1 Short answer problems

1. Suppose we form a texture description using textons built from a filter bank of multiple anisotropicderivative of Gaussian filters at two scales and six orientations (as displayed below in Figure 1). Is theresulting representation sensitive to orientation or is it invariant to orientation? Explain why.

   This resulting representation is invariant orientation. Since the filter bank has 6 versions of filter with different angles, it could generate a feature vector which would later be transformed into an orientation-invariant feature space. The reason to do so is that due to difference in lighting conditions and scale and orientation change due to perspective distortion. This method could avoid the influence of the factors mentioned earlier. We could improve its scale-in-variance by including more filters of different sizes.

2. Consider Figure 2 below. Each small square denotes an edge point extracted from an image. Say we are going to use k-means to cluster these points' positions into k=2 groups. That is, we will run k-means where the feature inputs are the (x,y) coordinates of all the small square points. What is likely clustering assignment that would result? Briefly explain your answer.

   Given that $k = 2$, we will have 2 clusters to split the two circles with same center into equal halves. The resulted boundary will be a straight line that satisfies this requirement, i.e. a line through the center points of both circles. The cluster centroids will be between outer circle and the center points.This following graph will show a possible prediction that satisfies this question.

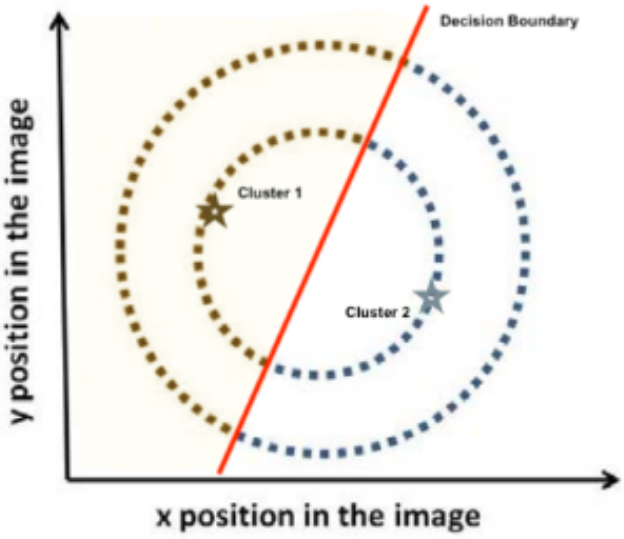


Figure 1: original photo of question 4

3. When using the Hough Transform, we often discretize the parameter space to collect votes in an accumulator array. Alternatively, suppose we maintain a continuous vote space. Which grouping algorithm (among k-means, mean-shift, or graph-cuts) would be appropriate to recover the model parameter hypotheses from the continuous vote space? Briefly describe and explain.

   I think mean-shift would be appropriate to recover the model parameter hypotheses from the continuous vote space, due to the following reasons:

   - For mean-shift, the target feature in original image leads to dense clusters of vote points in Hough space. Therefore the mean-shift search window will move gradually towards the center of most votes.
   - For k-means, we don't know the total patterns in the original image, therefore it's impossible to choose an appropriate K value beforehand. Also the spherical cluster assumption doesn't really work for the original graph.
   - For graph-cuts, it assumes each pixel as a graph node, therefore it's not very applicable to a continuous feature space. Rather than a mode finding, graph cuts is more of a graph-based method.

4. Suppose we have run the connected components algorithm on a binary image, and now have access to the multiple foreground 'blobs' within it. Write pseudocode showing how to group the blobs according to the similarity of their outer boundary shape, into some specified number of groups. Define clearly any variables you introduce.

```
# Question 1.4
# pseudocode

samples = []
temp = []
for blob in blobs:
    edge_point = []
    for pixel in blob:
        if (all neighboring pixels of pixel exist in blob) is false:
            add pixel to edge_point
    max_x= max(column index in edge_point)
    min_x= min(column index in edge_point)
    max_y= max(row index in edge_point)
    min_y= min(row index in edge_point)
    aspect_ratio = max((max_x - min_x)/(max_y-min_y),
        (max_y-min_y)/(max_x - min_x))

    center(r,c) = findcenter(edge_point)
    k = len(edge_point)
    mu = sum(abs(r_k,r_c)-(r,c)): k from 0 to K-1
    delta = sqrt(sqrt(sum(abs(r_k,r_c)-(r,c)) - mu: k from 0 to K-1)
    ratio = mu / delta
    sample.append((aspect_ratio, ratio))
    temp[sample] = blob

clusters = kmeans(samples, k)
for cluster in clusters:
    for sample in clusters:
        sample = blob[search sampleToBold]
```

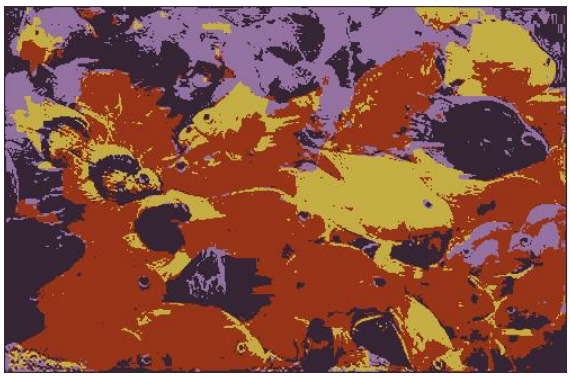
Figure 2: original photo of question 4

2 Problem 2

2.1 Color quantization with k-means

1. Use the functions defined above to quantize the fish.jpg image included in the zip file from part A. On you answer sheet, display the results of quantizing the image in (a) rgb space and (b) hsv space (as described above), print the SSD error between the original RGB image and the results of the(c-d) two quantization methods, and display the (e-f) two different Hue space histograms. Illustrate all of the results with two values of k, a lower value and higher value. Label all plots clearly with titles

   (a) The quantized graph in rgb space



(a) RGB space quantization when k = 4

(b) RGB space quantization when k = 16

Figure 3: (a) RGB space quantization

   (b) The quantized graph in rgb space

(a) HSV space quantization when k = 4

(b) HSV space quantization when k = 16

Figure 4: (b) HSV space quantization

   (c) RGB-Quantization SSD Error, K=4: 563,902,032
   HSV-Quantization SSD Error, K=4: 90,368,497

(d) RGB-Quantization SSD Error, K=16: 183,615,979
HSV-Quantization SSD Error, K=16: 9,614,085

(e) Hue space histogram when k=4

(f)

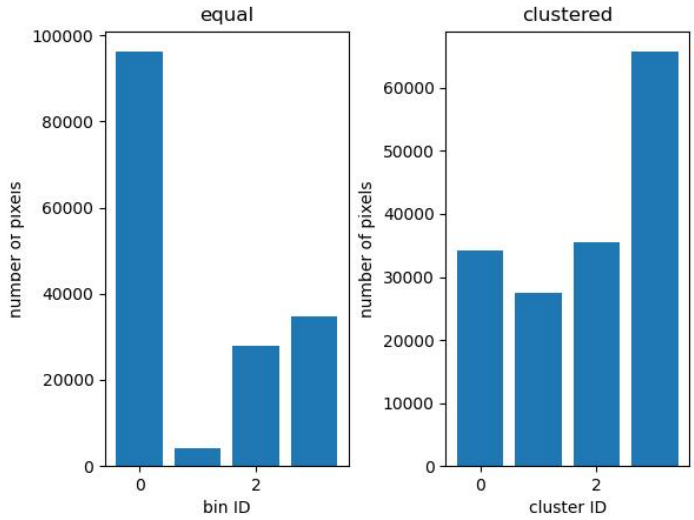


Figure 5: original photo of question 4

(g) Hue space histogram when k=16

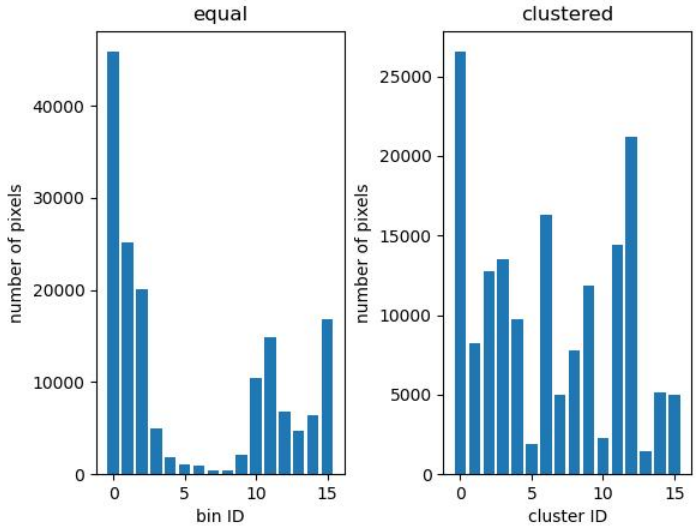


Figure 6: original photo of question 4

2. On your answer sheet, explain all of the results. How do the two forms of histogram differ? How and why do results vary depending on the color space? The value of k? Across different runs?

(a) **How do the two forms of histogram differ?**
These 2 forms of histograms differs in the sizes and ranges of the bins. While `hist_equal`

divides the hue space as k bins of same size. `hist_clustered` divides the hue space according to the clusters. The cluster result is generated with then k given. The bins in `hist_clustered` is likely to have different ranges since usually cluster will not divide points in space evenly.

(b) **How and why do results vary depending on the color space?**
As we can see from the result images, `quantize_rgb` reduce the total number of colors used in the photo to k RGB colors, while `quantize_hsv` leaves the brightness and saturation unchanged when quantizing. This causes `quantize_hsv` results to have more details, and more RGB colors in the results. The k in `quantize_rgb` refers to the number of RGB colors, while the k in `quantize_hsv` refers to the number of hues.

(c) **The value of k**
As we can see from the data of part $(c)(d)$, the SSD error drops as K value increases. The reason is that more cluster centers causes some data points having shorter distance to the closest cluster centers. As a result, the result image will be more similar to the original image in terms of colors and details. The lower the SSD error is, the closer the result image and original image look like.

(d) **Results across different runs**
Currently the results are the same among multiple runs since we're using a fixed random seed. And for the SSD errors, the one for `quantize_hsv` is always lower `quantize_rgb` despite different k values. This is due to the more RGB values in the graph generated by HSV quantization method.

2.2 Circle detection with the Hough Transform

2.2.1 Explain your implementation in concise steps

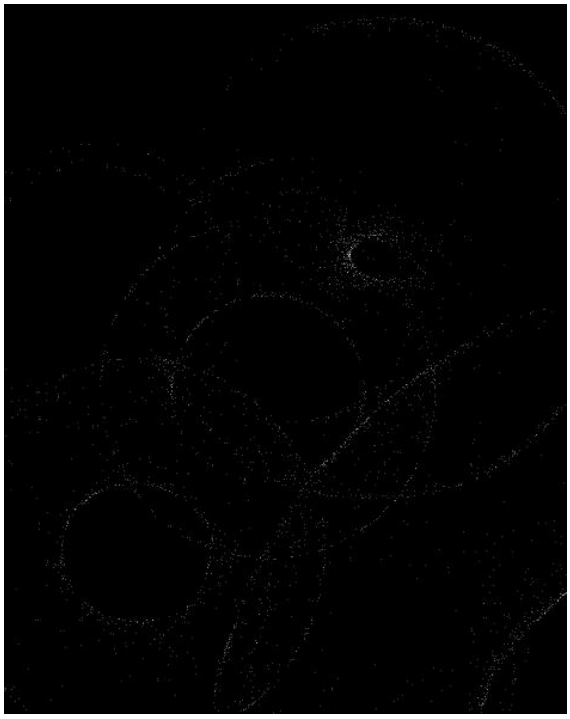
The implementation are as follows:

1. Read the RGB photo. Convert the photo into grayscale.
2. If `use_gradient` is true, use the first derivative filter on x and y direction to create a gradient estimate matrix for all data points. If it's false, skip and continue.
3. Find the edges in the image with Canny Edge Detector from `skimage.feature.canny`. Adjust the sigma value manually to get the optimal results.
4. Build the Hough Space accumulator array. Use a 2-d array with same size as the input image. Then implement the voting space in the following steps.
5. If *use_gradient* is not set for the edge pixel in the binary image, iterate θ from 0 to 2π, calculate the coordinate using formula $(x + r\cos\theta, y + r\sigma\theta)$, where (x, y) is the coordinate of the edge pixel, and r is the radius of circle to detect. Cast each vote to the nearest bin in the accumulator array.
6. Use the gradient value matrix if the gradient is used and get the gradient pertinent.
7. Set the threshold using the maximum voting present in the accumulator array. Save the coordinates larger than threshold value, which are the circle centers we need for the detected circles.
8. If the results are $2 \times N$, transpose it so that it becomes $N \times 2$. Return the centers.

2.2.2 Demonstrate the function applied to the provided images jupiter.jpg and egg.jpg. Display the accumulator arrays obtained by setting *use_gradient* to True and False. In each case, display the images with detected circle(s), labeling the figure with the radius.

For jupiter.jpg, my model produced the following images using different settings.

1. `use_gradient = True`, Radius = 110px

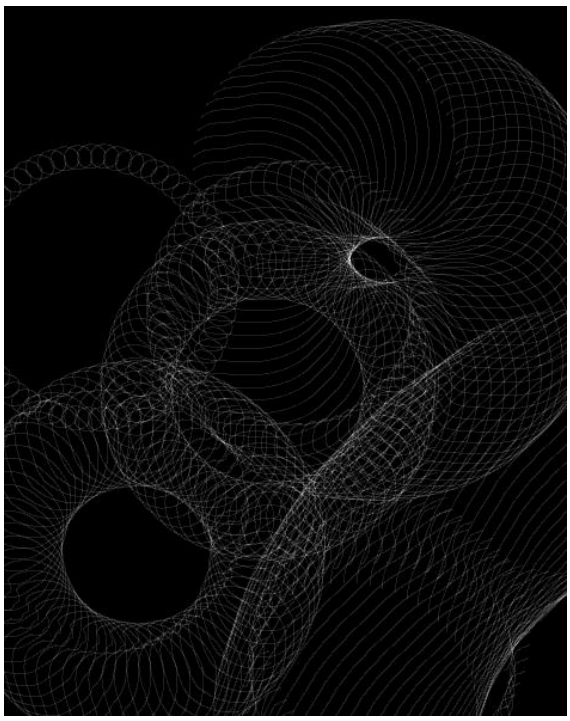


(a) Jupiter.jpg using gradient

(b) Jupiter.jpg circle when using gradient

Figure 7: (b)Results of Jupiter.jpg using gradient

2. `use_gradient = False`, Radius = 110px



(a) Jupiter.jpg not using gradient

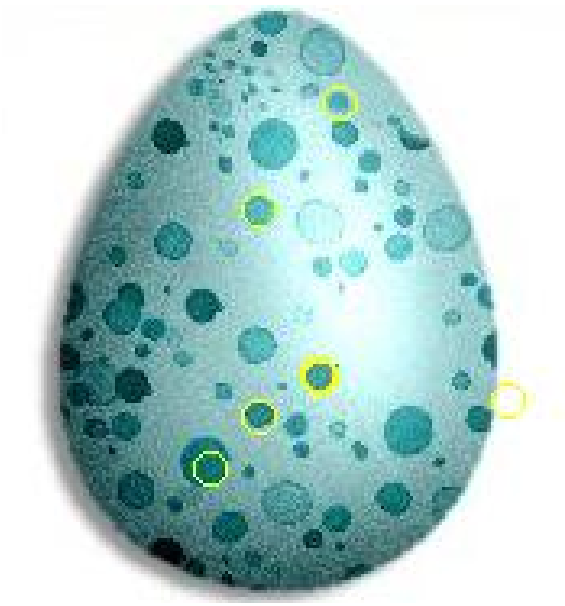
(b) Jupiter.jpg circle without using gradient

Figure 8: (b)Results of Jupiter.jpg not using gradient

For egg.jpg, my model produced the following images using different settings.

1. `use_gradient = True`, Radius = 5px

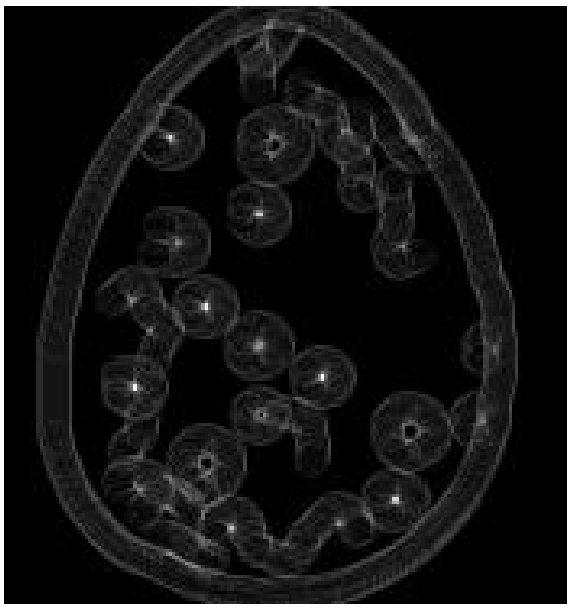
(a) Jupiter.jpg using gradient



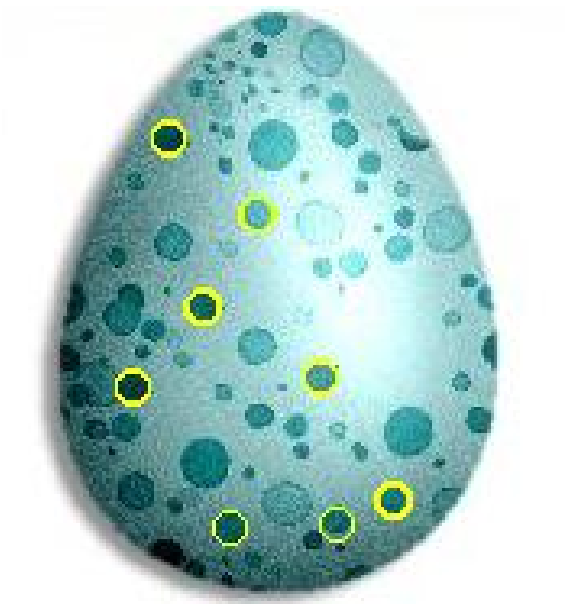
(b) egg.jpg circle when using gradient

Figure 9: (b)Results of egg.jpg using gradient

2. `use_gradient = False`, Radius = 5px



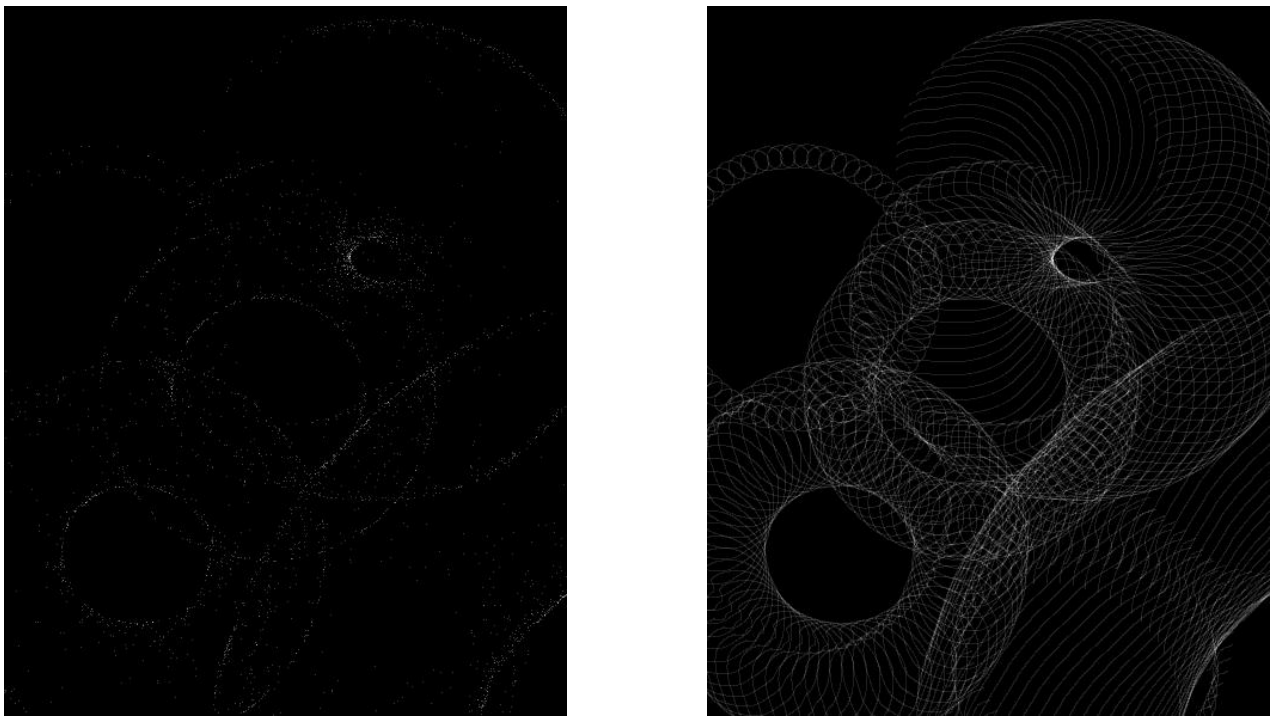
(a) egg.jpg using gradient



(b) egg.jpg circle when using gradient

Figure 10: (b)Results of egg.jpg using gradient

2.2.3 Display and briefly comment on the Hough space accumulator array



(a) Jupiter.jpg using gradient (b) Jupiter.jpg hough without using gradient

Figure 11: (b)Hough space results of Jupiter.jpg

The image I choose is jupiter.jpg. From the 2 images, we can see the one using gradient only shows some clusters and streaks. The reason is that each edge point could only cast 2 votes depending on the estimated gradient direction. The brightest points are the centers detected.

For the result without gradient, there are a lot of net-like structures. This is resulted by the overlapping circular votes. The accumulator array has more votes than the first result image using gradient.

2.2.4 Determine how many circles present

So the photo I use is the egg.jpg. The model detected 8 circles in the Egg.jpg.

I changed the voting threshold so that it could achieve a balance between existing circle can't be detected, and detecting non-existent circles. The threshold value I used in the end is 0.7 for this specific photo.

Another defect of this model is that it will count the same circle multiple times. I'm assuming that the reason might be, it counts circles by centers, and centers with same radius (by default in this setting) and very close get counted twice or more. To tackle this, in addition to the improved threshold value, I also used the connected component algorithm. The centroid of each blob is used as detector centers. So that the same circle won't be counted multiple times.

The result graph below shows the center of the circles detected and the corresponding positions in the egg.jpg

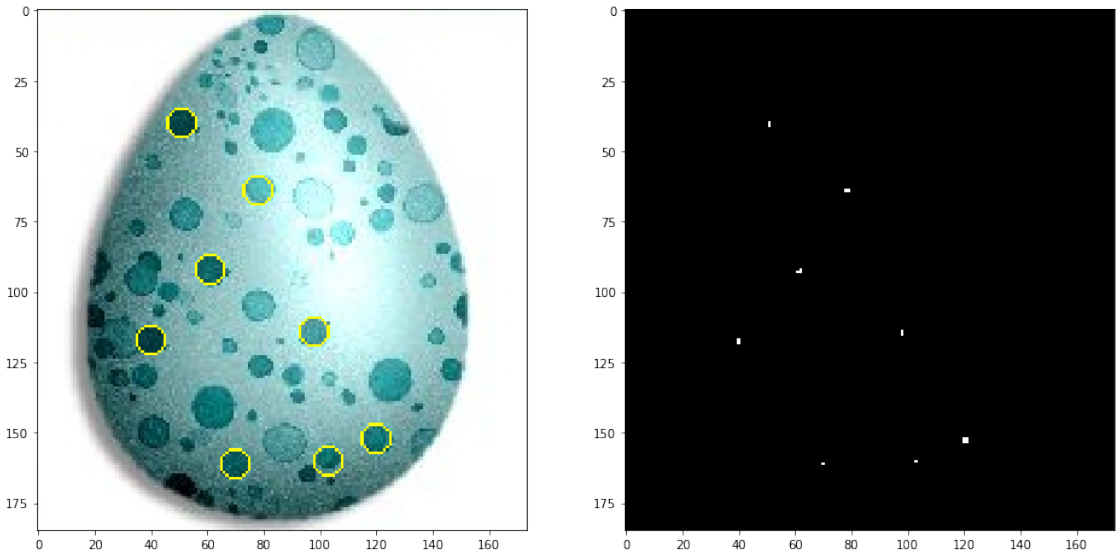
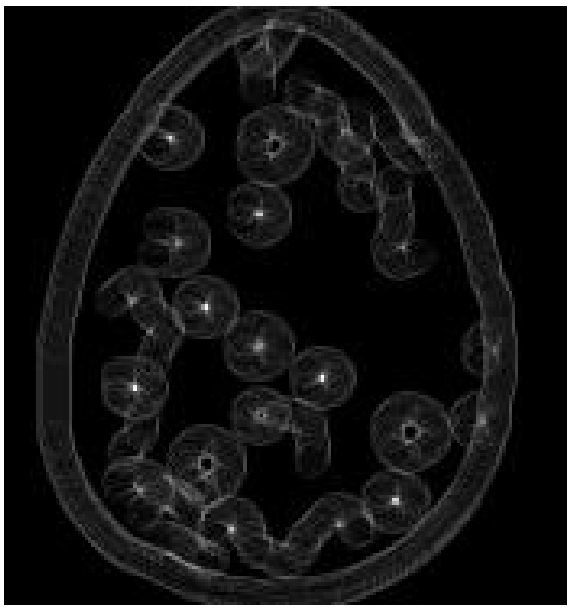


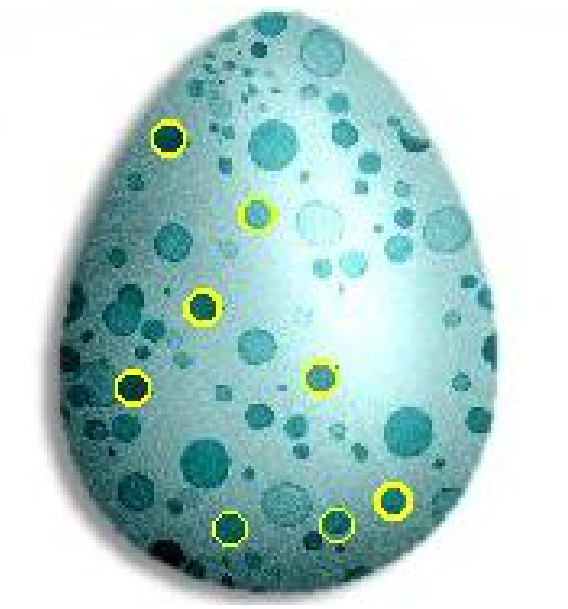
Figure 12: egg.jpg circle reduced

2.2.5 Demonstrate the impact of the vote space

I'm still using the egg.jpg. I divide the bin size by 2 in each dimensions, shrinking the whole to 1/4. The resulting graph I got has more adjacent circles. This might be caused by coarser granularity of the Hough space. 2 votes that supposed to be divided into 2 bins got in the same bin because of the bin size decrease causing a less fine Hough space.

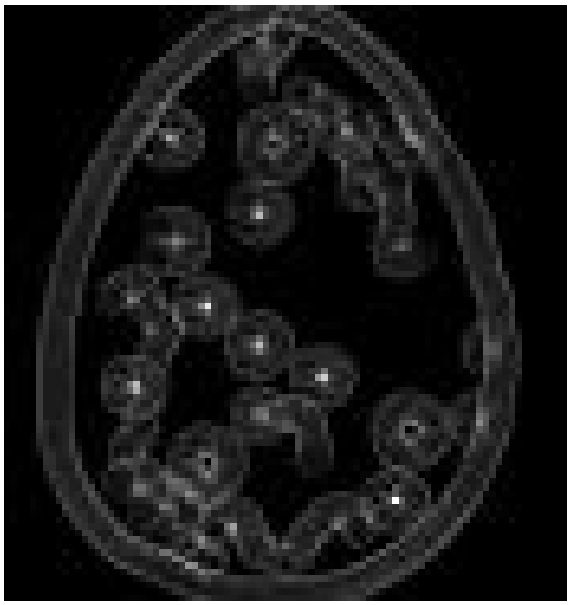


(a) Egg.jpg using gradient

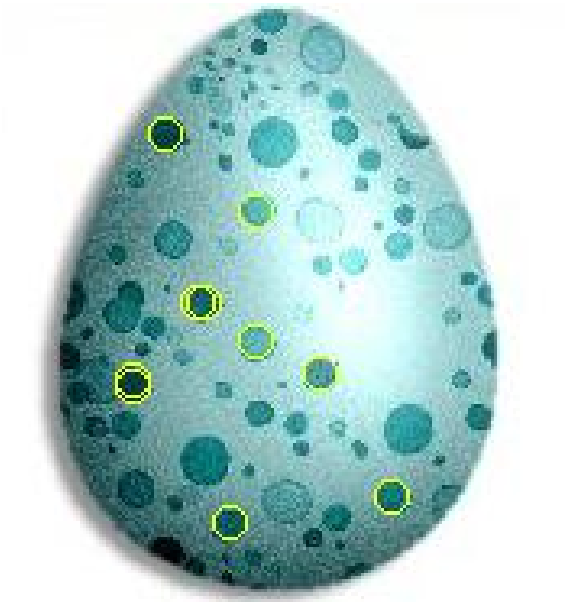


(b) Egg.jpg circle when using gradient

Figure 13: Voting space shape same as image shape



(a) Egg.jpg using gradient



(b) Egg.jpg circle when using gradient

Figure 14: Voting space shape half as image shape

3 Problem 3

Extend your Hough circle detector implementation to detect circles of any radius. Demonstrate the method applied to the test images.

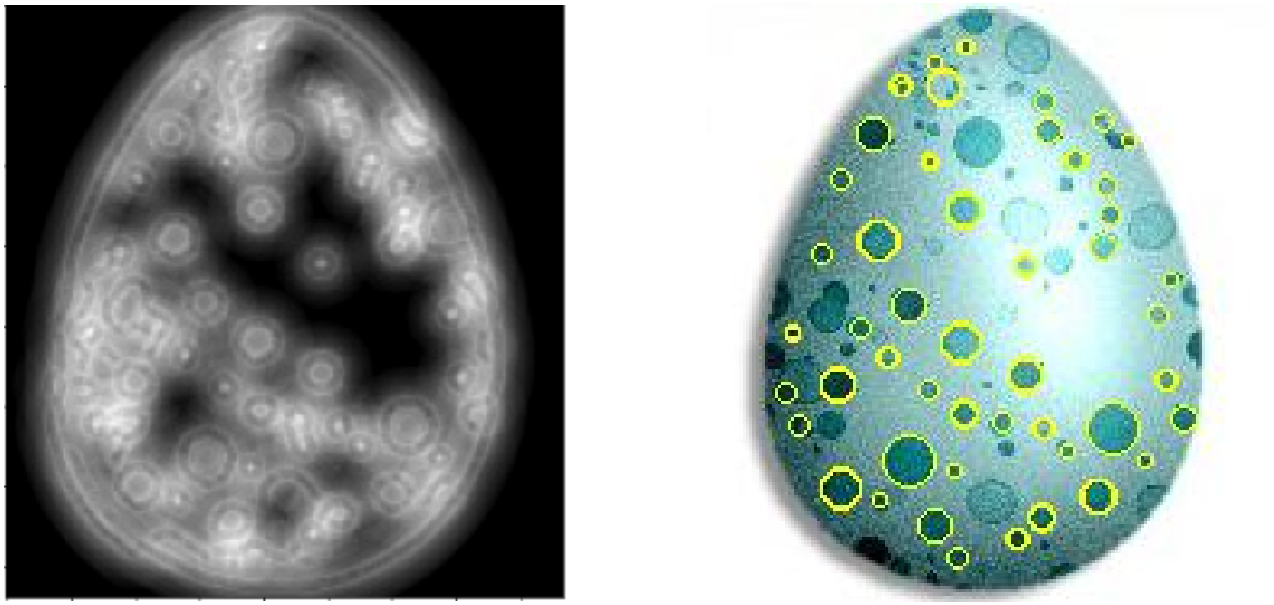


Figure 15: detecting circles of any radius Egg.jpg

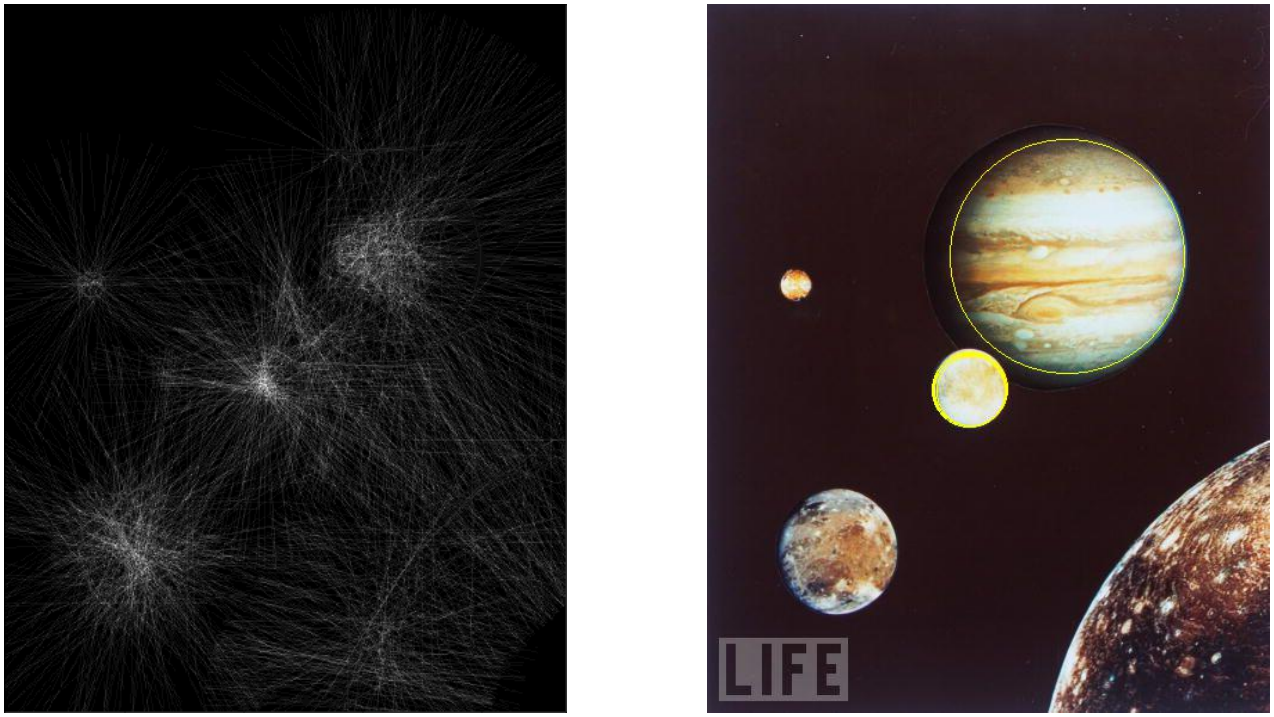


Figure 16: Detecting circles results of Jupiter.jpg using gradient

The algorithm works like this: first I'll parameterize radius in the Hough space, and now it becomes 3D since radius is unknown in addition to x, y. The graphs on the left are those accumulator arrays summed up from votes of different radius in the same center. Then search over a radius range depend on specific graph. In this case we could detect multiple circles.