**Task1:**

1. Your own Kmeans clustering based implementation – 1 mark

**Answer: in task1.py -- def CartoonNizer\_1(image):**

After reading paper about Kmeans on image segmentation, choose Gaussian-means(sigma=0.01) to implement the task.

Firstly, update\_centroids(centroids, hist) function and k\_means(hist, alpha) function are for Kmeans algorithm. Then CartoonNizer\_1(image) makes the source image cartoon.

This method is to finish the clustering otherwise find the centroids and update them, when the distribution is normal distribution. Use histogram to get every channel dataset for Kmeans. Then use Kmeans to assign new pixel to cartoon.

1. Bilateral filtering-based implementation – 2 mark

**Answer: task1.py - def CartoonNizer\_2(image): and my\_bilateral.py**

A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. This weight can be based on a Gaussian distribution. (From Wikipedia) The bilateral function in skimage module has two important parameters, which are sigma\_color (the larger value the larger radiometric differences) and sigma\_spatial (the larger value the larger spatial differences).

Two method in this part are the function from skimage module and my own bilateral function. The first one does not work well and make some changes in part3. The second takes more time on calculation.

My bilateral filter works well and this function takes about **6 minutes.**

1. Improved (your own/inspired) method and implementation – 3 mark

**Answer: task1.py - def CartoonNizer\_2\_2(image): and my\_bilateral.py**

1. For task 1 part 2 bilateral filtering, it is a technical processing that choosing the appropriate parameters. To improve the bilateral filtering performance, I choose to do the image processing **8** times with the same parameter as part 2. The final result is much better than performing once.
2. Combine Kmeans and bilateral filtering. Firstly, use Kmeans do the clustering. Then, smooth the image via bilateral filtering. The two images that are from part1 and part3\_2 show no big difference.
3. Change calculating on 2-d image into 3-d image.

**Task 2 (5 Marks total)**

The task is to find the number of rings (semi-circles, spheres and complete circles) in the attached image. Code function CountShapes\_ID(image). It takes the input image as a parameter and outputs the number of semi-circles, spheres and complete circles and also displays and saves the output image with highlighted objects.

CountShapes\_ID(image) in python file– 3 marks

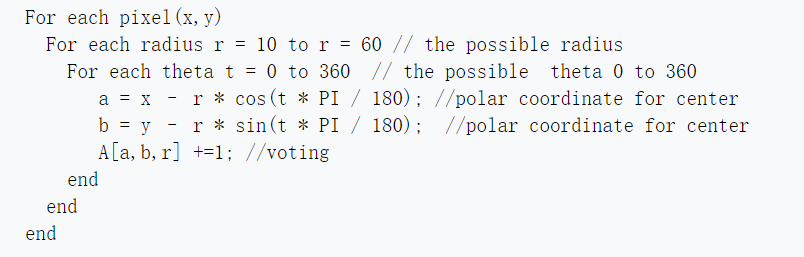
Report containing details – 2 marks

**Answer:**

1. Read the given materials to understand what is Hough transform. I also read some paper, code and blog about Hough transform that is about axis transform.
2. Then complete the function: CountShapes\_ID(image)
   1. CountShapes\_ID(image) is to read the task image and preform the task
   2. In the above function, it has three steps to complete the Hough transform.

Firstly, use skimage.color.rgb2gray to make input gray. Then use the canny detector (from assignment 1) to make gray image into edge image. Finally, do the Hough transform on the edge image, show it and save it.

* 1. The Hough transform is on my one and some condition statements are from hough\_circle code from skimage, opencv and matlab, code implementation and principle mathematic from github, baidu and google.
  2. Implement the algorithm below.



1. I choose radius range from (10, 15) because of time. If making the radius bigger, it can detect more space and may find more circles. It takes much time to run and the sum of five loop running time is 2680.01



Ps: may can use radius increasing 3 or 5 each step to save time.