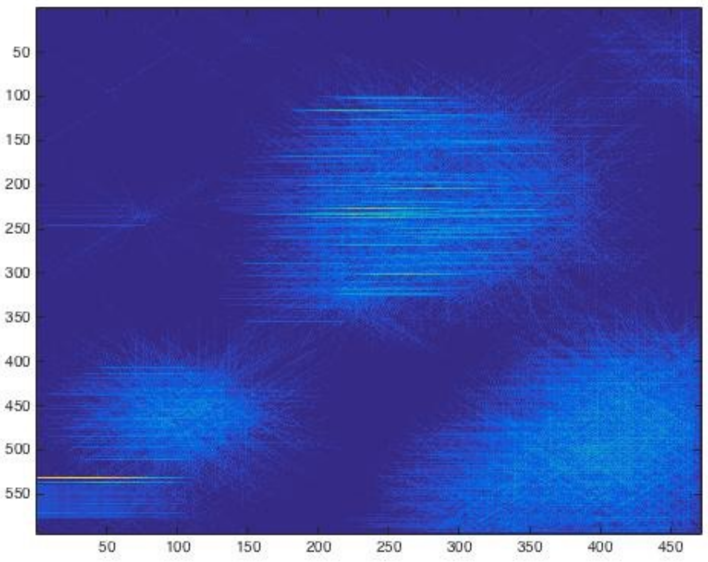
<u>Note</u>: All output plots are done with markers with the size of 2 * the radius. They are also plotted with second accumulator H2 which stores the votes on the points on all radius (from range 1 : user input radius).

Every radius will give a different output but above are the images when the radius is 10. The circles detect when scanning 0:360 were more accurate in terms of the centers, but it did not detect as much circles as when using the gradient. The gradient also picked up very few inaccurate points such as the middle of two circles where no circles exist on the image. Despite my efforts to further threshold, the gradient image still acquires a good amount of hough votes on those few inaccurate points. As for the accumulator, the useGradient detects more circles but looses a little accuracy due to the gradient directions, which is the possibility why there are few inaccurate circles. The accumulator with the range of 0:360 seems more accurate which explains why every circle it detected is correct and precise.

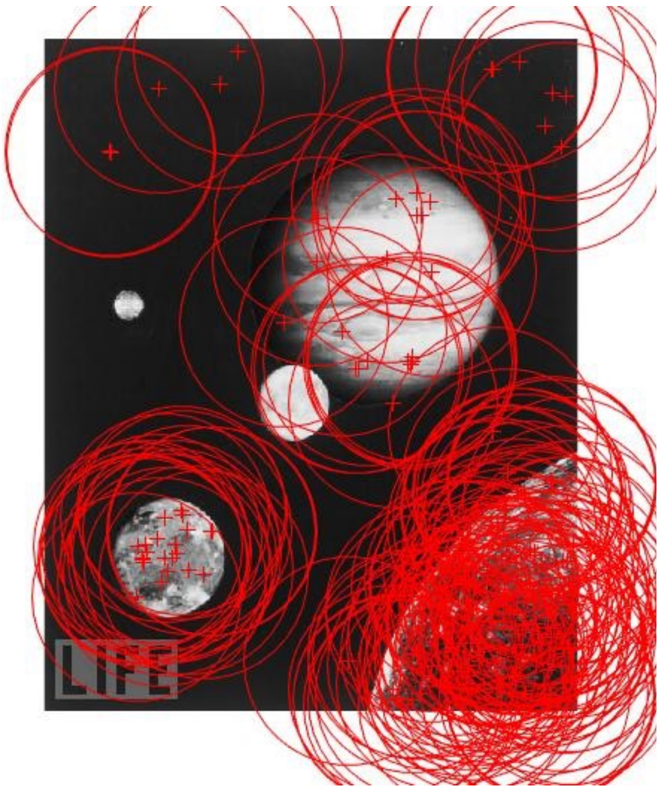
Chau, Vincent
998947424
ECS 189G

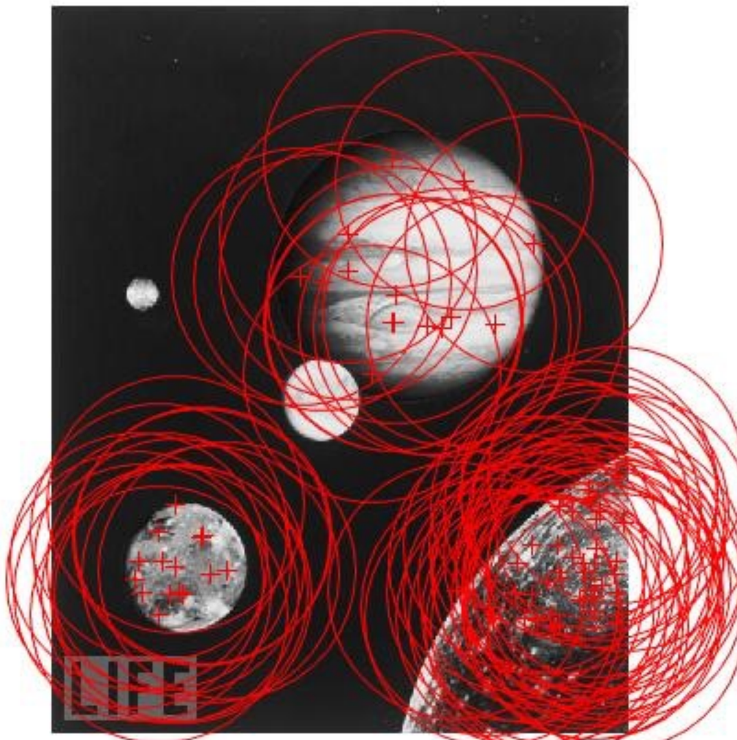
Accum: r = 10, useGradient = false r = 10, useGradient = true





Output: r = 10, useGradient = false r = 10, useGradient = true





Chau, Vincent
998947424
ECS 189G

I choose the radius of 70, which in both cases will search through radius 1:70. The accuracy would vary due to the object's sizes. The outputs on this image would be less accurate compared to the egg images because the circles in the jupiter image have a very great difference in sizes. Each moon/planet is a different size and the planet on the bottom right is cut off. For the first case when useGradient is false, as shown on on the accumulator image, it is a victim to noise. It may have been due to my usage of the canny edge detector and the edge function() and not using an additional method to extract the edges. The noise is seen on the top right and top left where centers are detected on the small cluster of noises in the accumulator. Other than that, the the circles are detected with very inaccuracies due to noise and radius size. For the useGradient = true, the circles detected were much more accurate as there were no noise and no random points detect on the left and right portion. Finding the center would vary which explains the points on the planets because the radius is up to 70 which may produce a better result for one planet over another due to the varying sizes.

C. (Explained in Part B above for both images)

D.

For images such as the jupiter where the circles size significantly vary, we need a method to cluster all points within a certain distance of each other. In the jupiter image when R = 70, many coordinates where found in the bottom right region on the cutoff oversized planet. A method to know that all those points correspond to the same planet would be to take the distance of each point from each other and take the mean (like k-means) and join them all at one point. Apply this method for every point in all the clusters until all points near each other are grouped and the points are merged/averaged. Get the size of the amount of groups then we will have the total amount of planets in the image.

More in depth, this can be done by calculating the collisions of circles and merging the colliding circles into one point. I experimented and tried to implement this in my final solution, but due to debugging and the remaining time constraints for this assignment, I did not complete the implementation. However, this can be possibly done with by calculating colliding points with the formula $(x1 - x2) + (y1 - y2) < (r1 + r2)$ for every circle and its closest neighbor. There would be no need to square the formula since the inequality holds without the squaring.

Formula from: (http://stackoverflow.com/questions/1736734/circle-circle-collision)

For images with more consistent size circles like the egg image, to determine how many circles are present, we just need to get the size of the centers array after the thresholding. In a good case such as r = 10 for the egg image, most of the circles are detected with only one coordinate at its center per circle. To estimate the number we just need the size of the centers array.

Chau, Vincent
998947424
ECS 189G

E.

Extra Credit: I implemented the circle detector to detect circles of any radius since my method searches from radius 1: to the radius of the user input. The test images are the same as the images in Part B.