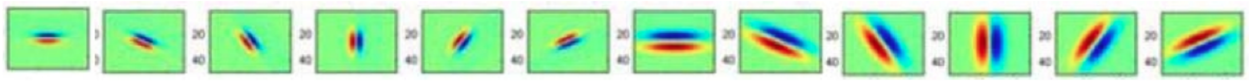
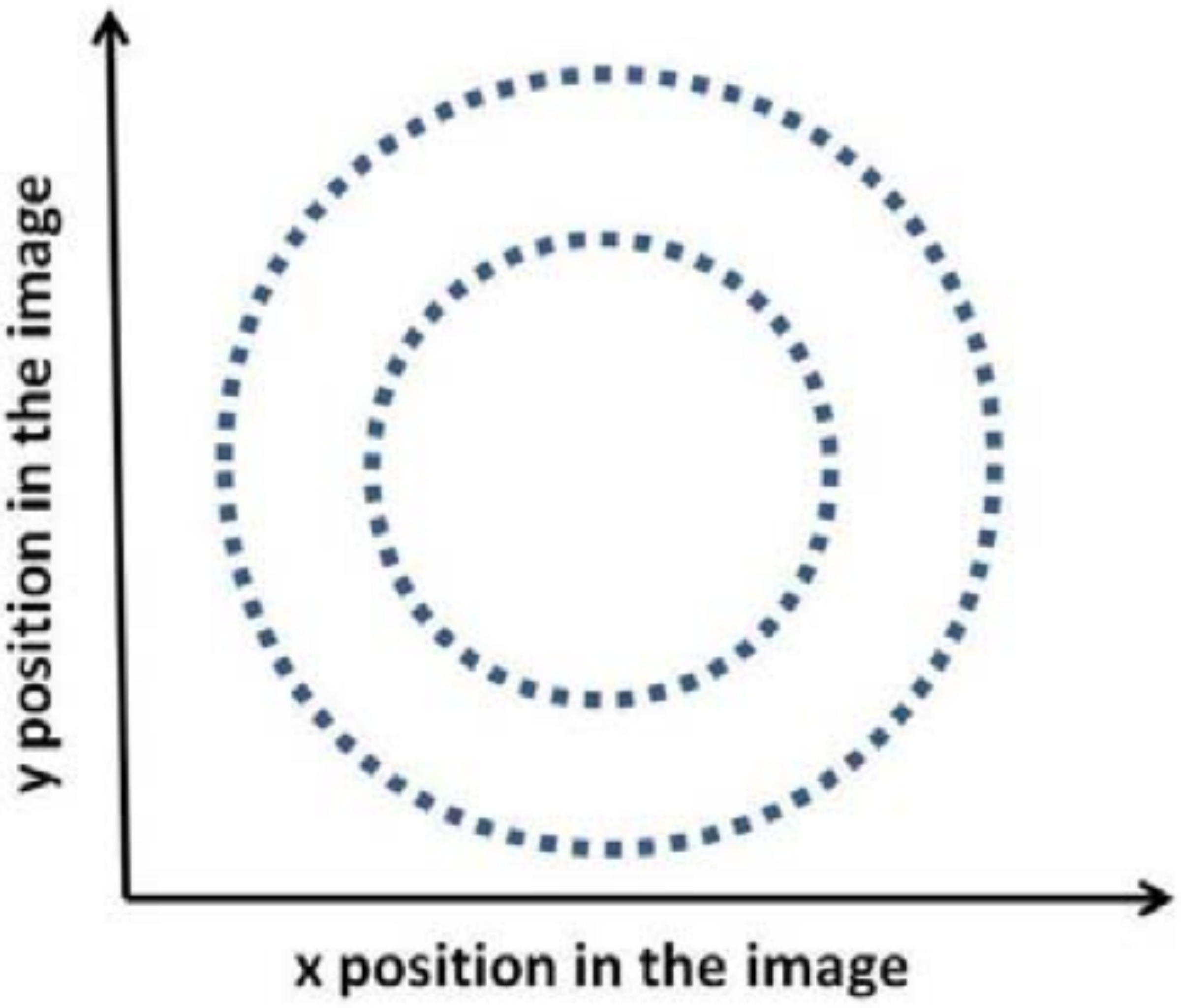
Joshua Reno
CS 4476: Intro to Computer Vision, Fall 2018
PS3

1. Short Answer Problems



1. The resulting representation based on the filter bank above is invariant to orientation since it includes filters that have been rotated across both scales. The purpose of this rotation is to make the texture description invariant to orientation.



2. A likely clustering assignment would split the two circles in equal halves across the center.
3. Mean shift would be appropriate for a continuous voting space since it converges to the center of mass or point of most votes. K-means would find the center of a cluster while graph-cut will will cut along lines which will not lead to the center.
4. Have a point inside a specific blob. For each point on the boundary, compute the displacement between the point inside the blob and the point on the boundary. Store this displacement in a

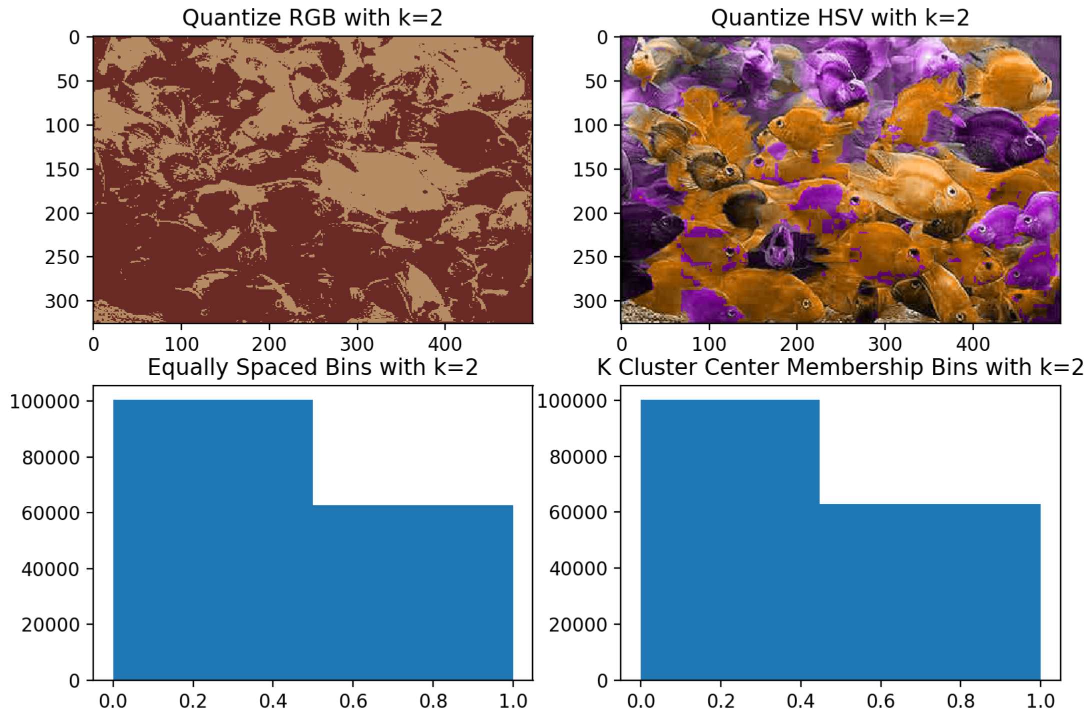
HashMap indexed by gradient orientation. For each point on the edge, use the gradient orientation to index into the HashMap and use the vector to vote for the points. The most voted points will give the center group.

2. Programming problem: content-aware image resizing

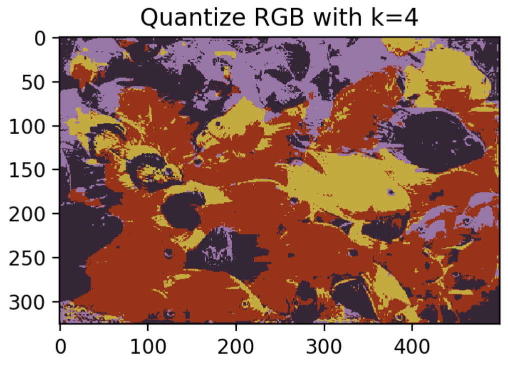
1. Color quantization with k-means

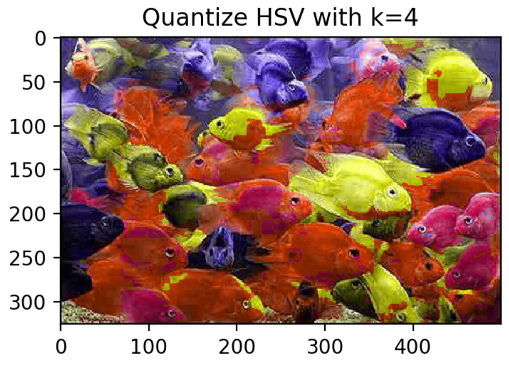
E.

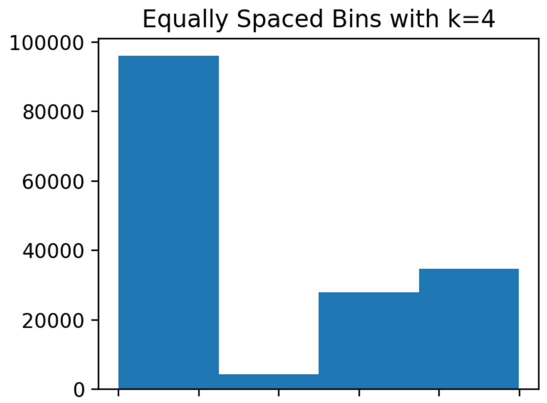
Original image (fish.jpg)

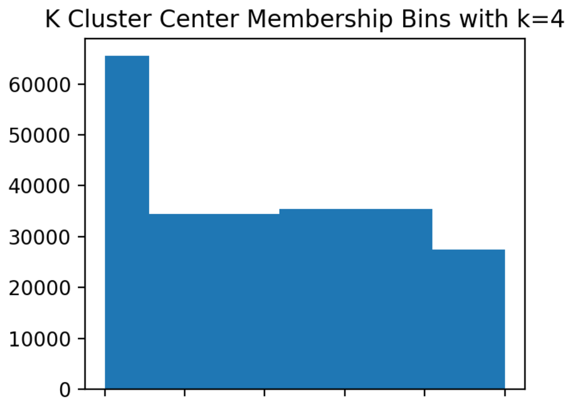


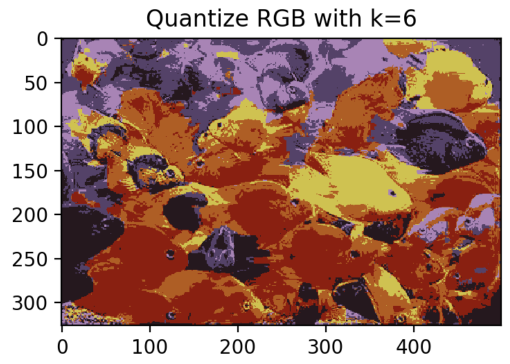
K	RGB error	HSV error		
2	106.18253374233129	43.98912065439673	[100413, 62587]	[100124, 62876]
4	103.26454396728016	32.45492433537832	[96138, 4275, 27912, 34675]	[65573, 34416, 35507, 27504]
6	95.66693456032719	28.93126584867076	[85888, 12342, 2183, 9452, 26737, 26398]	[56097, 32624, 9207, 2792, 35059, 27221]

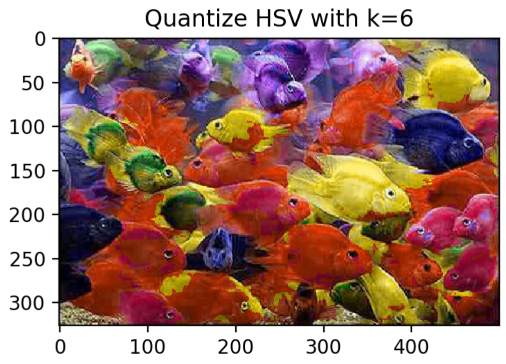


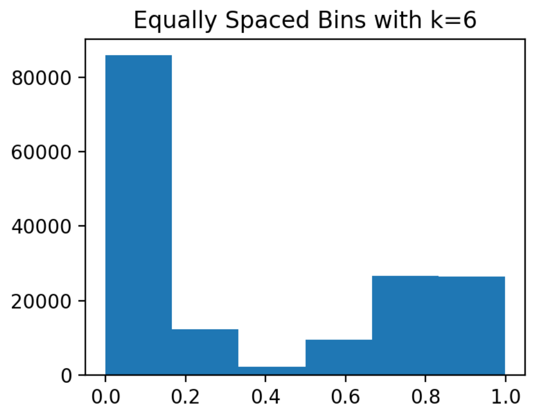


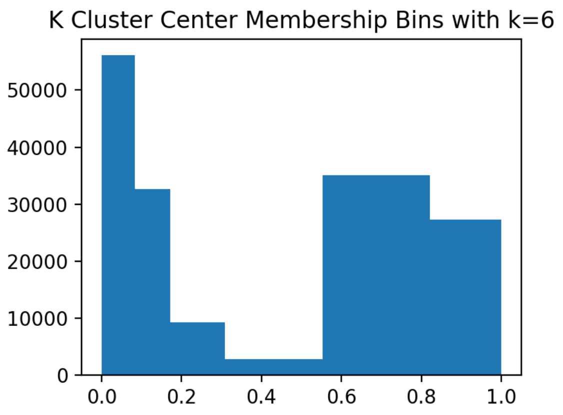








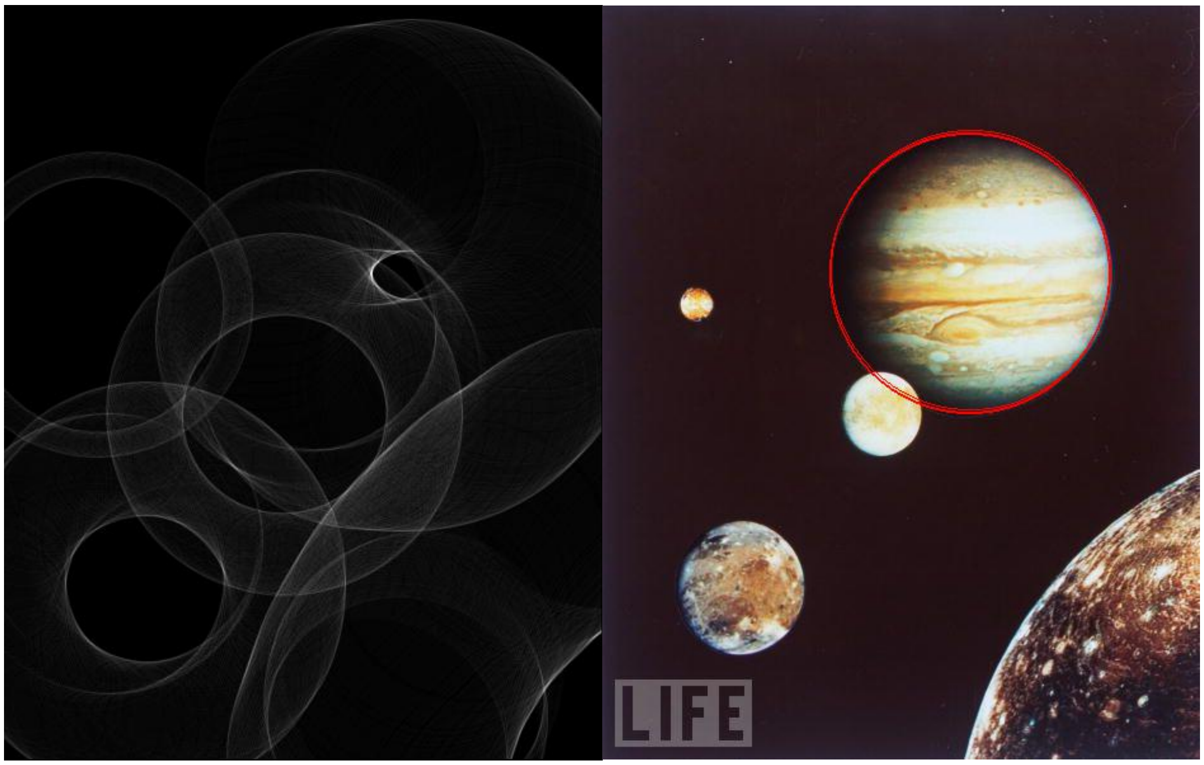


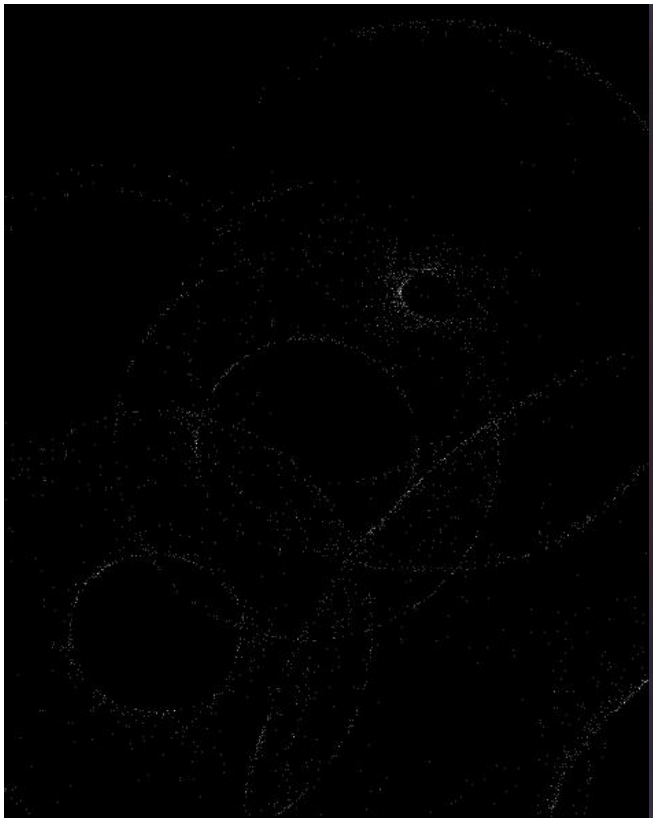


F.
The two histograms are different in the number and size of bins. histEqual divided the hue space into k equal bins while histClustered divided the hue space into the same number of bins based on k-means clustering. The results differed on the basis of the difference between quantizeRGB and quantizeHSV, where in the former the image is quantized to k RGB values while in the latter the hue channel is quantized to k values. Therefore, the brightness and saturation are unaffected. Consequently, when the three values for HSV are converted to RGB, there are more values. Regarding the value of k, this number determines the number of RGB values or number of hues for quantizeRGB and quantizeHSV, respectively. As we increase the value of k, the value of error falls as the quantized image can have more and closer values to the original input image. Also, quantizeRGB has more error values than quantizeHSV since the latter allows the image to have more RGB values than quantizeRGB's limit of k.

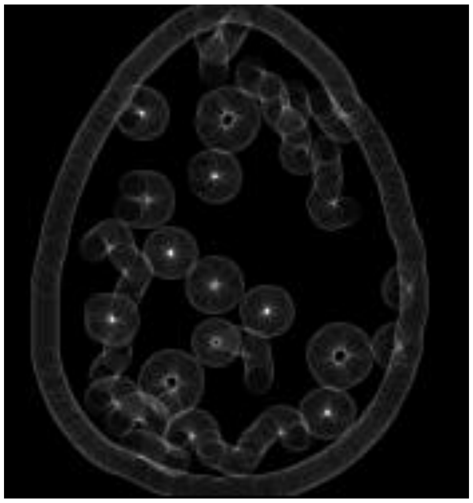
2. Circle detection with the Hough Transform

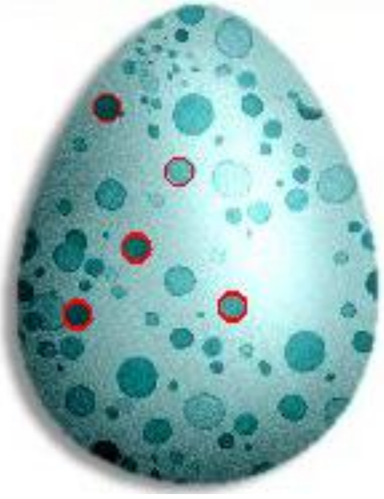
a. For my implementation of the Hough transform, the input image is converted from an RGB image to a binary image and the canny edge detection method (from skimage.feature) is run with a sigma of 6 for jupiter.jpg and 3 for the egg.jpg. For each edge pixel, when useGradient is 0, a and b are calculated using a = x + rcos(theta) and b = y + rsin(theta) for thetas from [0, 2π] and vote for that [b, a] in the accumulator array. This creates a circle of the radius around each edge. If useGradient is 1, the gradient direction at each edge point is used for theta in the equations for a and b given above. Instead of using [0, 2π] and making a circle at each edge point, two points are voted for. After the votes have been cast by each edge pixel, creating the accumulator array or Hough space, the function takes all points above a specified magnitude in the accumulator array or Hough space, choosing which pixel will be the circle's centers. These centers are output as an Nx2 matrix with the 0 column the x locations and the 1 column the y location. These locations are used as the center of the circle to draw a circle of the specified radius.
b. Jupiter (radius of 110) and Egg (radius of 5) with useGradient values of 0 and then 1.

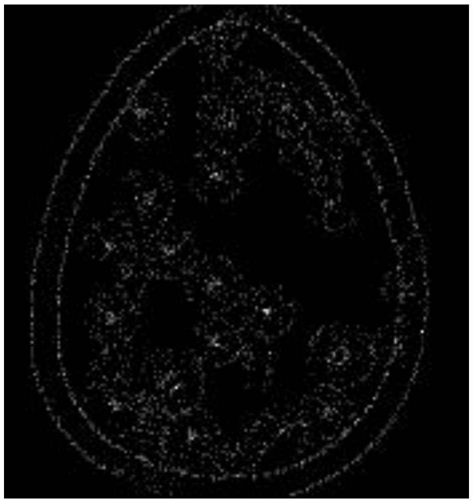


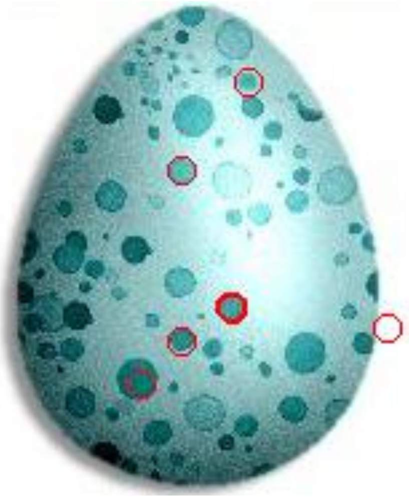


LIFE

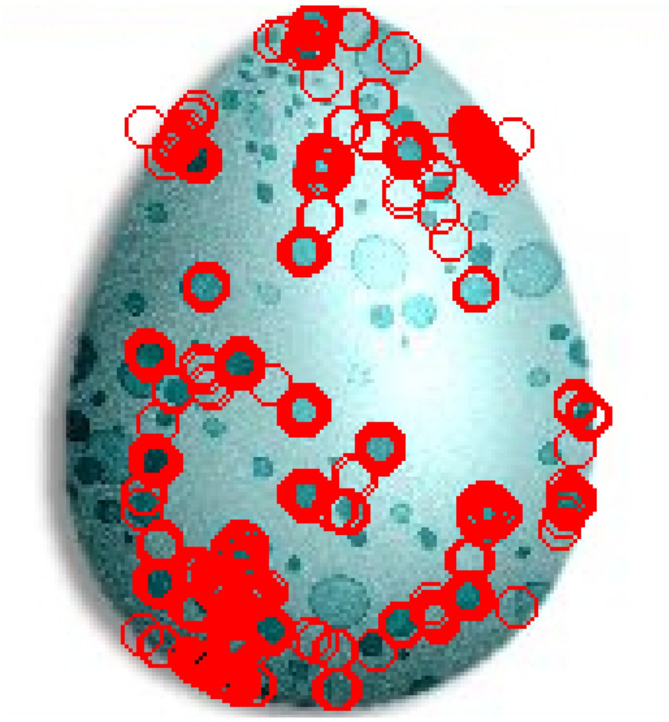
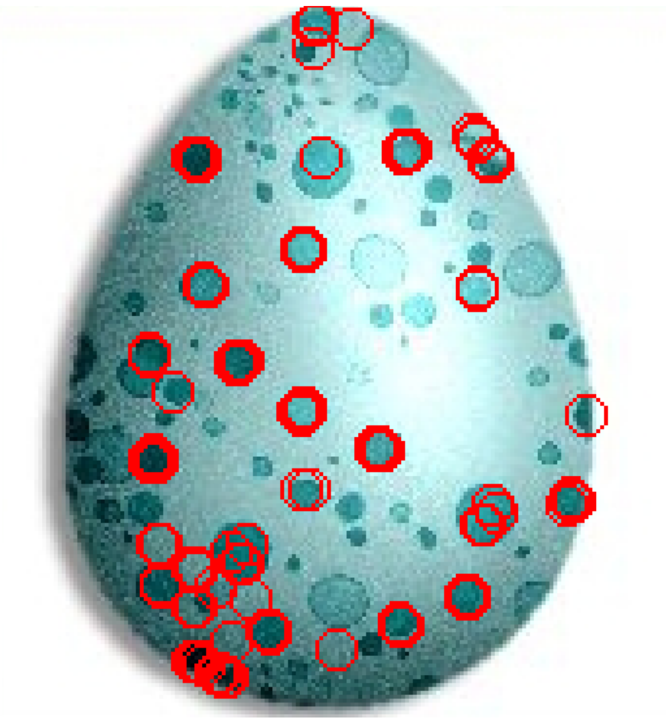
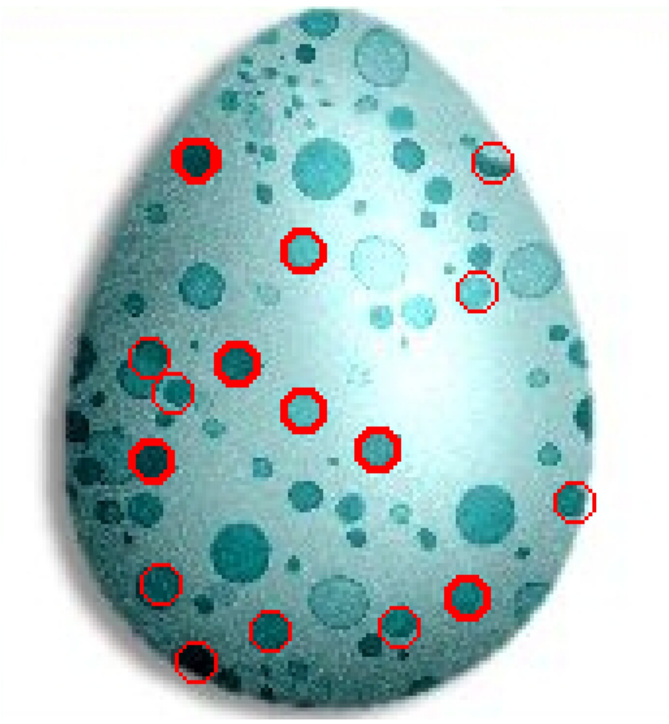
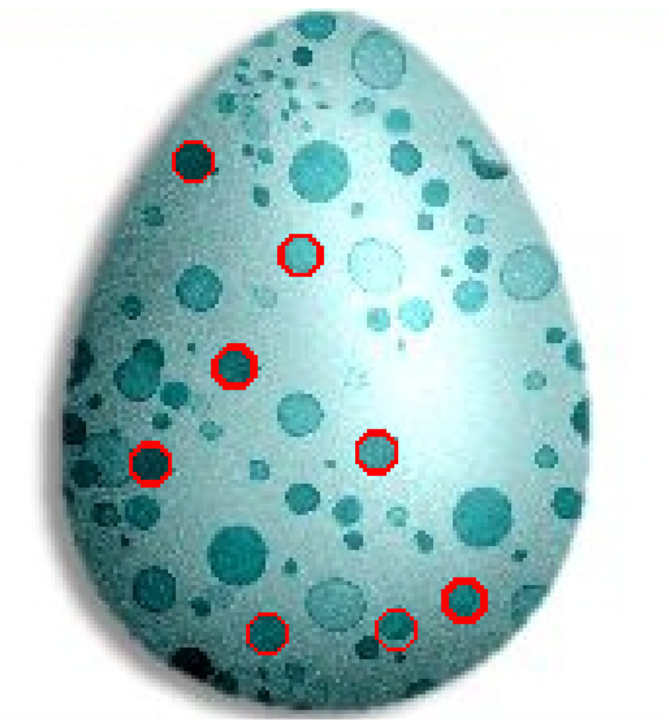


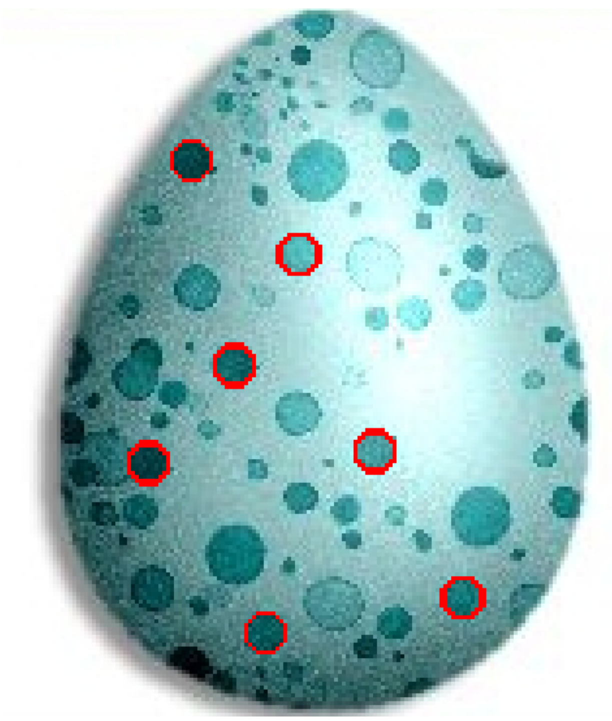


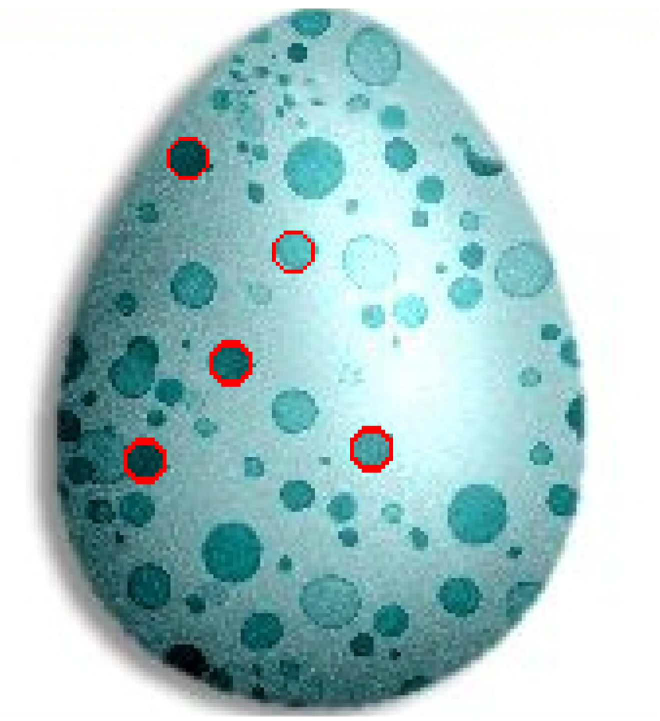


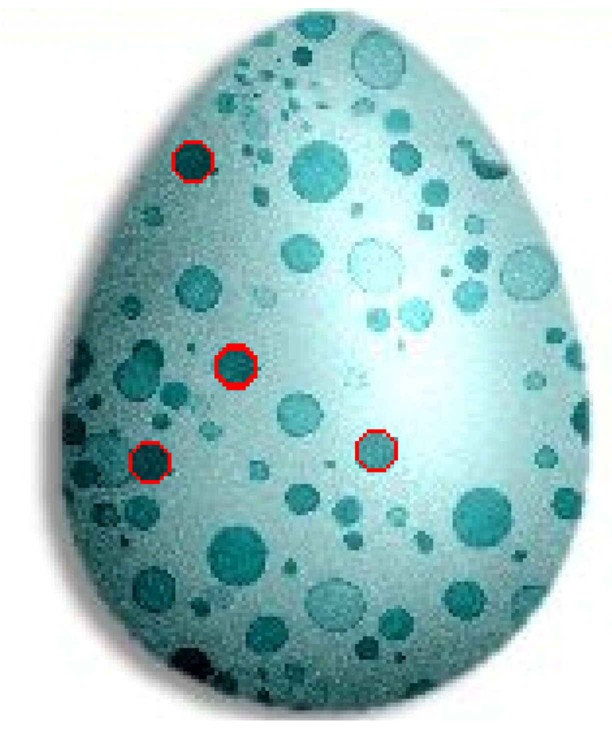


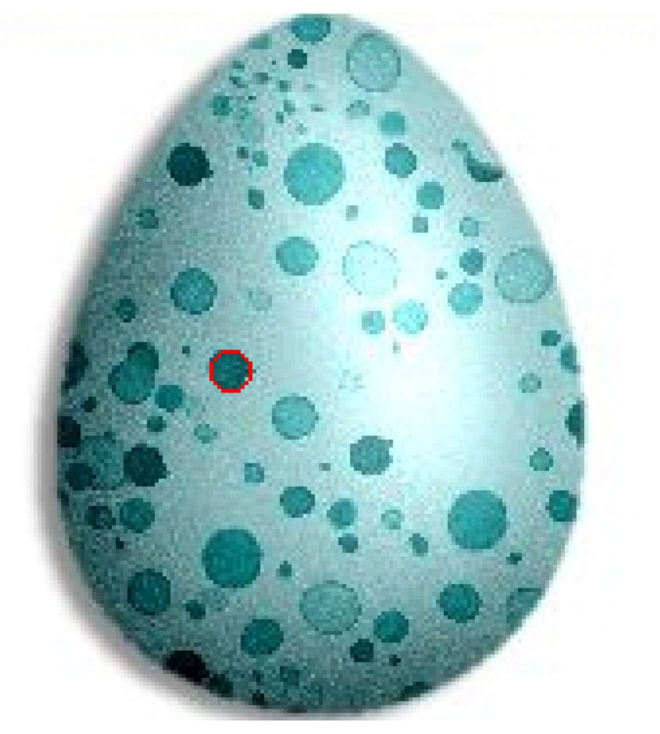
c. Regarding the two Jupiter accumulator arrays, we can clearly see the circles made by each point and the overlapping circles that represent the center of Jupiter in the first image. The second only shows two points for each edge point. This results in a decrease in the number of points but the maximum circle still appears around Jupiter.
d. For this part, I experimented with the number of circles by changing the threshold by allowing circles with values below the maximum value from values of 0.3 to 1.0 times the maximum value. Values from 0.1 to 0.2 were not included because they were messy and 0 is trivial. From left to right downwards we have 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and 1.









e. The following image shows an increase in bin size for the accumulator array which results in the following smoother accumulator array.