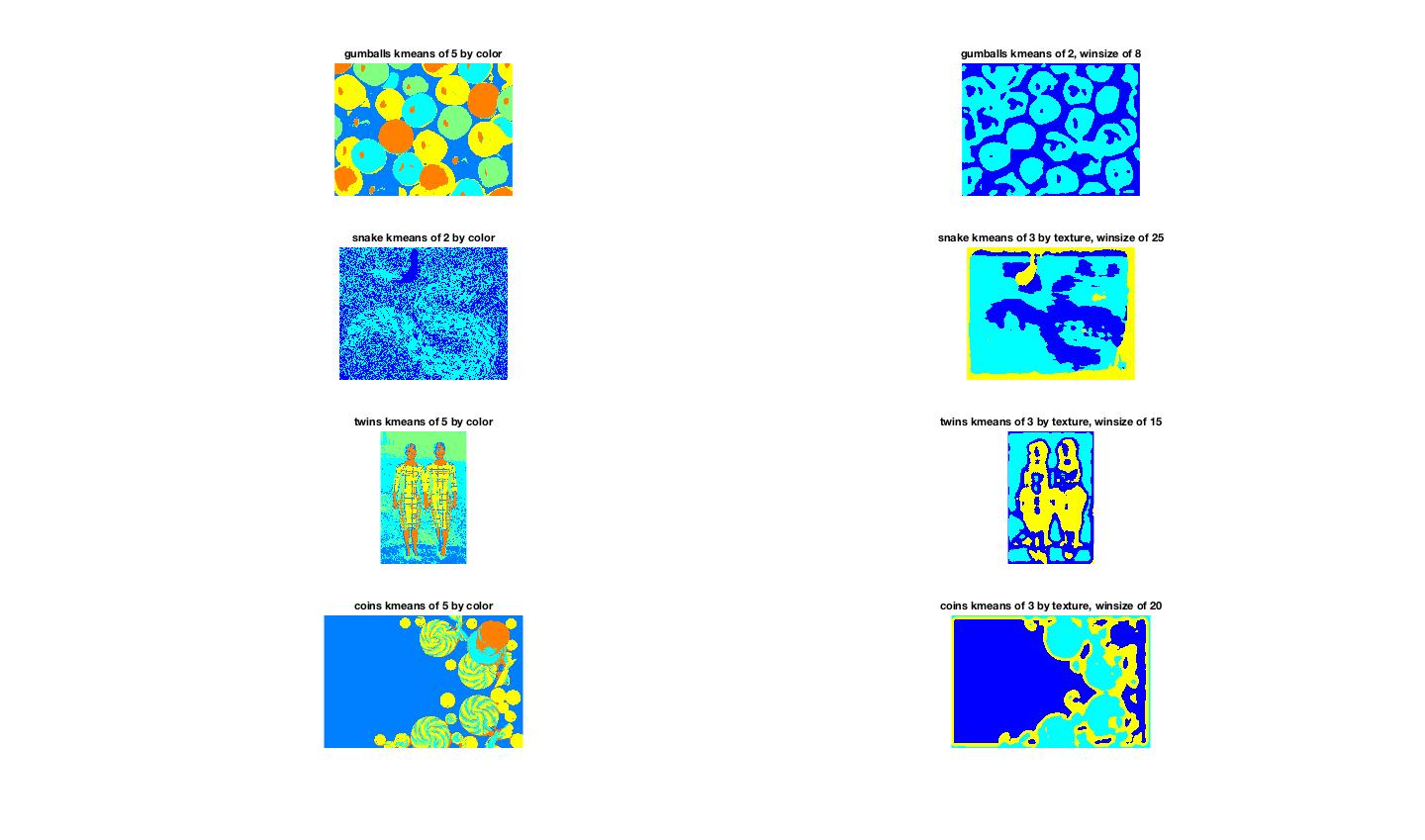
Assignment 2

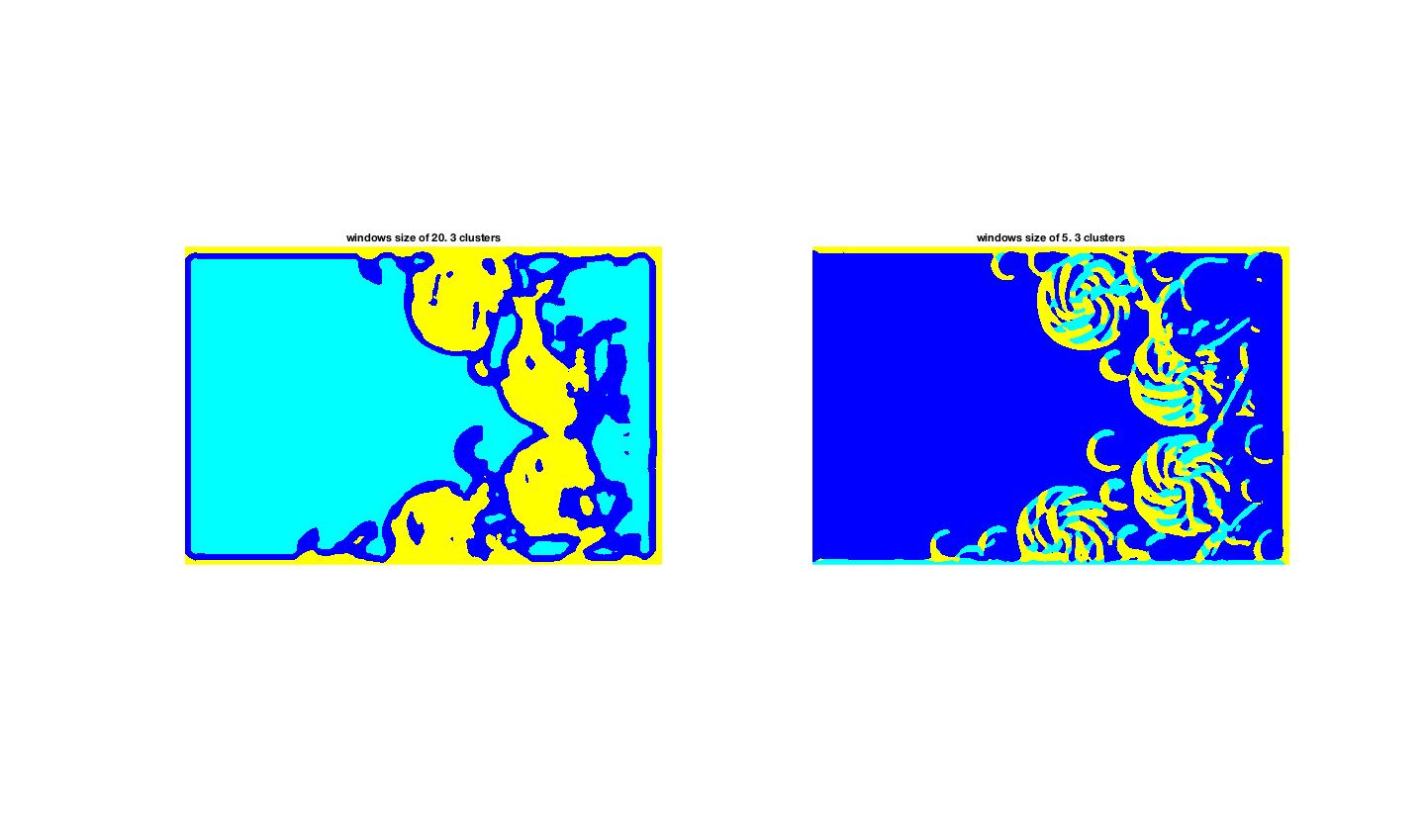
**Part1**

Image results of gumballs, snake, twins and coins. Each has k means by color on the left and kmean by texture on the right.

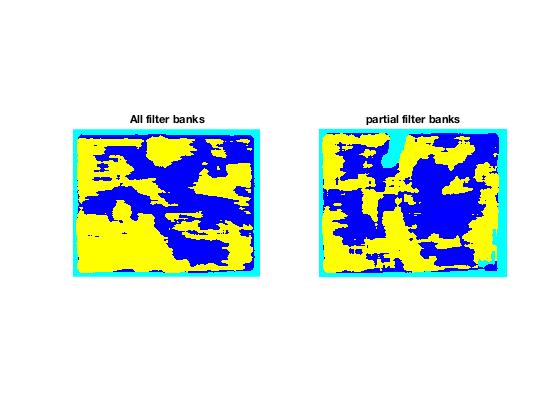
****

**Explanation:**

1. Gumballs: For kmeans by color I use 5 color because from the original image. But this number could vary. I noticed most balls have an orange dot. That is the light reflection. For Kmeans by texture. I use 2 cluster because I assume the balls is one texture and the voids (where there are no balls) is another cluster. And the result worked out pretty well. The optimal window size is obtained from trials.
2. Snake: For kmeans by color I used 2 color because there are not that many color variations in the image. For kmeans by cluster I want to make sure the snake pattern gets covered. Snake has a unique texture but the problem is it is changing direction. Therefore I use a wide (25) window size.
3. Twins: For both kmeans by color and texture the results seem promising because the outputs are symmetric.
4. Coins: Kmeans by color recognized the hat, the coins, and the lollipop. In Kmeans by texture, the hat disappeared because it is thought to have the same texture as the background. The coins and the lollipops, on the other hand, are recognized.



Explanation: Above is the results comparison between using a large window size and small window size for k-mean clustering on texture for the coins image. The Large window clearly could recognize lollipop as a texture. Whereas a smaller window thinks there are multiple textures within the candy. The tradeoff is that large window cannot detect smaller patterns like the coins.



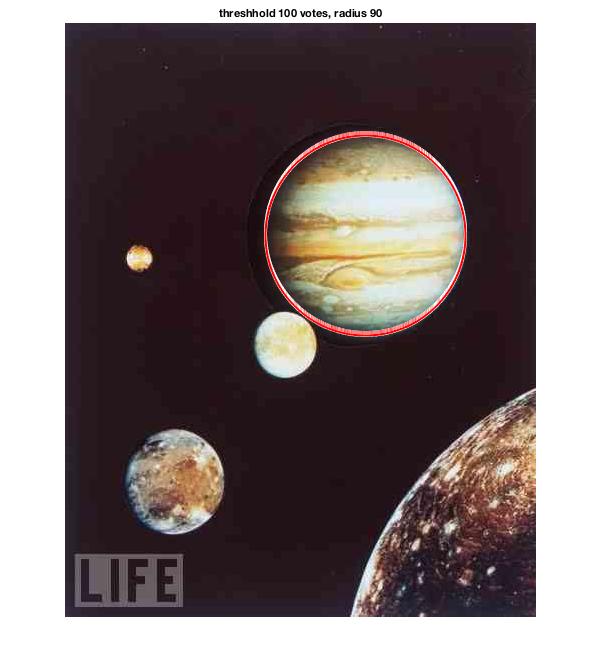
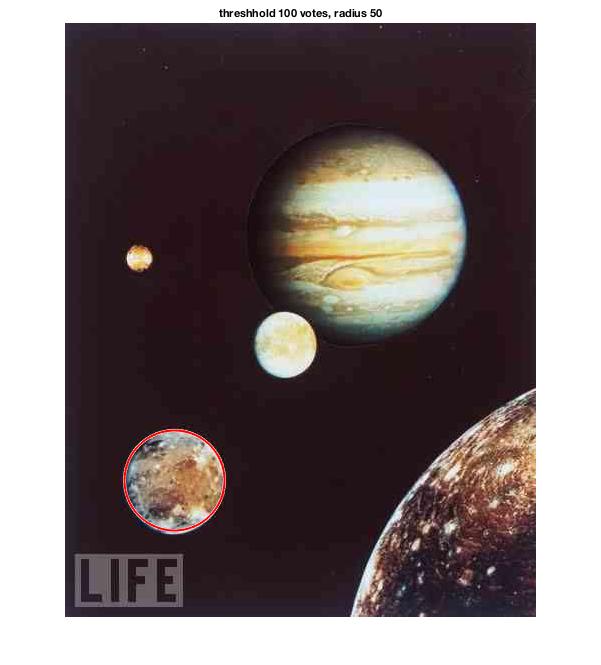
Explanations: Above are the results from kmeans clustering by texture (on snake image), using two different filter banks. Clearly when using all the filter banks, the snake pattern is more complete. The snake curves around. As a result, its body texture changes. With a more complete filter bank the snake body can be recognized fully.

**Part 2**

1. Implementation step: First I run canny edge detection on the grayscale image.

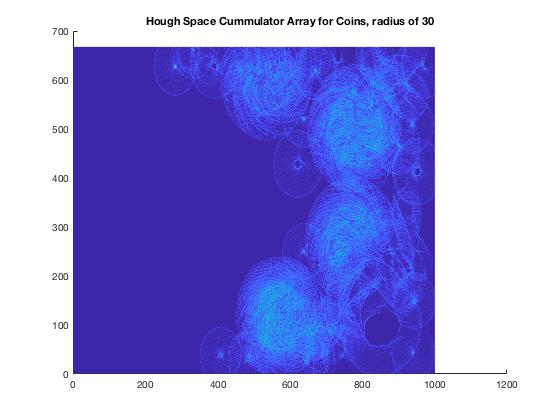
Run a loop through each pixel at of the edge image. Make a circle centered at this pixel and for each point on the perimeter cast a vote. To do this, for each pixel run loop through each angle from 0-2pi (step of 0.01). compute the perimeter pixel index, and let that index cast a vote. I created a “counter\_matrix” same size as the image with each element set to zero initially. When a perimeter pixel casts a vote, increment value at the same location in the “counter\_matrix” by one.

At the end of the iteration. Most pixels will have some votes more or less. However, the centers of circles will likely to have more votes than regular pixels. Create a threshold. Display the pixels that have more votes than the threshold.



Explanation: For each image result, I used a different radius and accumulator array threshold vote count to determine how many circles are detected. By increasing the radius, I am able to detect the large circular objects. Between image 3 and 4, I changed the search radius from30 to 39. By doing this I was able to detect the slightly larger coins I wasn't able to detect before.

C )



Explanation: Here is the accumulator array for coins.jpg. I tried to detect the smaller coins. Since there are a lot of noises many “circle” centers were detected. However, by raising the vote count threshold to 200. Only the correct circles remain.



D )





Explanation: I use two different bin size for the angle step size. One with 0.01 and one with 0.1. To keep the search radius the same, I found a inverse relationship between the bin size and the vote counts. After increasing the bin size, the vote counts in the accumulator array dropped. Therefore I had to lower the threshold count from 200 to 21 in order to detect the circles.

Extra credit: Exploiting the gradient



Explanation: I take advantages of the gradient on the edge of the circle in order to save computation time. First of all, I compute the Gx, Gy gradient at each pixel of the image. Then I take the arctangent of the two gradients in order to get the net gradient angle. After that I casted votes in the accumulator array toward that specific angle. This saves a lot of time. The results are promising as well. I also had had to lower the accumulator array threshold because there are not that many votes total.