L. Carter Price
Computer Vision
PS2 - 9/25/2019

Short answer problems

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

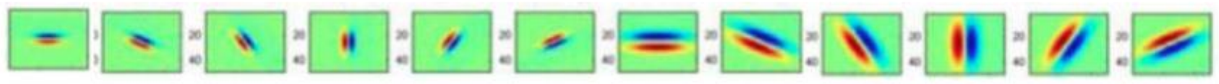
Figure 1: Filter bank

This filter bank is sensitive to orientation because the all filters are not symmetric. For example, the purely horizontal filter shows the blue line on top and the red on the bottom. If the opposite were included in the filter, red on top and blue on the bottom, and this were true for all other filters as well, then this filter bank would be invariant to orientation.

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 a likely clustering assignment that would result? Briefly explain your answer.*

A likely clustering would be two halves. This is because the clustering is based on minimizing the Euclidean distance between the points. The image below shows possible locations of the k-means clustering centers with k=2 groups.

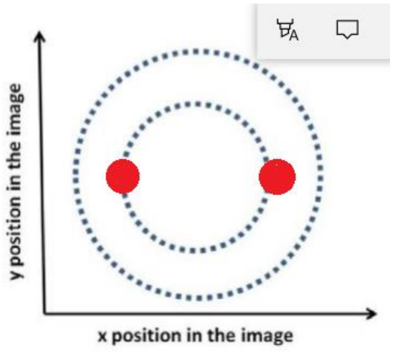

Figure 2: Edge points

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.*

Mean-shift would be appropriate because it goes to the location of the highest density. With the continuous variable there would be a much larger distribution of votes in the Hough space making it challenging for k-means and graph-cuts which must be discretized to function correctly. Also, according to the scikit-learn documentation, continuous or "smooth density of samples" is an ideal situation for Mean Shift:

Mean shift clustering aims to discover "blobs" in a smooth density of samples. It is a centroid-based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region.

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.

```
Find the centroid of a blob
#create a model blob
For each boundary point:
    Compute a vector to the blob centroid
    Store in a table referenced by gradient orientation
#compare to all other blobs
For each blob:
    For each boundary point:
        Calculate gradient
        Match gradient to model table to vote for a reference point
    Choose highest voted reference point
Use K-means clustering (or other clustering method) to group blobs based on reference points
```

Programming Part 1

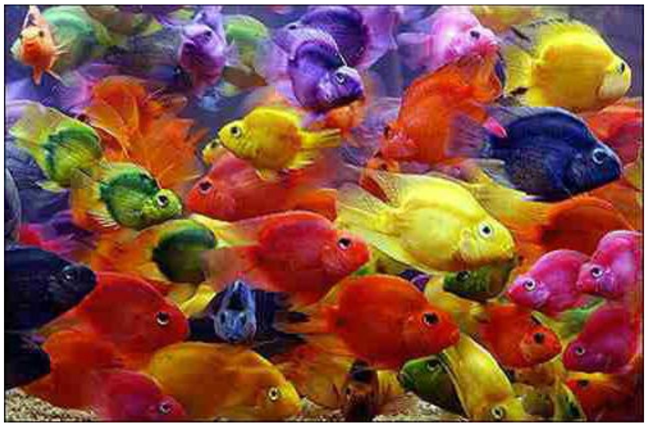

Figure 1: original Picture.

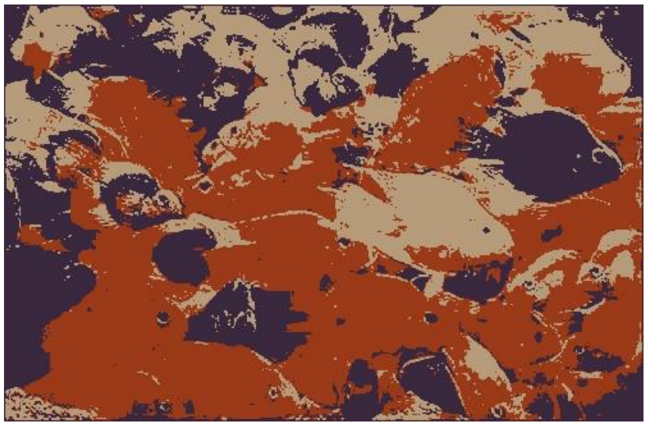

Figure 2: Quantized by RGB, k = 3.

In figure 2, we see the image transformed into 3 distinct color spaces, where the dominate colors found by K-means are orange, blue and tan (which seems to be a fusion yellow and some other minor colors).

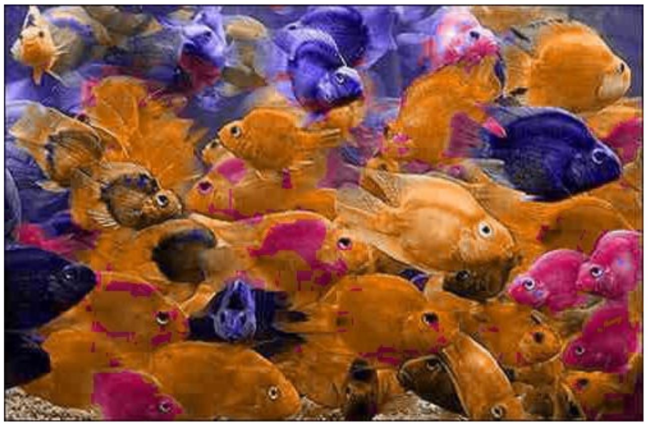

Figure 3: Quantized by Hue, K=3.

The hue quantization in figure 3, does not seem to have as large of a change as the RGB quantization, supported by the sum of squared error being dramatically less for the hue quantization seen in Table 1.

Table 1: SSD values for quantized images, K= 3.

SSD RGB	50,006,325
SSD Hue	17,424,951

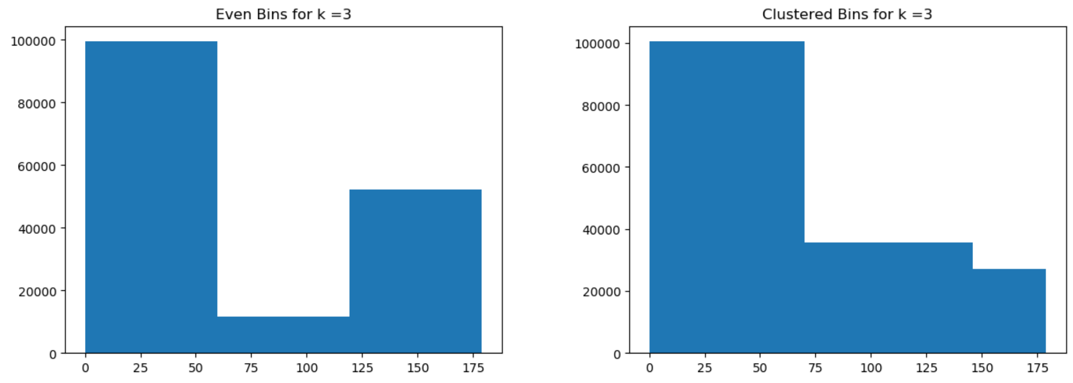

Figure 4: Histograms for K = 3. Evenly divided bins (left) Bins re-sized based on cluster centers. (right)

Bin	**Even**		**Clustered**	
number	**Range**	**number**	**Range**	**number**
1	0-59.67	99409	0-70	100419
2	59.67-119.33	11440	70-146	35557
3	199.33-179	52151	146-179	27024

The histograms seen in figure 4 show how the even interval bins illuminate where there are dramaticaly less hue values in the interval from 59 to 188. The clustered bins show how the highest even bin from the even histogram was split into two bins for the clusterd versions. In the clusterd bins, the transition points are roughly evenly spaced between each of the k-means centers.

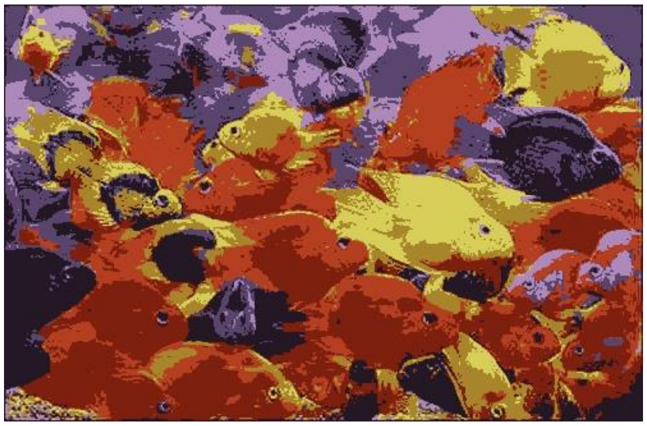

Figure 5: Quantized RGB, k=7.

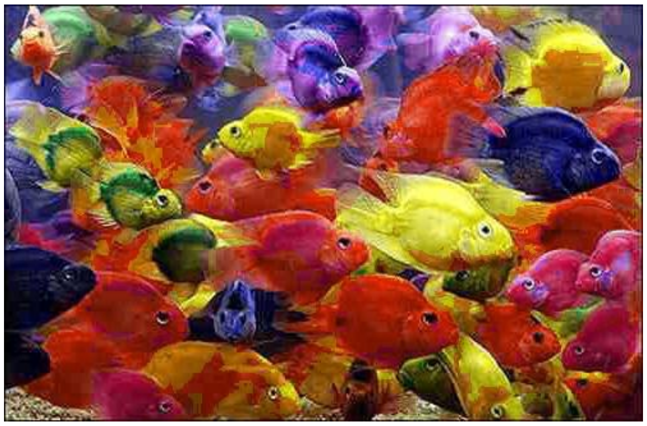

Figure 6: Quantized by Hue, K=7.

Figures 6 shows a transformation into 7 different colors, the 3 colors from the k=3 clustering seem to have been broken down even further though we see that orange and blue still dominate. It is difficult to distinguish the differences between the original image and figure 6; quantizing the hues into 7 clusters does not seem to make many dramatic changes visually though we know there is some detail lost based on the SSD.

When comparing table 1 and 2, one can see the splitting into a greater number of clusters reduces the SSD. This is true for both the RGB and hue quantizations.

Table 2: SSD values for quantized images, K= 7

SSD RGB	46,477,012
SSD hue	12,562,722

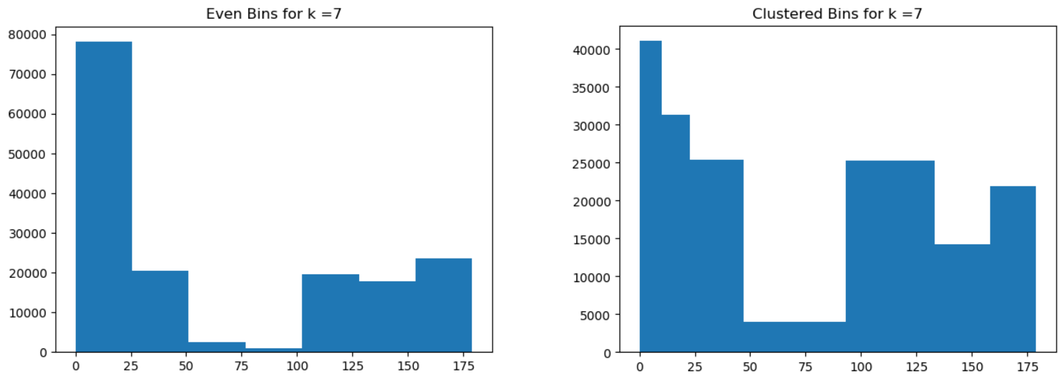

Figure 7: Histograms for K = 7. Evenly divided bins (left) Bins re-sized based on cluster centers. (right)

Bin number	**Even**		**Clustered**	
	End of range	**number**	**End of range**	**number**
1	25.57	78157	10	41030
2	51.14	20428	23	31253
3	76.71	2486	47	25395
4	102.29	1003	93	3995
5	127.86	19585	133	25241
6	153.43	17766	158	14164
7	179.00	23575	179	21922

How do the two forms of histogram differ?

The evenly split histogram generally has a larger difference in the number of values in the highest bin vs the lowest bin. The clustered bins change the bin size to group the different hues together based on their closest neighboring hues. However, they had a generally similar shape.

How and why do results vary depending on the color space?

The RGB color space is quantizing based on all 3 image channels while the hue quantization is only based on the single hue channel. Also, when comparing the color spaces, the hue only dictates the primary color while the amount of white or black in each is controlled by the saturation and value respectively. So, when quantized by hue, if two colors where in a similar color family, lets say red, then the saturation could control the brightness of the color and the pinks and red distinction would still show through in the image. Therefore, we saw less dramatic changes when quantizing hue compared to RGB.

The value of k?

The differences where pretty dramatic based on the k value of clusters. The higher the value of K the more clusters the color space could be broken into. With increasing K value, the image transformed less and less. As k increases to the size of the input image, it returns to the initial image with virtually no changes due to clustering.

Across different runs?

There were slight variations between the different runs based on the random initialization of the cluster centers when doing clustering. However, it was extremely challenging to notice these differences in most cases.

Programming Part 2

a. Implementation description

1. Change image to grayscale
2. Blur the image (filter size jupiter -11 egg-5
3. Run a Canny edge detector over the blurred image.
4. If Gradient enabled, get the sobel gradient in X, Y then use arctan2 to get gradient direction.
5. Initialize the accumulator matrix to all zeros.
6. For all pixels in the image, if they are over 100 then start adding votes to the accumulator.
7. If Gradient enabled, then the new vote is the current pixel location + and – the specified radius times the sin or cos of the gradient direction from step 4 (sin for y, cos for x).
8. If the gradient is off, then loop through angles from 0 to 180 degrees by 2 degrees. Add all these points to the accumulator.
9. Take the top 30% of all votes.
10. Run Mean-shift on the remaining votes.
11. The resulting centers are resulting circle centers.

b.

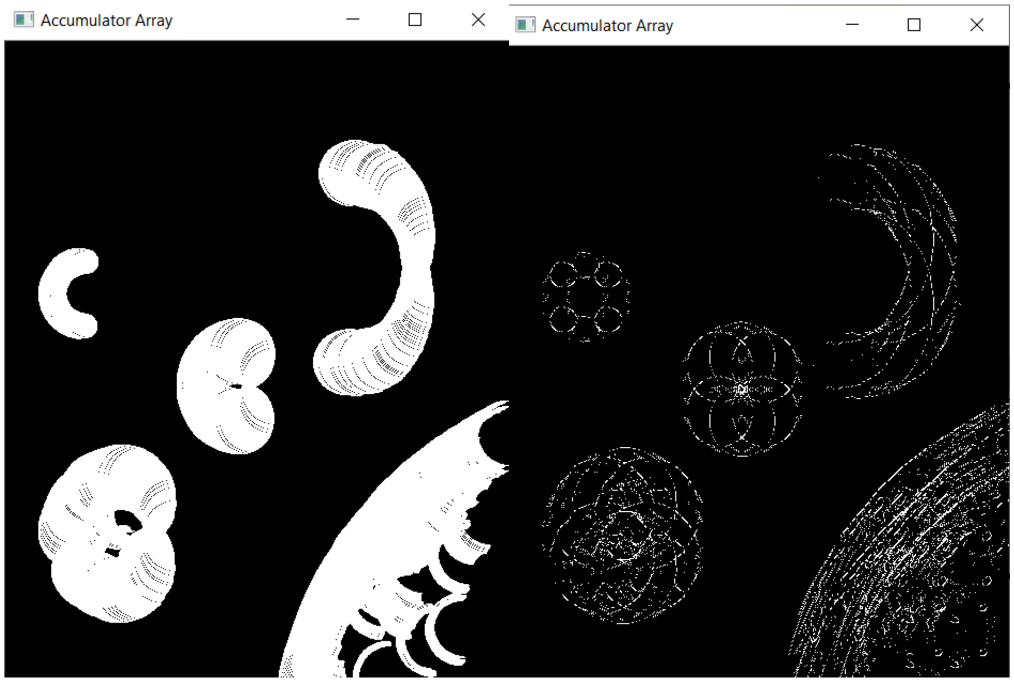

Figure 8: Jupiter image accumulators, radius 30. No Gradient (left) gradient(right)

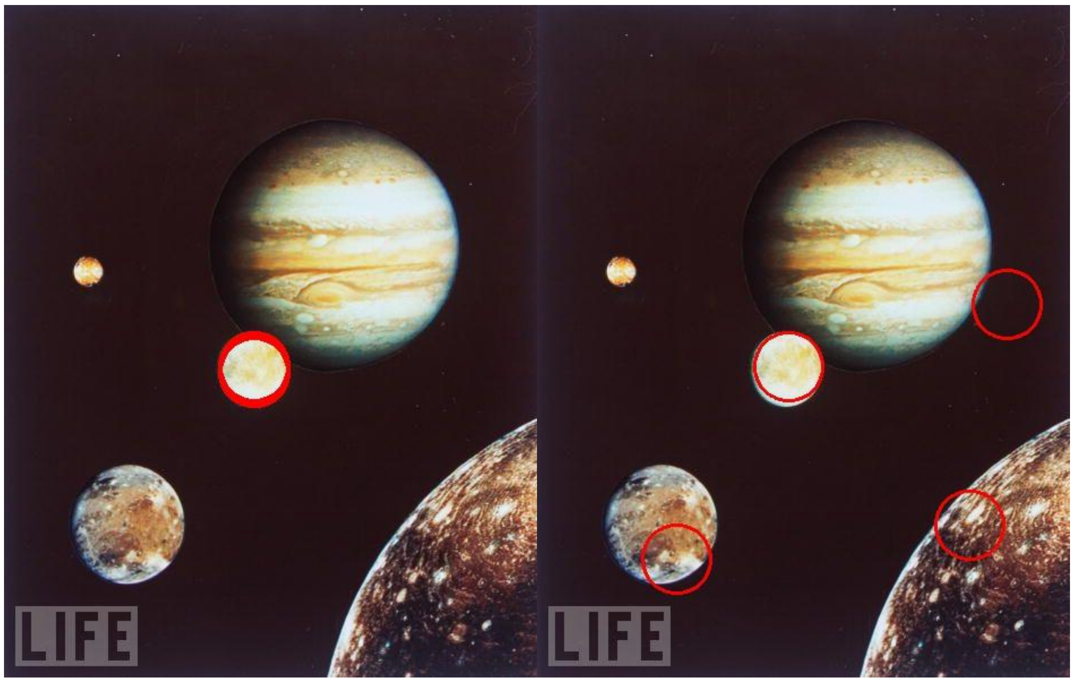

Figure 9: Jupiter radius 30. No gradient (left) gradient (right)

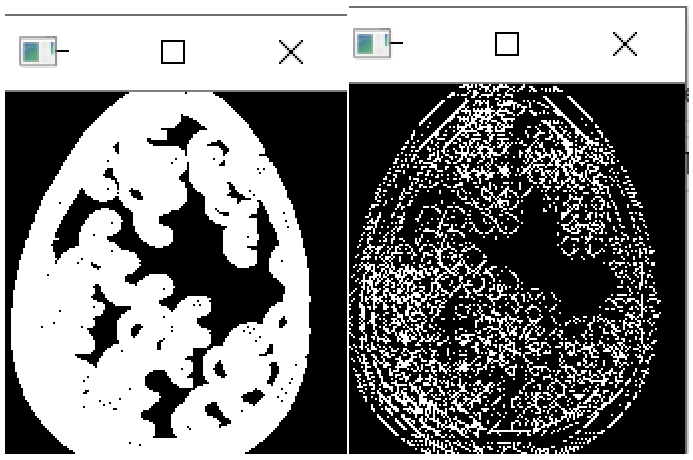

Figure 10: egg image accumulators, radius 8, No Gradient (left) Gradient (right)

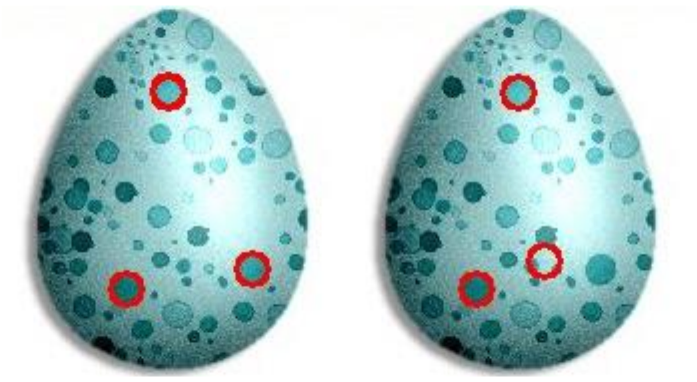

Figure 11: Eggs, radius 8. No Gradient (left) Gradient (right)

(c) As can be seen from Figures 8 and 9 as well as 10 and 11, the gradient vs no gradient has a major impact upon the accumulator array and the final-results. When iterating over many discrete values for theta at each edge location, the accumulator array is much denser. For Jupiter seen in figure 8, there was a point that had over 48 votes while the maximum value in the gradient accumulator array was only 7. The sparse nature of the gradient version allows for noise to have a greater impact and sneak into the top results. The method I used was to take the top 30% of all points in the accumulator array and then apply mean-shift to those to determine the final circle location. Using this method, the noise usually snuck into my results as can be seen in figure 9. However, there may be an improved methodology that would eliminate this problem.

d.

I started off by simple taking a percentage threshold of the highest points in the accumulator. However, this method was generally unreliable for keeping the true circles and rejecting the false positives. Next, attempted K-means, but this did not really make sense because it requires you to specify the number of clusters and essentially you preprogram the number of circles. Lastly, I used mean-shift, and ultimately settled on a combination of mean shift and thresholding the accumulator array in post processing. It did not completely remove all false-positives, but seemed to be a significant improvement on previous methods.

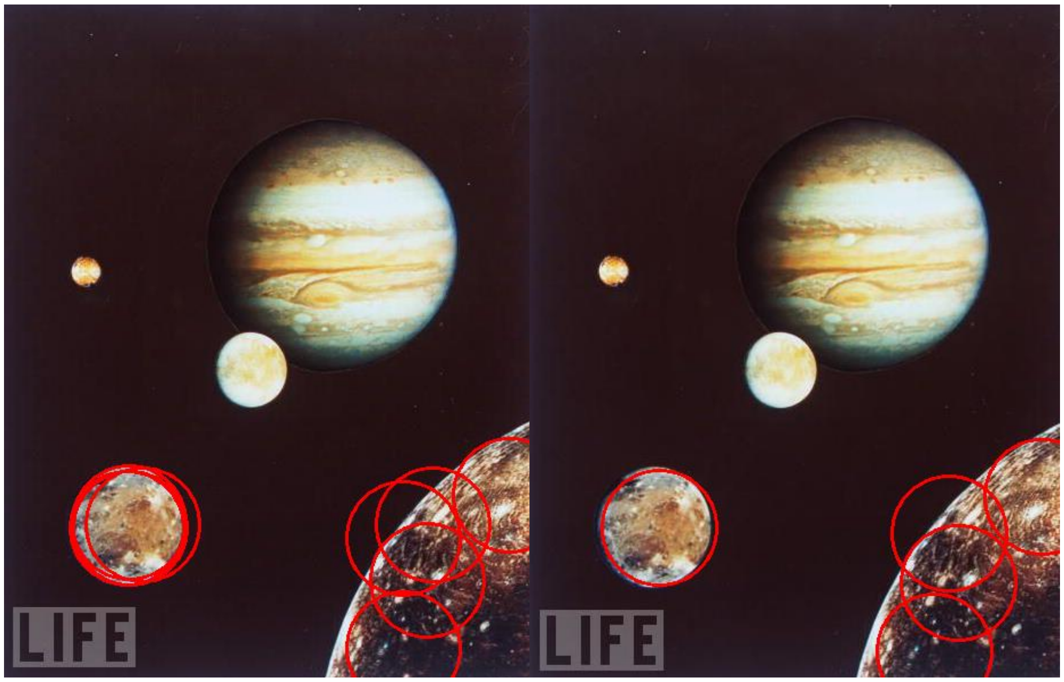

Figure 12: Jupiter, Radius 50, useGradient = 1, No mean-shift enabled (left) Mean-shift(right). Shows how mean-shift used as a post processing technique can reduce the number of circles and false positives. The body in the bottom left hand corner of the image has many repetitive circles around it. However, when mean -shift is used, the repetitive circles are eliminated as seen in the right image.

e)

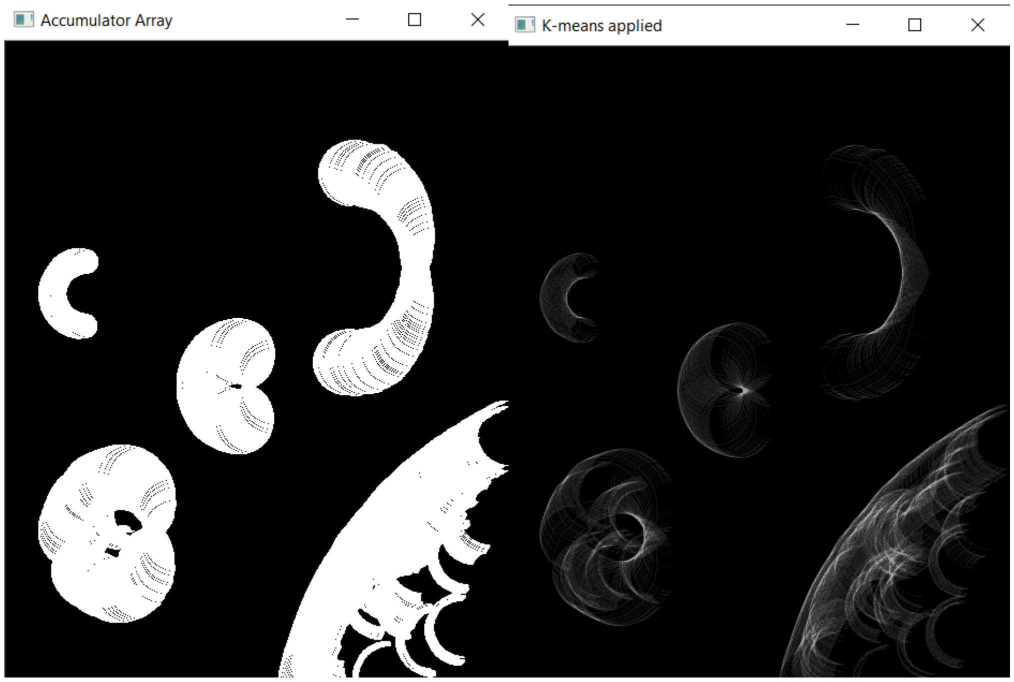

Figure 13: Jupiter images. radius = 30. Accumulator array (left). K-means K = 8 applied to the accumulator. Shows that quantizing the accumulator space more greatly reveals the peak vote point towards the center of the image.

A few additional images for reference

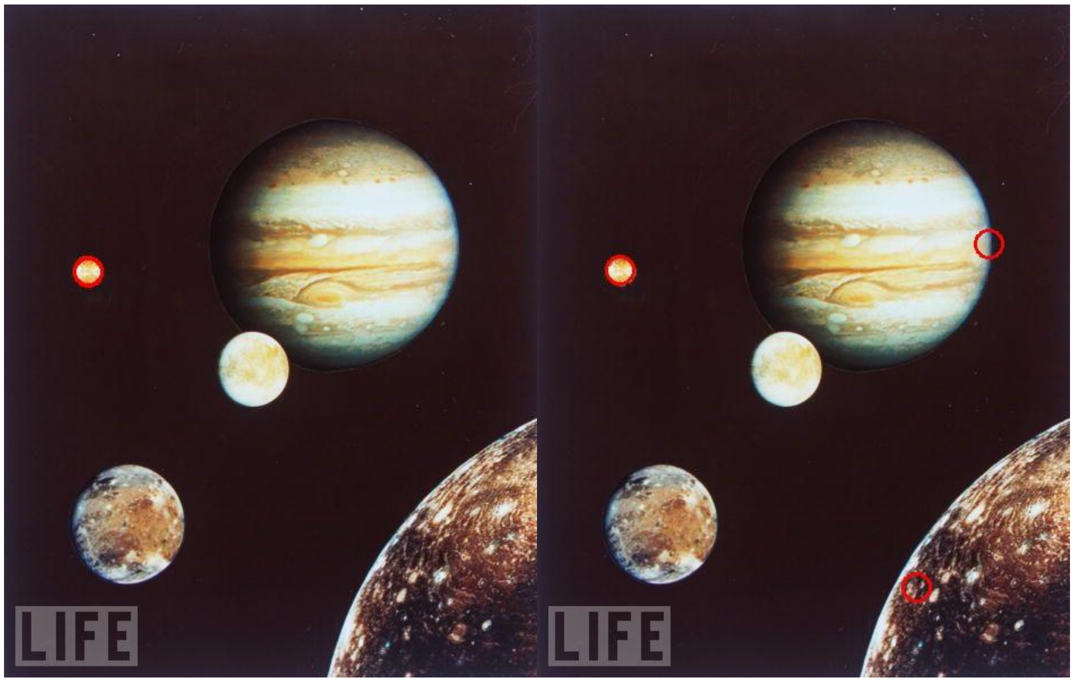

Figure 14: Hough circle, radius 12, No gradient(left) Gradient (right)

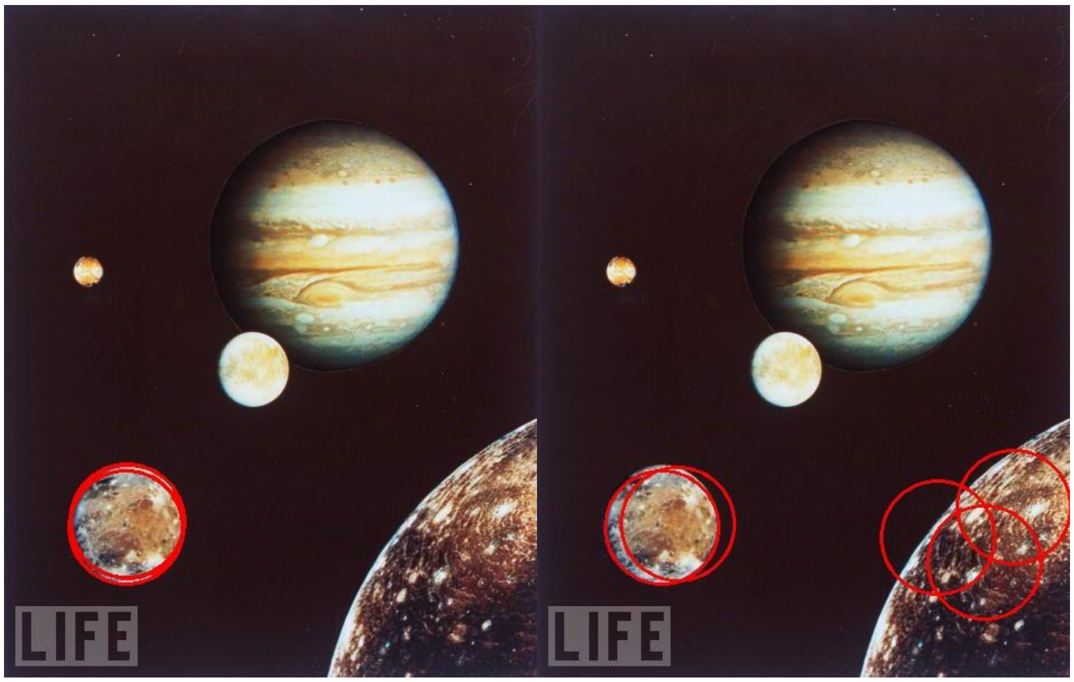

Figure 15: Radius 50, No gradient(left) gradient(right)

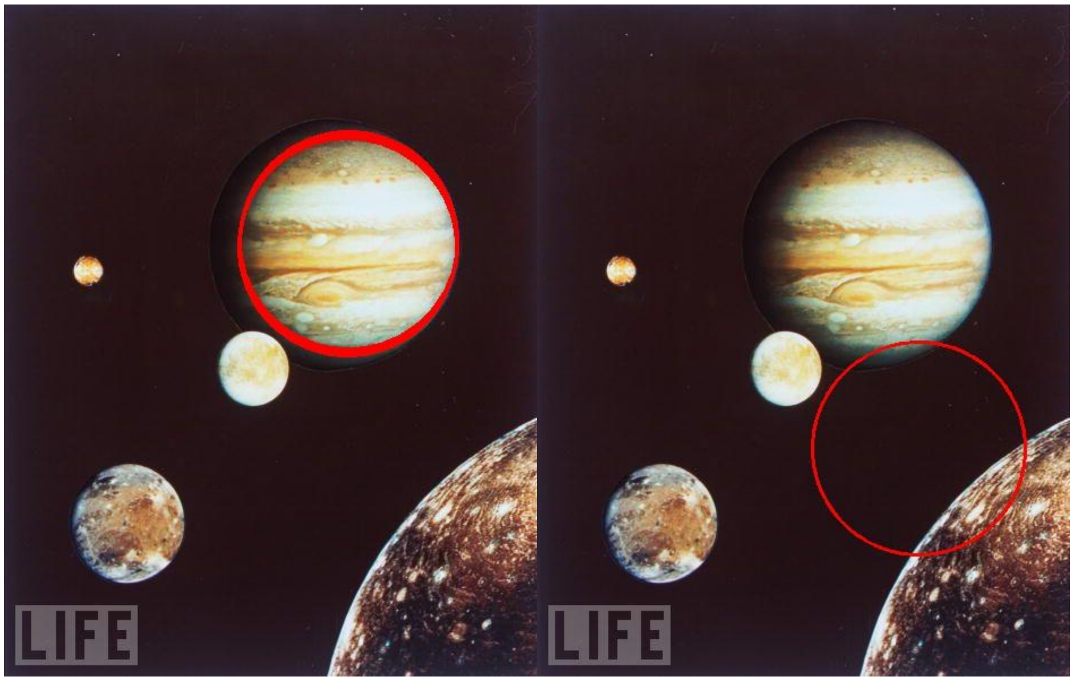

Figure 16: Radius 96, NO gradient(left) gradient (right)

Extra Credit

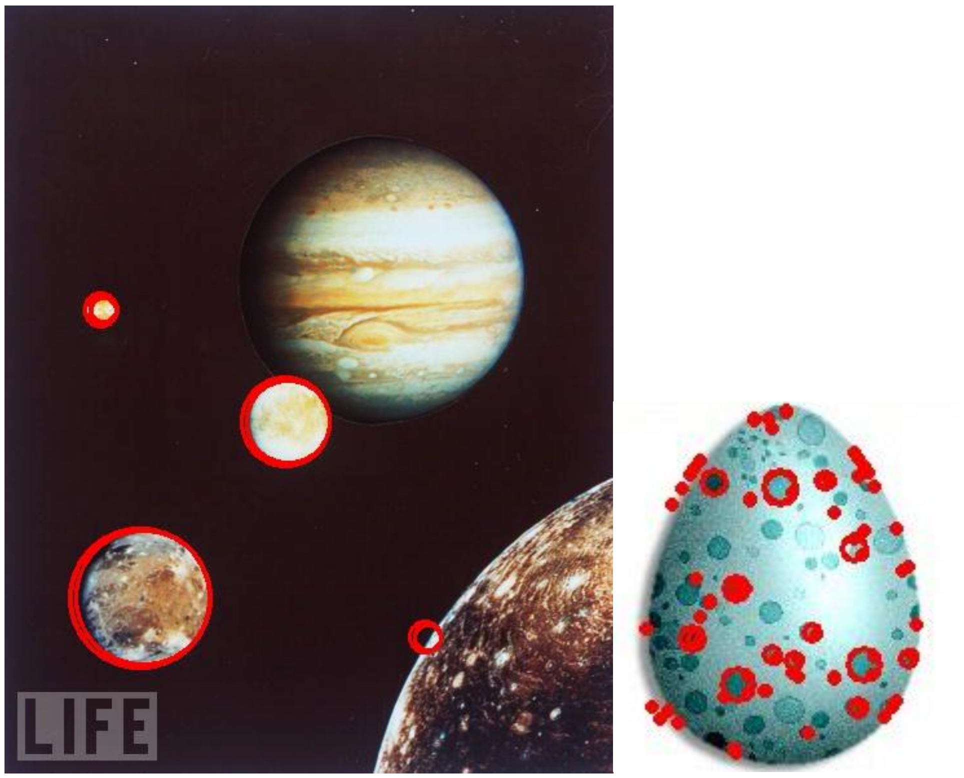

Figure 17: The results of applying any multiple radius detection for Hough circles to the images.

The Code:

Added an additional outer for loop, to cycle through more radii. Also, an additional dimension was added to the accumulator H. The function includes inputs for the range of radii to check.

```
H  = np.zeros((im.shape[0],im.shape[1],int((max_rad-min_rad)/2)))
for i in range(hough_image.shape[0]):
    for j in range(hough_image.shape[1]):
        if hough_image[i,j] > 100:
            k = 0
            for r in range(min_rad,max_rad,2):
                for theta in range(0,180,2):
                    a = round(i - r*c_ang[theta])
                    b = round(j - r*s_ang[theta])
                    if a < im.shape[0] and b < im.shape[1]:
                        H[a,b,k] = H[a,b,k] + 1
                k = k + 1
```