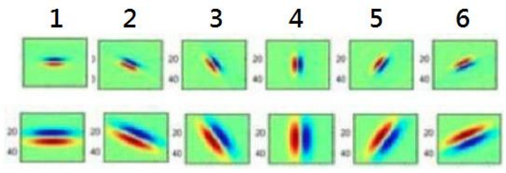
ECS189G Computer Vision Problem Set 2

WEI-CHIH CHEN

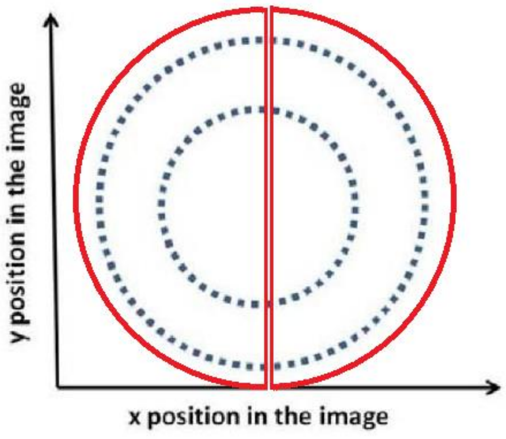
ID: 912448776

I. Short programming example

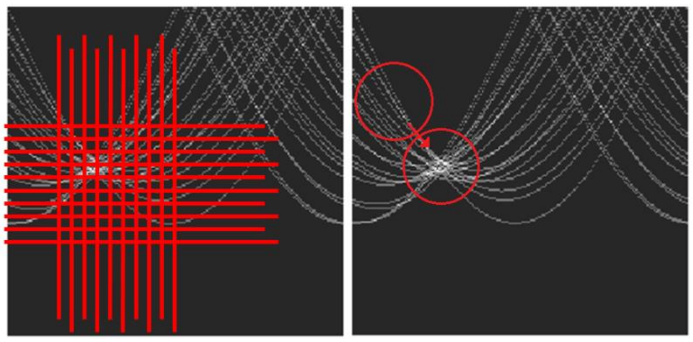
1. It is invariant to orientation. This texture can be represented by the filter bank which is invariant to orientation. We can catalog these filter by orientations in below graph order. In group 1, it is sensitive to horizontal since this filter is derivative in y direction. And so does the other 5 groups which are derivative in five other orientations. Therefore, this filter bank includes six different orientations which means this texture do not emphasis in any orientation. So no matter how we rotate this texture, the response of the filters remains the same.



2. It depends on the initial two points for the k-means. But no matter which two points are chosen, k-means clusters the image into two half shown below. Because the way we clusters points by distance between center, we can only get spherical clusters at the end of k-means algorithm.



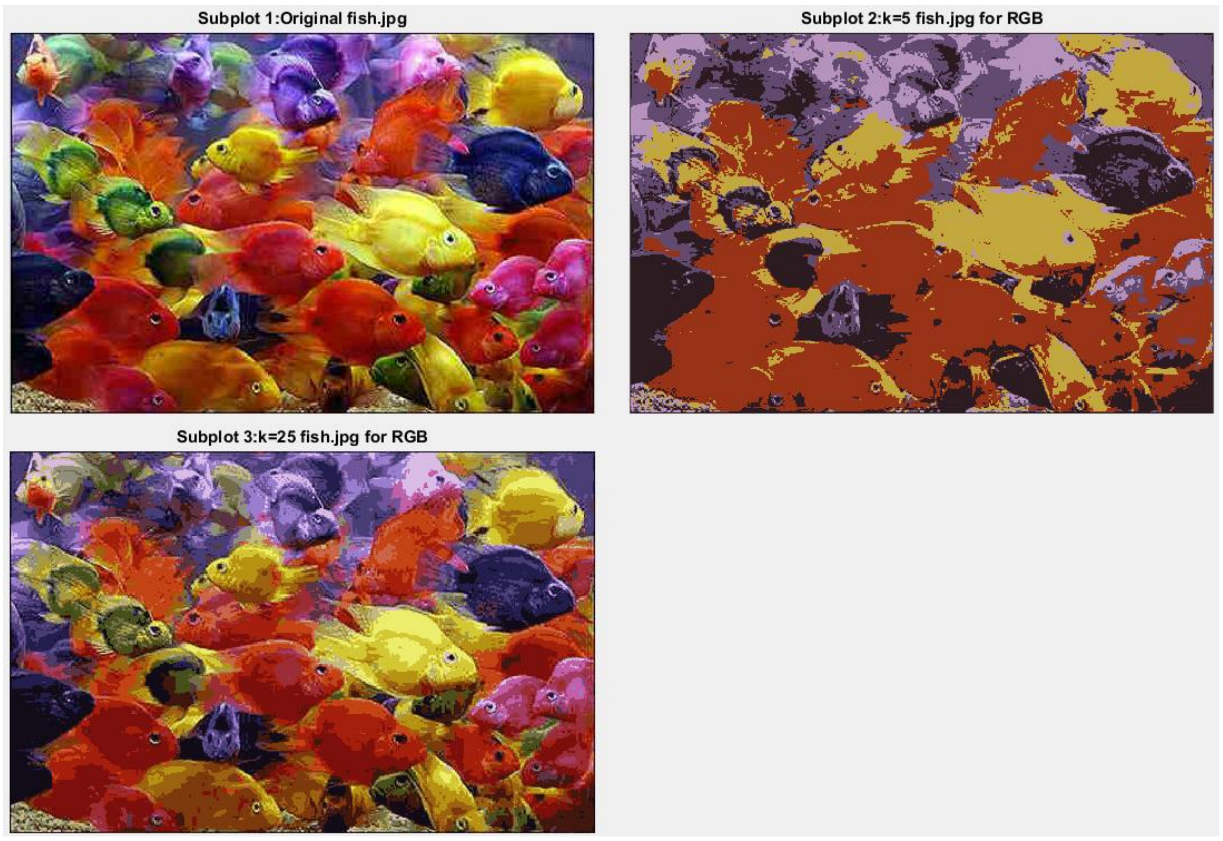
3. Mean-shift is more appropriate. To collect votes by discretization of the parameter space is try to find which specific area on the Hough space is most density. That means this point can truly represent a specific shape, like line or circle, back to original image. Mean-shift algorithm is doing the same thing which try to find the most density region. For example from the lecture slide, if we discretize a Hough space and try to find which small region get most vote, mean-shift can get the mass center of mass and get the exactly same result.



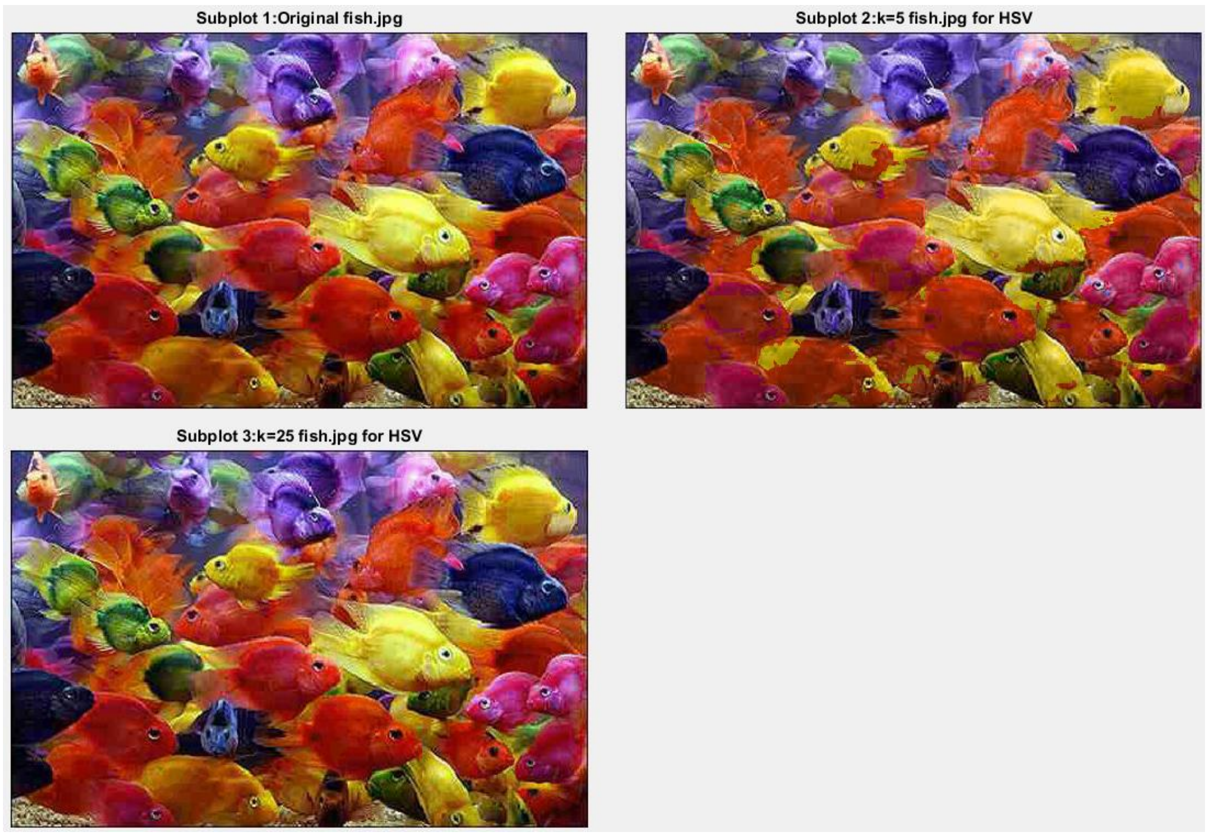
4. (1) Supposed we have $R_1, ..., R_K$ number of blobs.
   (2) And we quantize aspect ration into $R_1, ..., R_m$ level.
   (3) Using these two parameter, let number of blobs as x-axis and aspect ration as y-axis. And then use k-means to group different clusters in below. The blobs with the same clusters are the same group.
   Step 1. Randomly initialize the cluster centers, $c_1, ..., c_K$
   Step 2. Given cluster centers, determine points in each cluster
      - For each point p, find the closest c_i. Put p into cluster i
   Step 3. Given points in each cluster, solve for c_i
      - Set c_i to be the mean of points in cluster i
   Step 4. If c_i have changed, repeat Step 2
   (this algorithm from slide 7 page 40)

II. Programming problem: content-aware image resizing

1. Color quantization with k-means
   (a) See Code
   (b) See Code
   (c) See Code
   (d) See Code
   (e) See Code
      a. RGB, k=5 and k=25 graphic for fish.jpg


Subplot 1:Original fish.jpg
Subplot 2:k=5 fish.jpg for RGB
Subplot 3:k=25 fish.jpg for RGB

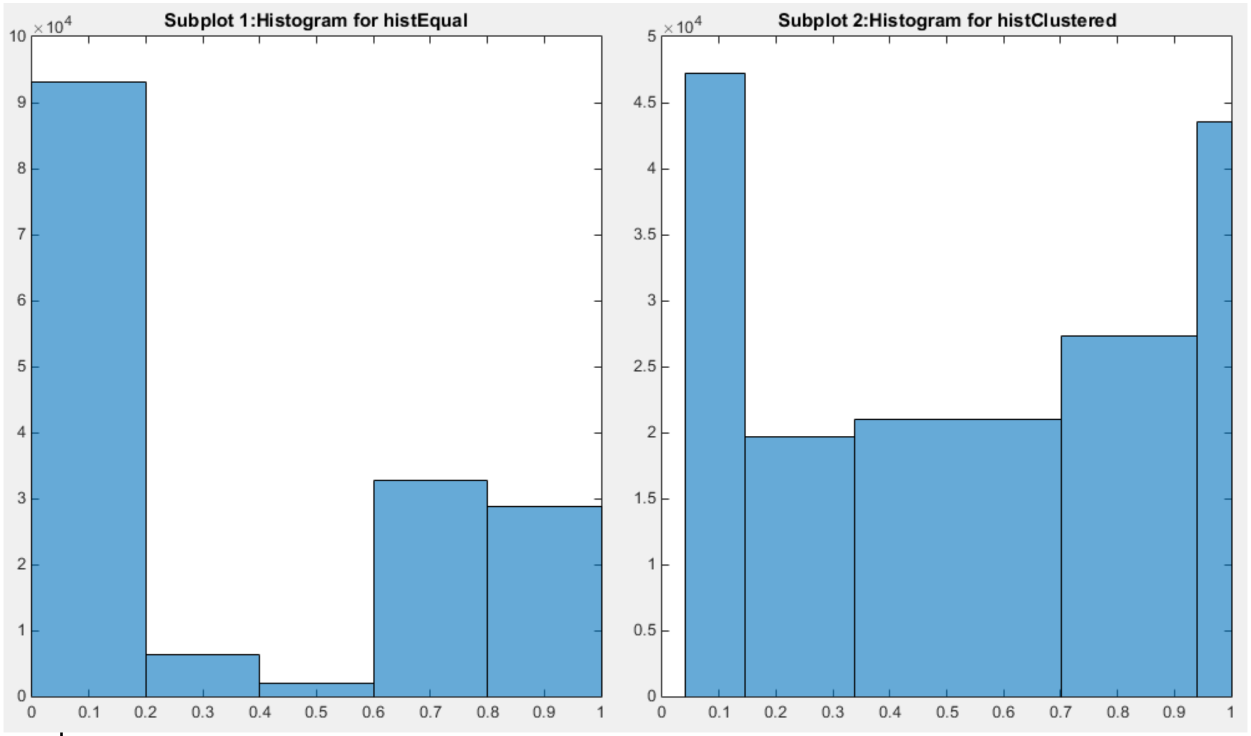
b. HSV, k=5 and k=25 graphic for fish.jpg


Subplot 1:Original fish.jpg
Subplot 2:k=5 fish.jpg for HSV
Subplot 3:k=25 fish.jpg for HSV

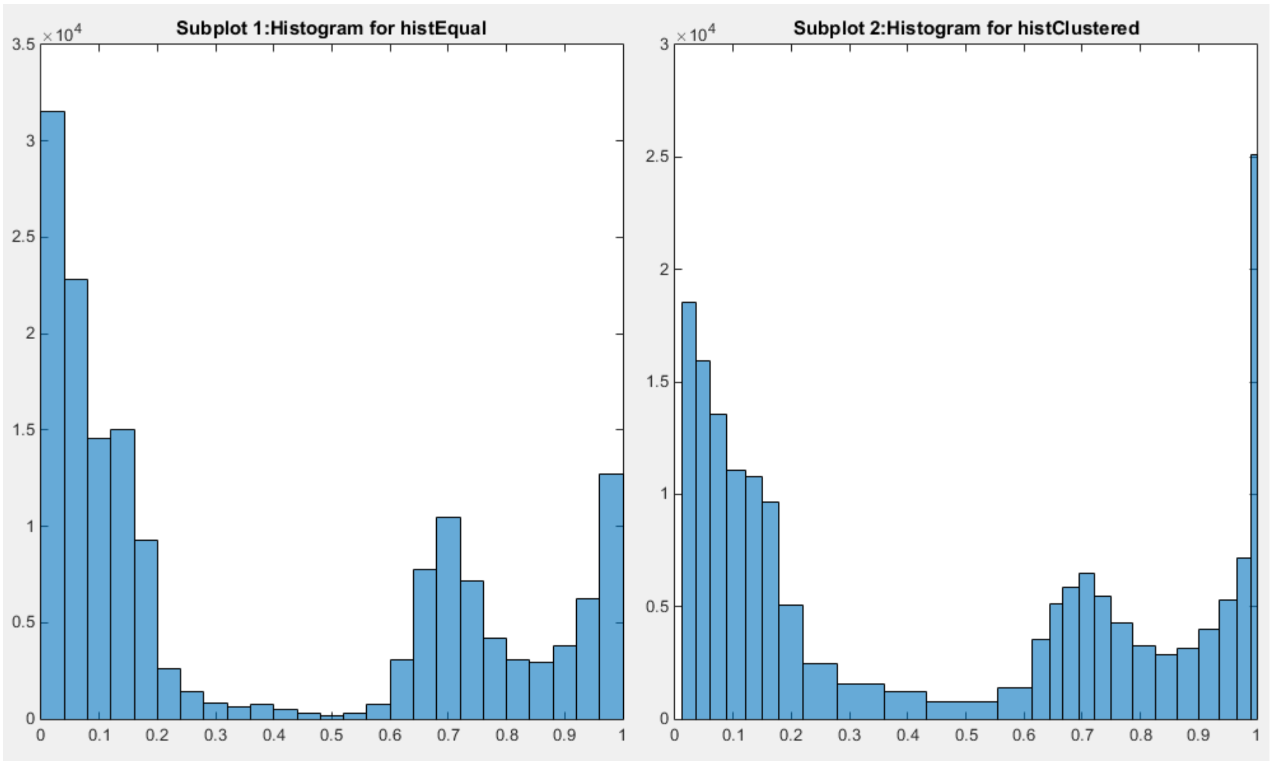
c. SSD errors

SSD error	RGB	HSV
K=5	463,803,422	70,716,903
K=25	135,154,986	3,813,824

d. Histogram for K=5. Left is **histEqual** and right is **histClustered**.



K=25. Left is **histEqual** and right is **histClustered**.



(f) Explain:

1. How do the two forms of histogram differ?

   HistEqual represents color distribution. If a picture is bias on a specific color, histogram distribution is bias on that specific side. For example, in our fish.jpg picture, this image is inclined to purple like and read like colors, the histogram distributes uneven on smaller and larger hue value.

   HistClustered represents how many pixels was group into a cluster. And I used k-means to cluster pixels. It reflects on how pixels group together based on hue value. So the histogram distribution can be easily shown that some small range of hue values are group into a cluster and some large range are groups into another.

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

   After k-means quantization, image is classified into only k different colors on **RGB** color space but classified into more than k different colors on **HSV** color space. Because we only classify color by hue value in HSV and remain saturation and value unchanged, the result of HSV quantization should be more colorful. But after HSV quantization, image remains k color system.

3. The value of k?

   Larger k gets more quantization level. Therefore, result image is more colorful.

4. Across different runs of k-means (with the same value of k)?

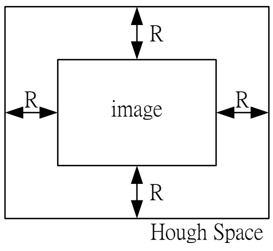
   More runs of k-means is more accurate classify k groups in given image which means we get less sum of square error. For example, if we run only 5 iterations, kmeans may even get wrong classification and get a large SSD error.

2. Circle detection with the Hough Transform

(a) STEP1 image→Canny: `edge(im,'Canny')` to get canny edge.

STEP2 Canny→Hough:

1. Set a Hough Space: I set size of Hough Space is image plus radius which shown below



2. If useGradient is 0, plot a circle on Hough Space when pixel in canny is 1. And every point on the circles should plus one.

EX: pseudo code:

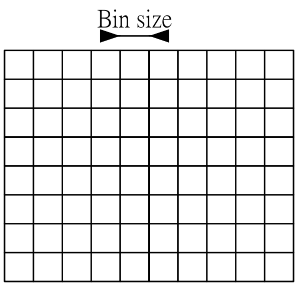
```
For loop: Run every pixels of canny image
{    If pixel==1, loop θ from 1 to 360
        {    a= $\mathrm{radius} + 1 + \mathrm{x} - \mathrm{radius} * \cos(\frac{\theta}{360} * 2\pi)$
             b= $\mathrm{radius} + 1 + \mathrm{y} + \mathrm{radius} * \sin(\frac{\theta}{360} * 2\pi)$
             HoughSpace[b,a] +=1
}}
```

3. If useGradient is 1, calculate gradient direction by imgradient function. Each value of gradient direction represents θ as that pixel. And using $\theta - 45° to\ \theta + 45°$ as predict range to get a and b of HoughSpace.

EX: pseudo code:

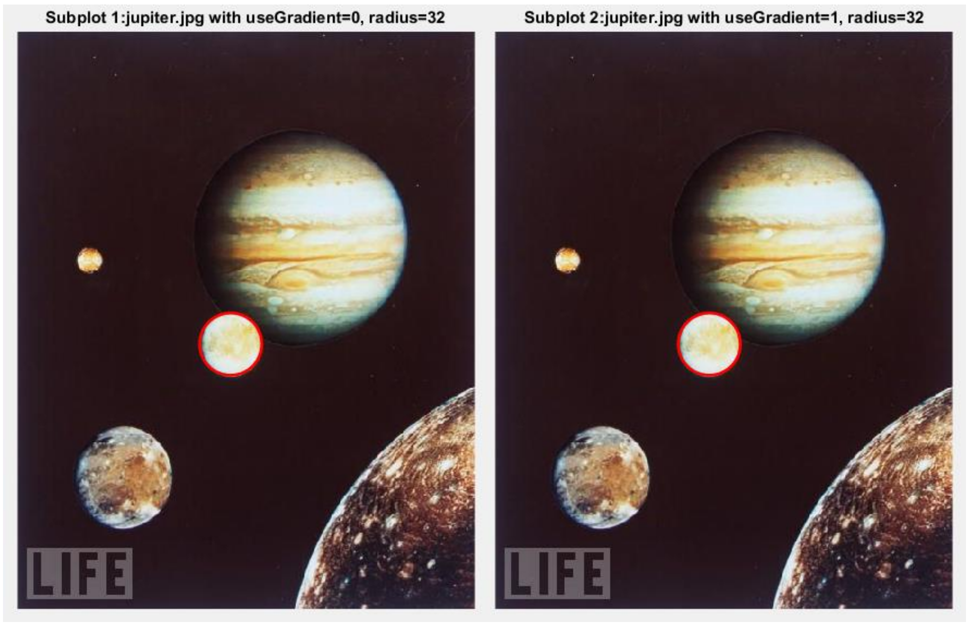
```
For loop: Run every pixels of gradient direction (same size as image)
{    For loop θ from $\theta - 45° to\ \theta + 45°$
     {a= $\mathrm{radius} + 1 + \mathrm{x} - \mathrm{radius} * \cos(\frac{\theta}{360} * 2\pi)$
      b= $\mathrm{radius} + 1 + \mathrm{y} + \mathrm{radius} * \sin(\frac{\theta}{360} * 2\pi)$
      HoughSpace[b,a] +=1
}
```

STEP3 Voting: Quantizing HoughSpace into bin size shown below. And find max value within each bin. And find max values of all bins. Every bin value large than 0.9*max value view as target fitting circles.

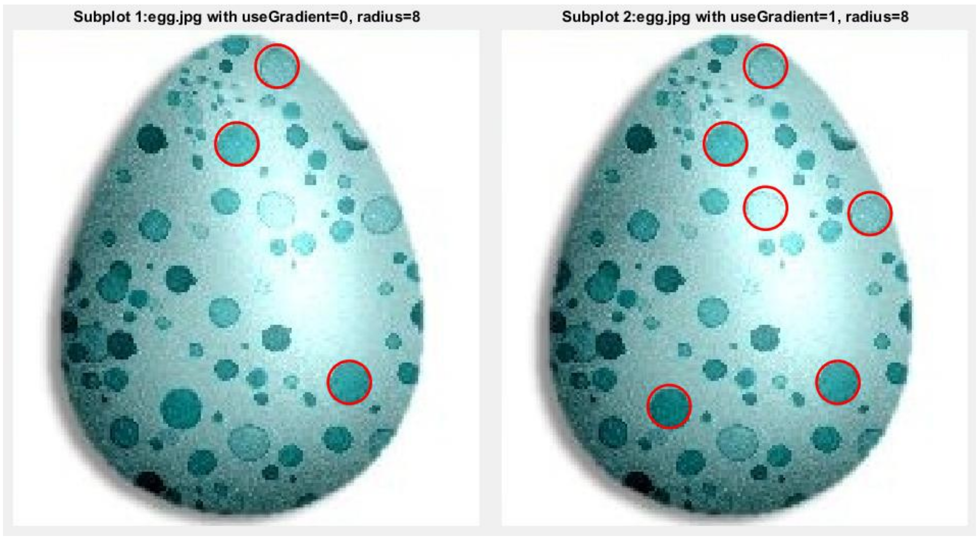


STEP4 trace back HoughSpace coordinates to original image coordinates which are `centers`.

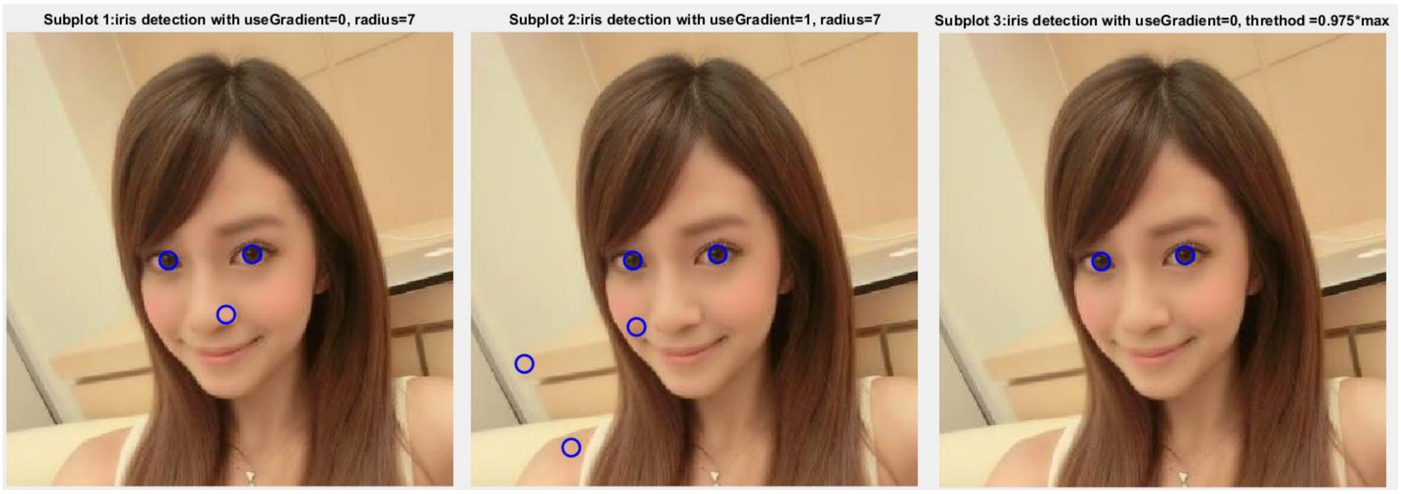
(b) Jupiter.jpg


Subplot 1:jupiter.jpg with useGradient=0, radius=32
Subplot 2:jupiter.jpg with useGradient=1, radius=32
LIFE
LIFE

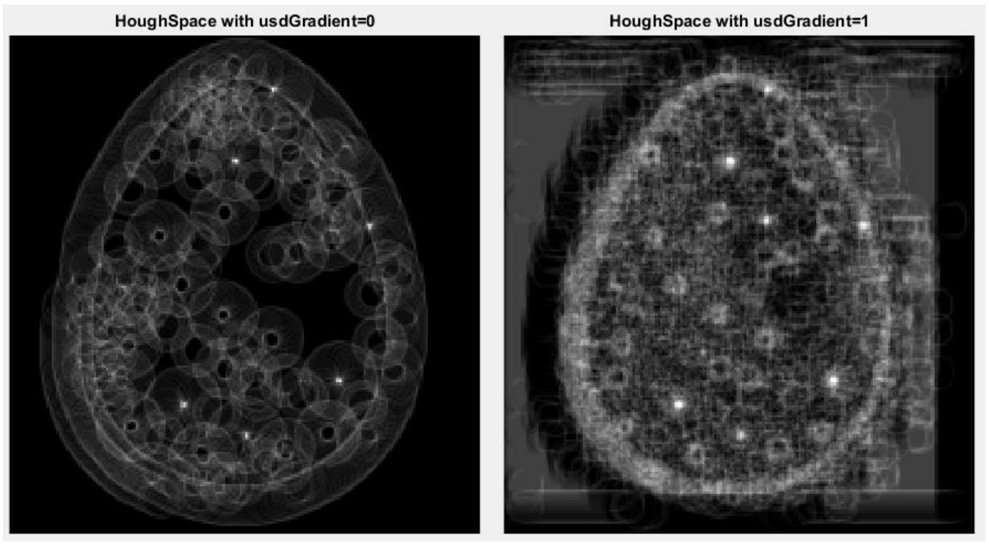
Egg.jpg


Subplot 1:egg.jpg with useGradient=0, radius=8
Subplot 2:egg.jpg with useGradient=1, radius=8

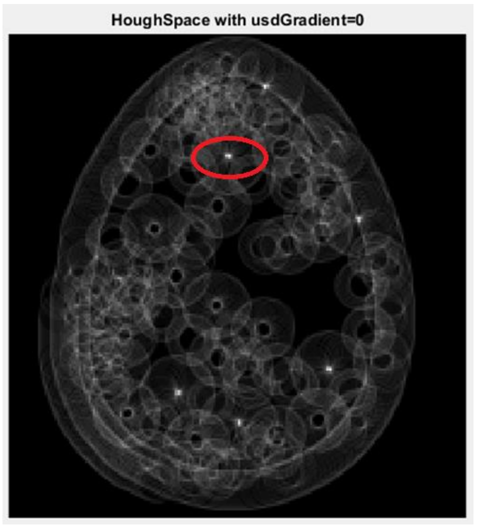
Iris detection with bin size=30, threshold votes=0.9*max votes. If we specific select bin size=5, threshold votes =0.975 max votes and useGradient=0, we can perfectly detect iris. Or if we add some other features in the image, we can eliminate other unrelated fitting circles.


Subplot 1:iris detection with useGradient=0, radius=7
Subplot 2:iris detection with useGradient=1, radius=7
Subplot 3:iris detection with useGradient=0, threthod =0.975*max

Accumulator arrays with `useGradient` to 0 and 1.



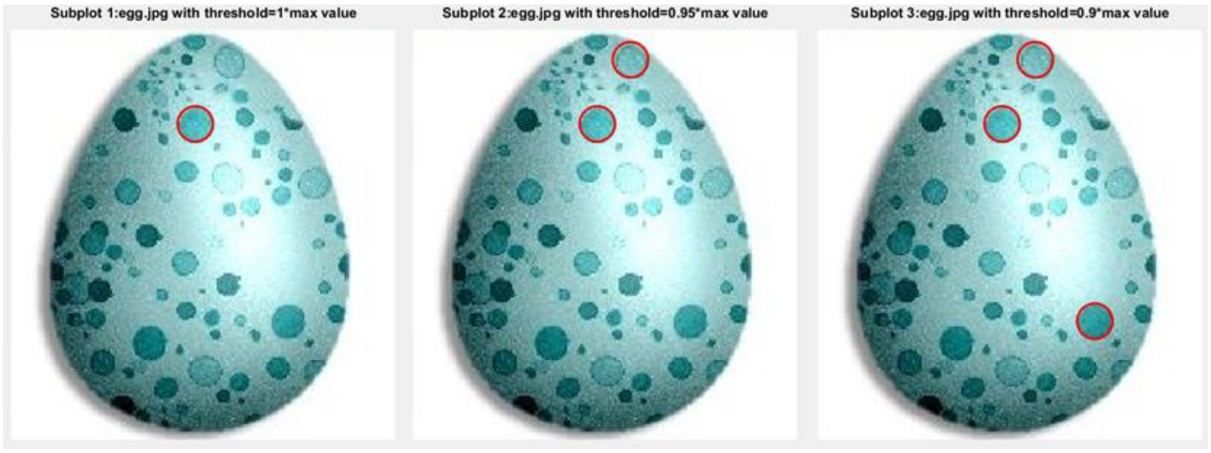
(c) HouphSpace

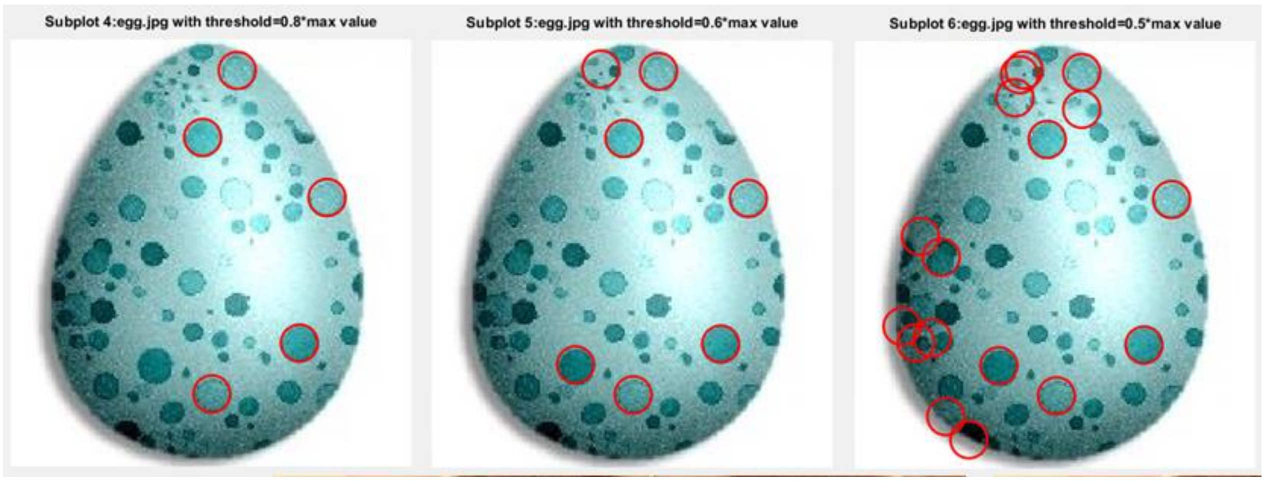


In this egg.jpg image, I set radius =8 which is the red circle shown right. If some pixels get higher votes, the whiter is shown in right image. Therefore, circles with radius around 8 can get higher vote in accumulate array. So we can see there are six possible candidates which radiuses are near 8. I set 0.9*max value of Houph Space accumulate array as threshold votes. All candidate votes higher than threshold votes are chosen as final fitting centers in original egg.jpg image. In this case, highest pixel gets 268 votes and the other two selected centers get 256 and 246 each which can be seen in previous (b) image.

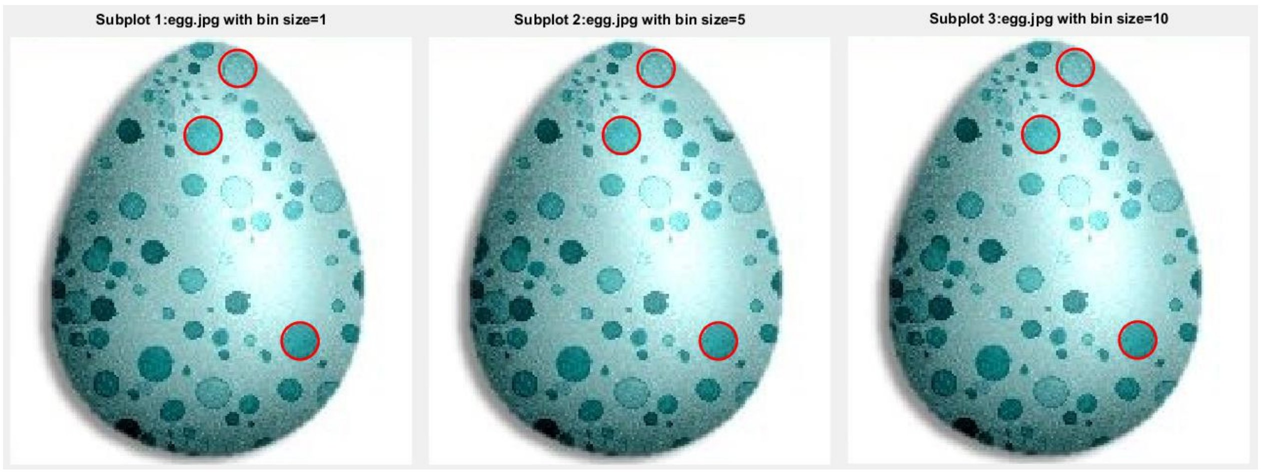
With other radius value circles, all of them cannot accumulate votes in a same point. Therefore, all of them do not get high enough votes.

(d) If I set threshold votes from max value to 0.5* threshold votes, the candidate of possible fitting circles vary from one to five and even with non-fitting circles. Below image is experiment with useGradient=0 case. For useGradient=1 case, I set threshold votes to 0.9* threshold max votes. And also get six possible candidate fitting circles which shown in (b) picture.

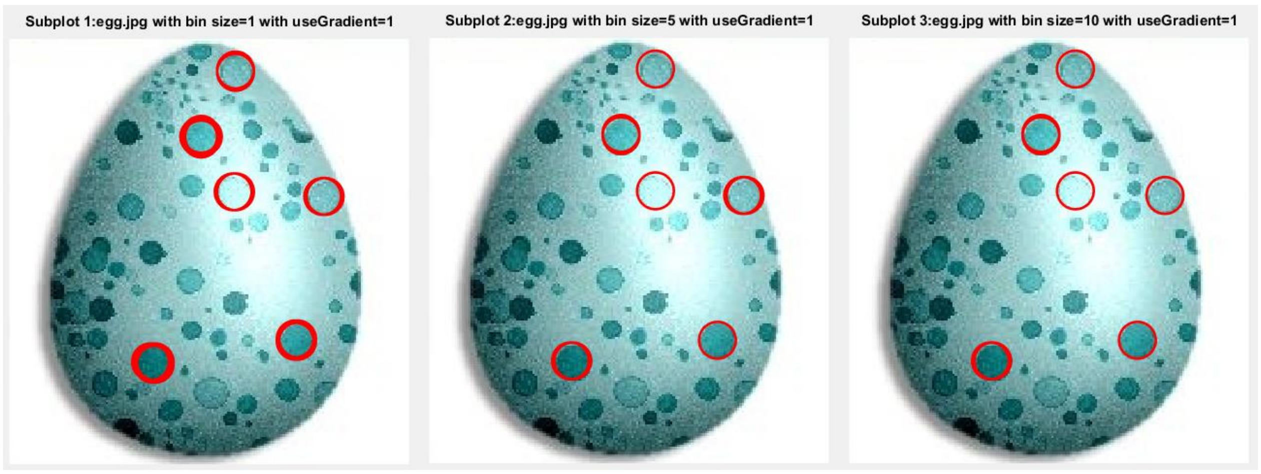

Subplot 1:egg.jpg with threshold=1*max value
Subplot 2:egg.jpg with threshold=0.95*max value
Subplot 3:egg.jpg with threshold=0.9*max value


Subplot 4:egg.jpg with threshold=0.8*max value
Subplot 5:egg.jpg with threshold=0.6*max value
Subplot 6:egg.jpg with threshold=0.5*max value

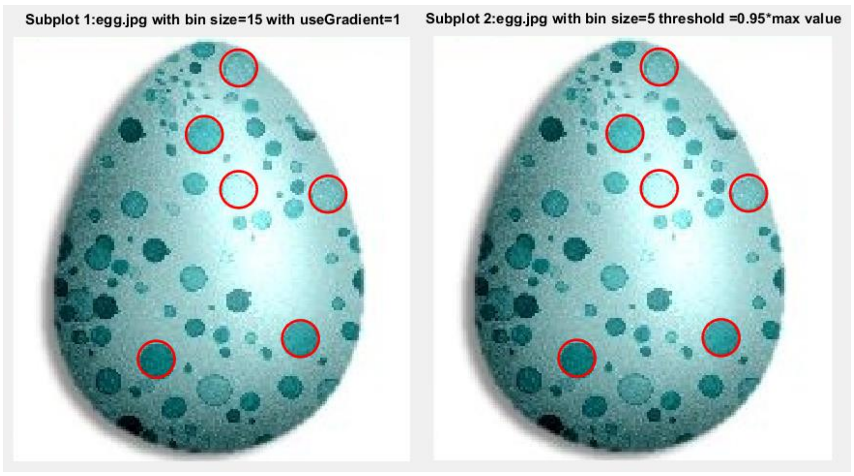
(e) I choose egg.jpg with useGradient=0 case in below image. With bin size 1, 5 and 10, it does not affect the select result.


Subplot 1:egg.jpg with bin size=1
Subplot 2:egg.jpg with bin size=5
Subplot 3:egg.jpg with bin size=10

If I choose egg.jpg with useGradient=1 case with bin size 1, 5 and 10, it does affect the select result. Since gradient direction can focus accumulator array votes on a more possible center, it may not focus enough that causes reselect same fitting circles.


Subplot 1:egg.jpg with bin size=1 with useGradient=1
Subplot 2:egg.jpg with bin size=5 with useGradient=1
Subplot 3:egg.jpg with bin size=10 with useGradient=1

To solve this problem, we can either choose higher bin size or select higher threshold votes. Both of them are effect with useGradient=1 case.


Subplot 1:egg.jpg with bin size=15 with useGradient=1
Subplot 2:egg.jpg with bin size=5 threshold =0.95*max value

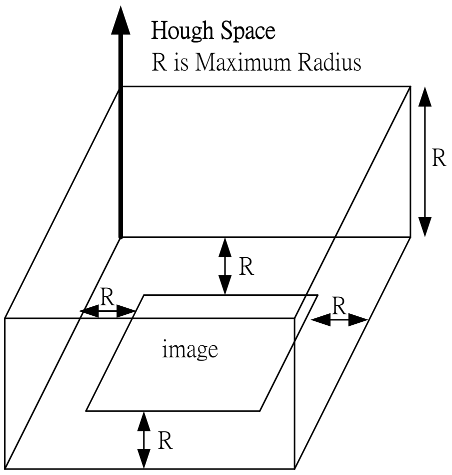
III. Extra credit

Based on 2D algorithm, I increase Hough Space to 3D. Below is my algorithm.

(a) STEP1 image→Canny: `edge(im,'Canny')` to get canny edge.

STEP2 Canny→Hough:

1. Set a Hough Space: I set size of Hough Space is image plus radius which shown below

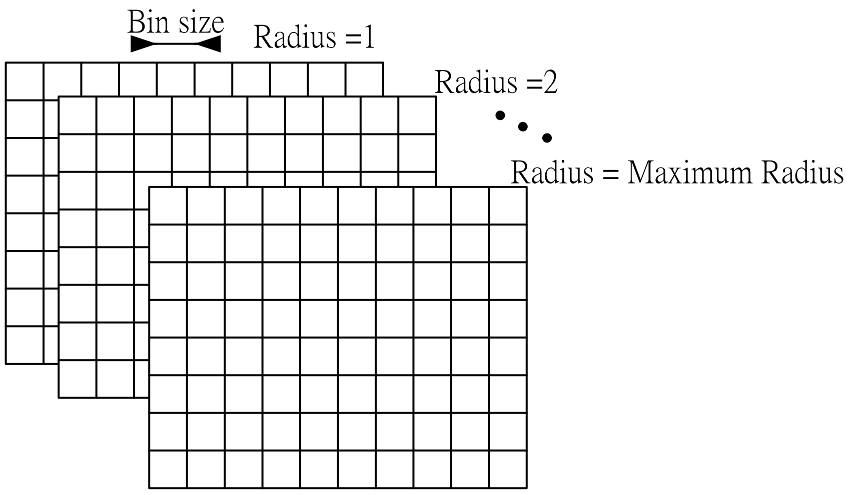


2. useGradient is 0, plot a circle on Hough Space when pixel in canny is 1. And every point on the circles should plus one.

EX: pseudo code:

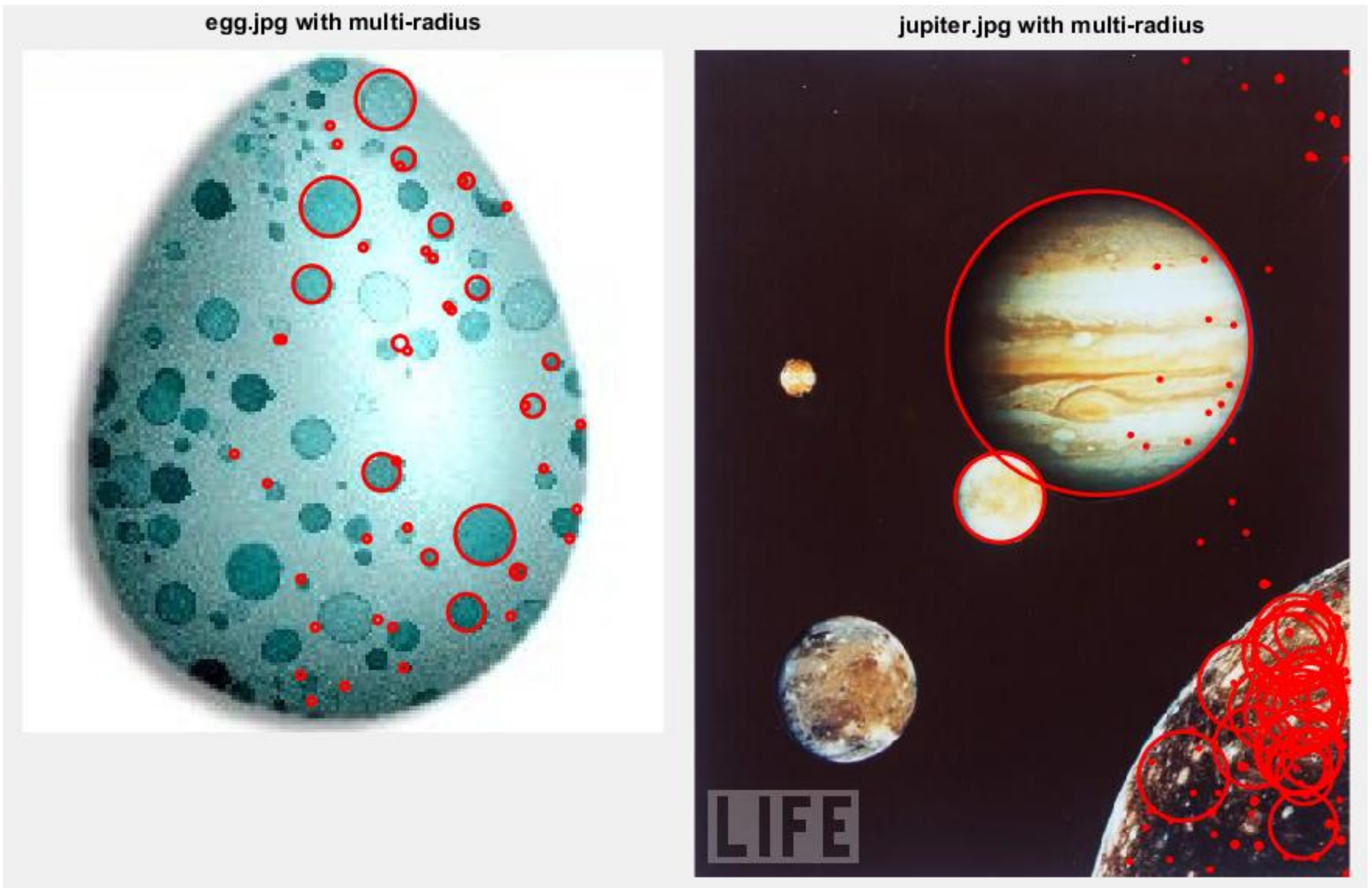
```
For radius: Run from 1 to Maximum radius
    For loop: Run every pixels of canny image
    {     If pixel==1, loop θ from 1 to 360
          {     a= Maximum radius + 1 + x − radius ∗ cos($\frac{\theta}{360} * 2\pi$)
                b= Maximum radius + 1 + y + radius ∗ sin($\frac{\theta}{360} * 2\pi$)
                HoughSpace[b,a,radius] +=1
    }}}
```

STEP3 Voting: Quantizing HoughSpace into bin size shown below. And find max value within each bin. And find max values of all bins. Every bin value large than 0.9*max value view as target fitting circles. In different radius, I calculate its own accumulator array.



STEP4 trace back HoughSpace coordinates to original image coordinates which are `centers`.

But in order to get better fit circles, I need to tuning parameter several times. Besides, sometimes small radius can randomly get higher votes than large radius. I try to tuning a better result. Below are my best results. Although it still misses some circle, I think there is still have another way to improve accuracy.



[NOTE] I named this code as “detectCircleswithoutR.m” in folder.