In the Name of God

Signals and Systems

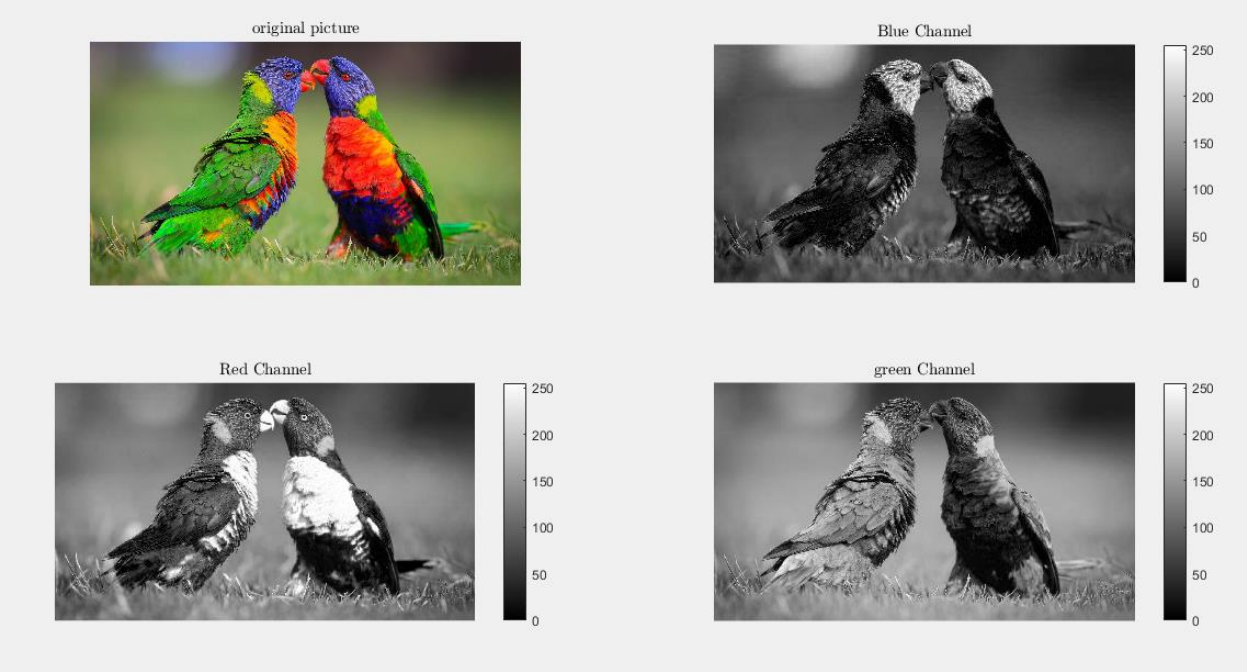
Matlab_Homework_2 Report

Armin Panjehpour – 98101288

1- Image Processing:

1.1- some simple functions for image processing:

1.1.1-



The first index of the third dimension is the red channel and the second one is the green channel and the third one is the blue channel. We can split the channel i using this line of code:

```
iChannel = pic1(:,:,i);
```

As we can see in each channel, the intensity of its colors is high(goes to the white).

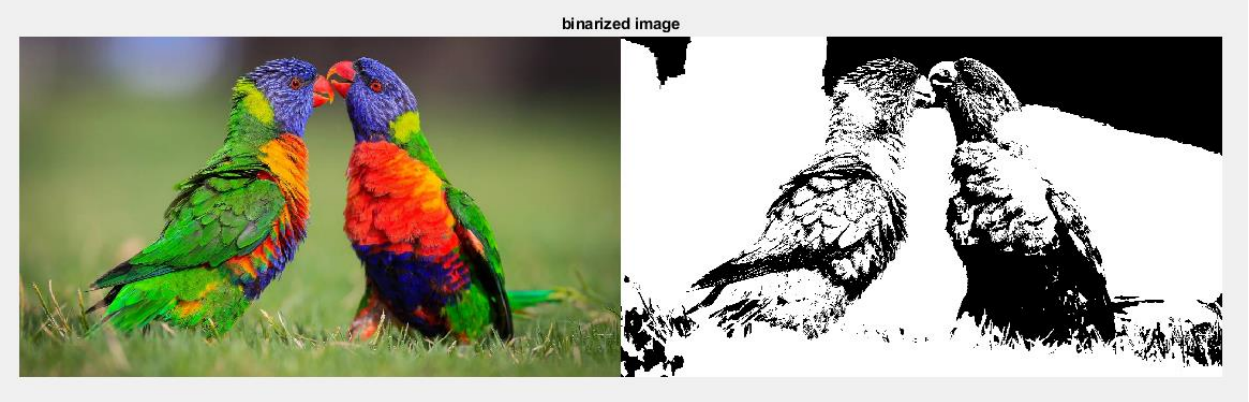
1.1.2-

rgb2gray converts RGB values to grayscale values by forming a weighted sum of the R, G, and B components:

$$0.2989\,R + 0.5870\,G + 0.1140\,B$$

but when we get the mean of sum of RGB channels, the weights of all of them is 1/3. That`s why our picture that most parts of it are green, have less intensity, because it`s weight is 1/3 and the green parts are a little bit darker. B/R colors have more intensity, because weight of them is 1/3 and it`s more than the rgb2gray method.(But the differences are very low)

1.1.3-



imbinarize function use Otsu`s method to binarize the images. Actually it chooses a threshold to minimize the intraclass variance just like what we`re going to do soon in question.1.3. This function makes all the pixels with the intensity lower than the threshold, black and the pixels with the intensity more than the threshold, white.

1.2- Edge Detection:

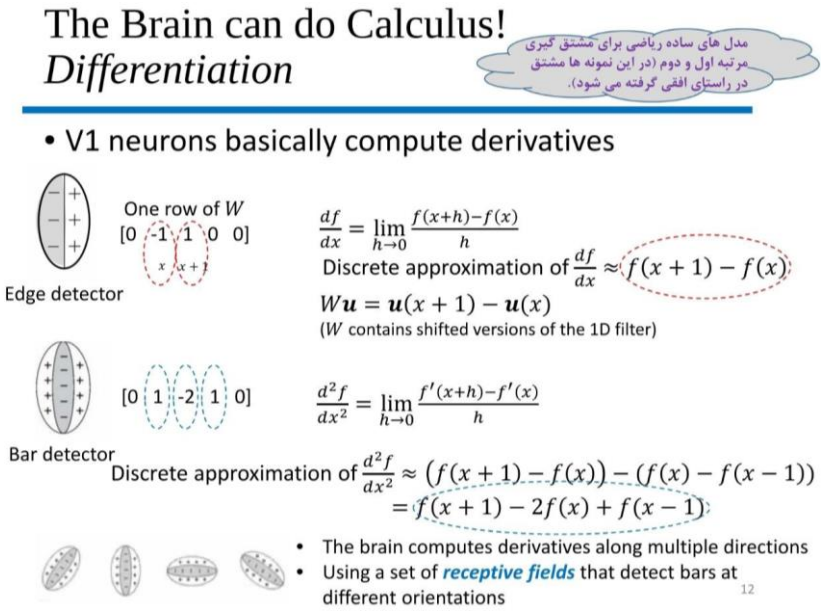
1.2.1- **Sobel-Feldman** is an algorithm in image processing to detect the edges in the image. Actually this is a discrete differentiation operator which computes the gradient of each points`s intensity in the 2 principles directions, horizontal and vertical. This algorithm convolves the image with a filter in the horizontal and vertical directions. We use 2 kernels to convolve with the image which each of them calculate the derivatives, one for horizontal changes and one for vertical changes. The two kernels and their convolution to the image:

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} * A, G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix} * A$$

The final image will be the magnitude of G_x and G_y: $G = \sqrt{{G_x}^2 + {G_y}^2}$

Actually, something like this happens in the brain too. In primary visual cortex, we have neurons with receptive fields like this which they calculate derivatives for edge and bar detection:

Depend on the receptive fields, they will compute first order or second order derivatives:



1.2.2-

Original image:

Edges:

☺☺

1.3- Image segmentation:

1.3.1- K-means:

$$S = argmin_{S=[S_1,S_2,...S_k]} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2$$

In K-means algorithm we`re looking forward to make the S minimum. Actually this is the variance of each cluster, the less the variance is, the better we have done the clustering. Actually each data has aim to the cluster with the closest mean to the data. Competitive and Unsupervised learning is so much close to this algorithm. (K-Means clustering is an unsupervised learning algorithm)

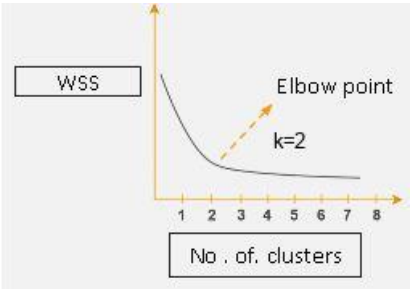
1.3.2-

In this question, we have number of the clusters but in general we have to find the optimum number of clusters. This is done with some method called, **Elbow Method**. We have to use something called within-sum-of-squares(WSS) as a measure to find the optimum number of clusters needed for our data. WSS is defined as the sum of squared distance between each member of the cluster and its centroid:

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

Where x_i = data point and c_i = closest point to centroid

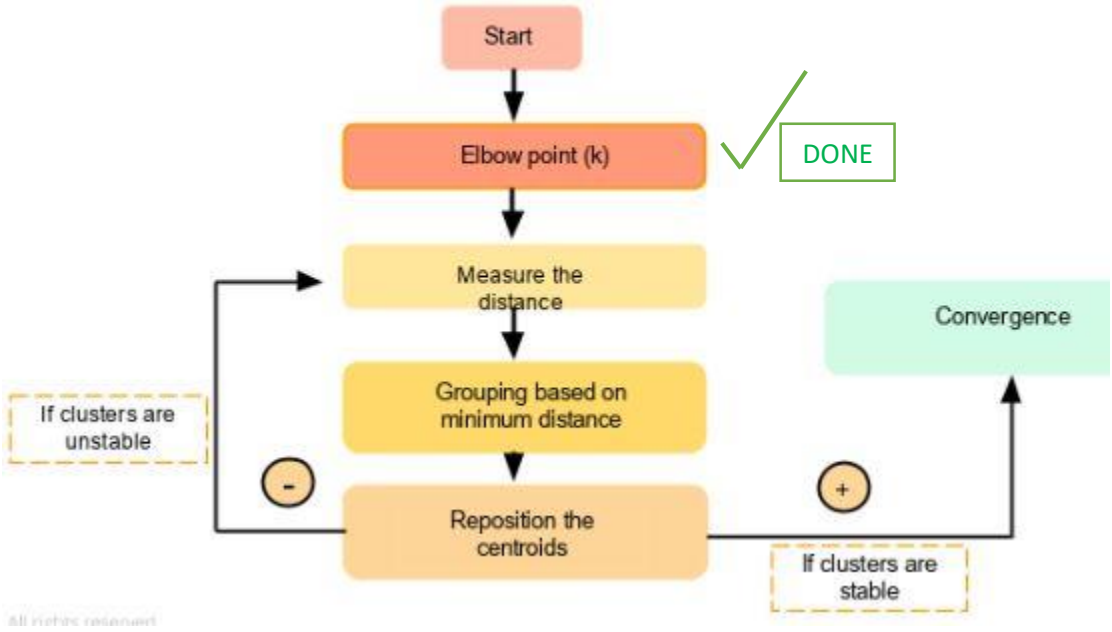
WSS will be measured for each value of k. The optimum k which is our number of clusters will be chosen like this. We plot a curve between WSS and k and for example it would be like this:



As you can see, there is a very small change from k=2. So the optimum number of clusters would be 2 or 3 and after that we won`t see any different but increasing the number of clusters for nothing!

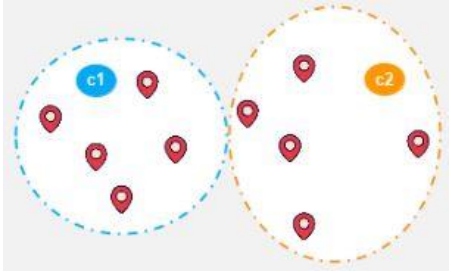
So what is the next Step?

Here`s a flowchart of the algorithm:

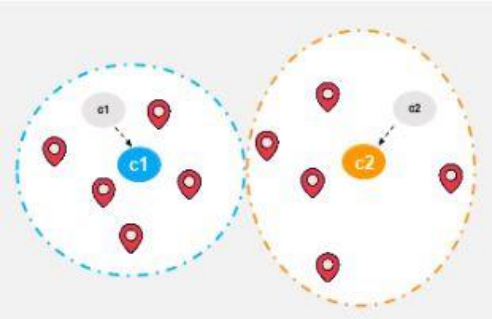


Now we randomly create k pairs of (x,y) which they`re the initial coordinates of our k clusters.

Now we calculate each point`s distance to the cluster centers and put each point(data) in the cluster with the minimum distance of it`s center. After putting all the points(data) in the clusters,



We update the centroids by getting the mean of the coordinates of each point in each cluster and the new centroids would be like this:



Now again we calculate the distance of each point to the centroids and put them in to cluster with the nearest center. We repeat this algorithm until the points won`t change their cluster and we can say the system is now converged.

At a glance the hole algorithm is:

Step 1: Choose K random points as cluster centers called centroids.

Step 2: Assign each x(i) to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)

Step 3: Identify new centroids by taking the average of the assigned points.

Step 4: Keep repeating step 2 and step 3 until convergence is achieved

So this is our K-means algorithm! ☺

١,٣.٣/٤-

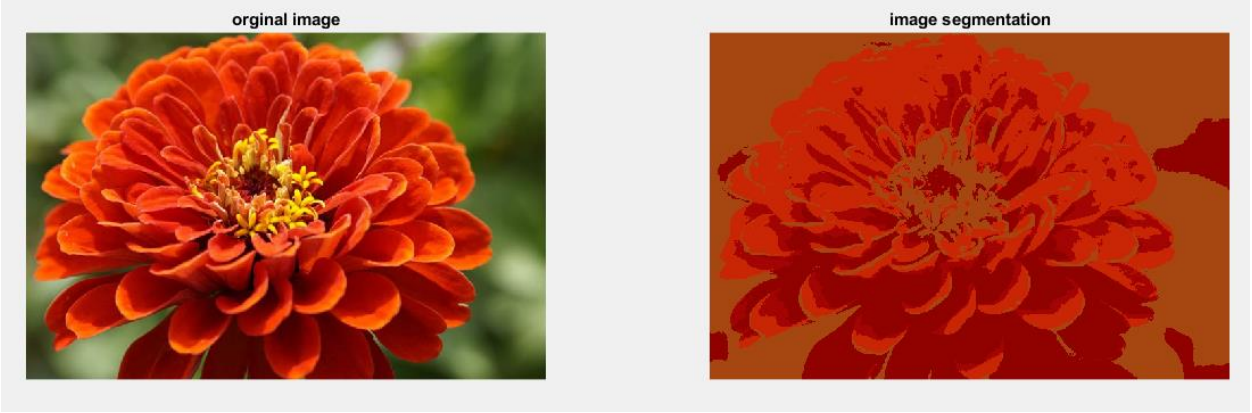
We don`t have to find the optimum number of clusters here and it is given to us. Here, it is 4.

If the random initial coordinates of centroids are far away from each other, we would have the best segmentation. For example here the random initial coordinate of centroids are:

```
x =

    13   221    76   441


y =

   482   338   318    41
```

As we can see the centroids are far from each other at first.

But if the initial centroids are close to each other, the segmentation happens but not as best as it can. For example:



In this example, the initial centroids probably are close to each other and all of their initial position is one the flower. This is why the segmented image is like that. Initial centroids:

```
x =

   193   217    55   266


y =

   153   260   394   170
```

- Here my algorithm works with the RGBs instead of coordinates. I use coordinates just at the first for getting the RGBs of the centroids and after that, the distance that I measure, is the Euclidean distance of the RGBs. RGB of each pixel will be aim to the centroid with the nearest RGB to that pixel. This algorithm runs until it converges. System converges when clusters don`t change their center and pixels don't change their cluster.

1.3.5- Otsu`s method:

This algorithm is used to do automatic image thresholding. This algorithm returns a threshold which lets us separate pixels in to 2 classes, foreground and background. This algorithm is equivalent to optimal K-means algorithm performed on the intensity histogram of the image.

The hole point of this algorithm is to find the threshold for binarizing it. So how?

The algorithm searches for a threshold that minimize the within-class variance, defined as a weighted sum of variances of two classes:

Minimize the *weighted within-class variance*:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Which the weights and the variances define as the equations below:

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \qquad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

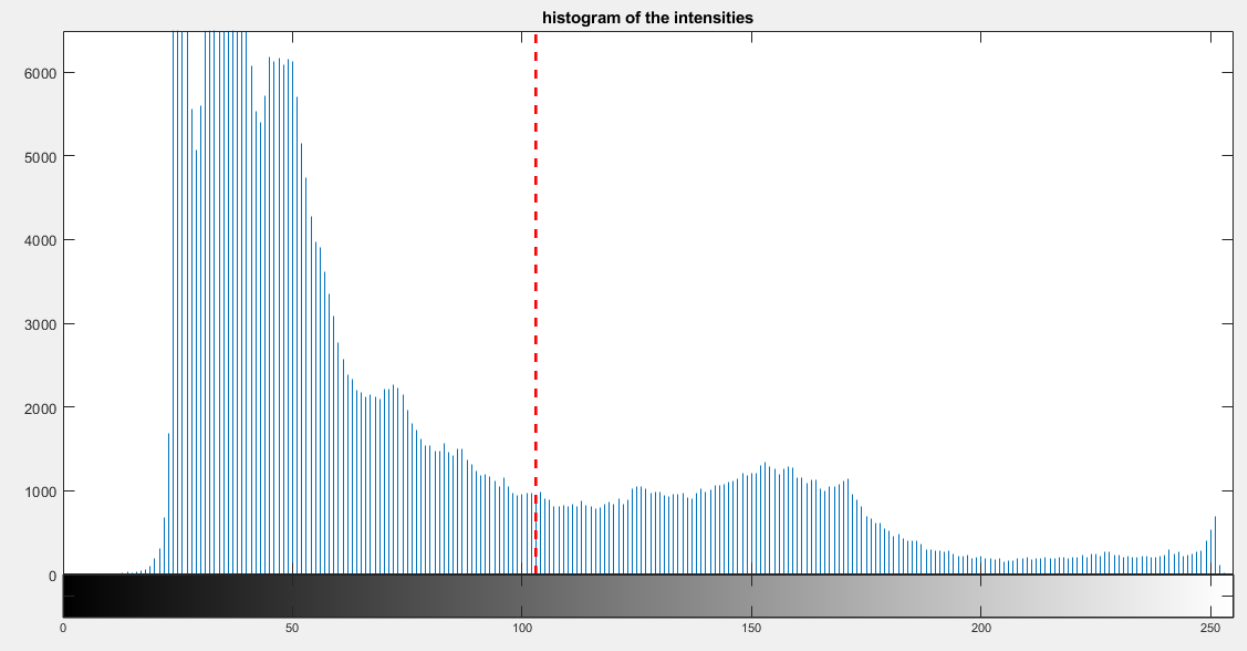
We can show by some maths, that minimizing the within-class variance is equivalent to maximizing between-class variance:

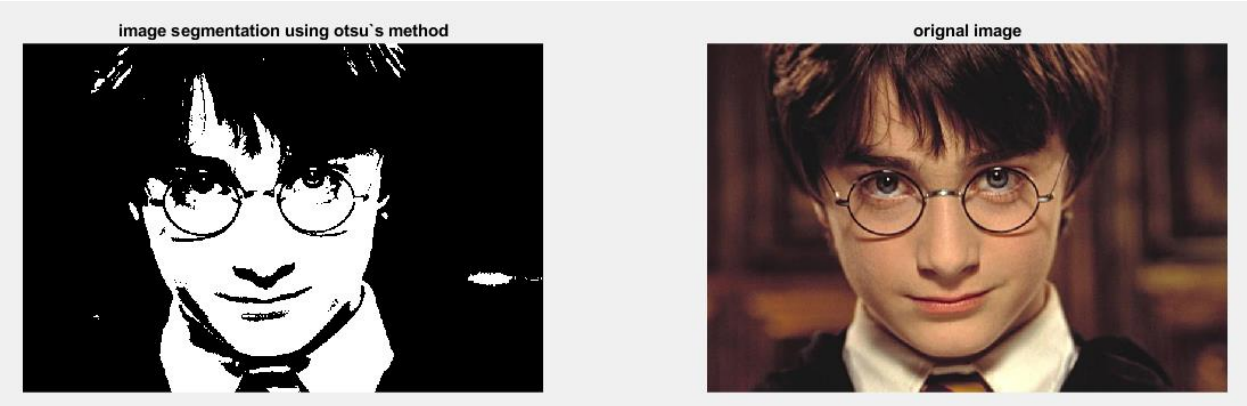
$$\sigma^2 = \underbrace{\sigma_w^2(t)}_{\text{Within-class}} + \underbrace{q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2}_{\text{Between-class}}$$

I`ve implemented the algorithm using maximizng the between-class variance.

The algorithm is that at each step we get the threshold something between 1 to 256 and calculate the between-class variance and at the end, the threshold which gives us the maximum between-class variance is the final threshold.

1.3.6-



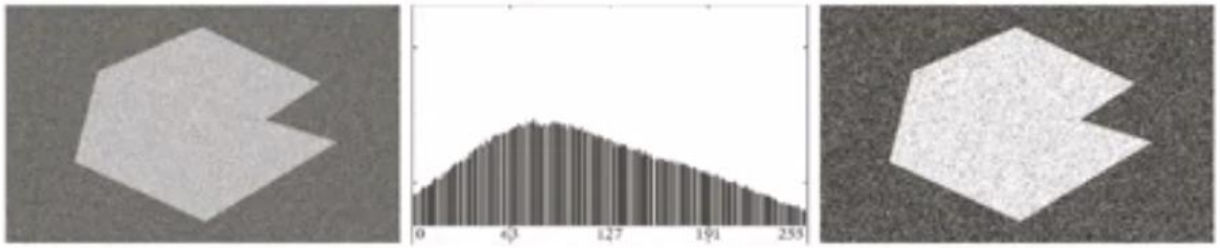


1.3.7- Otsu Vs K-means:

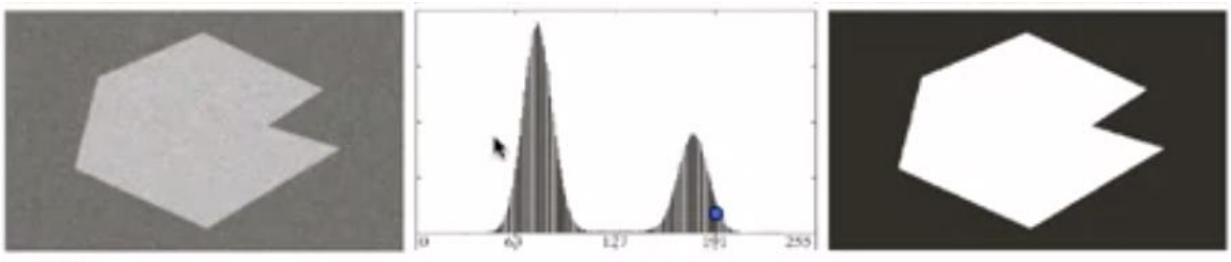
Actually, Otsu`s method is equivalent to K-means method in multilevel thresholding. They`re both based on minimizing the within-class variance as we discussed. But there is a difference which Otsu`s method is an exhaustive algorithm of searching the global optimal threshold which we saw that it check all available thresholds but K-means is a local optimal method. Moreover, Otsu`s method needs to compute a grayscale histogram of the image but K-means method doesn't need this. We can say K-means algorithm is more efficient because of multilevel thresholding than Otsu`s method.

One of the cons of the Otsu`s method is that we have 2 classes in this method but in K-means we have as much as cluster we need. What`s bad about this? When we have just 2 classes for every image, we will have a larger within-class variance which causes missing of weak objects in the image and maybe we need more than 2 classes.

Another cons of Otsu`s method is when we have a noise on the image. When we have a noise on image, the histogram will be affected and the Otsu`s method won`t work good. For Example:



As we can see the Otsu`s method has failed in here. So in Otsu`s method we need to pre-process the noisy image to remove the noise and then do the segmentation. This is what we don`t like. The pre processed of the image and Otsu`s method:



One of the cons of K-means algorithm is that finding the optimum number of clusters is sometimes hard.

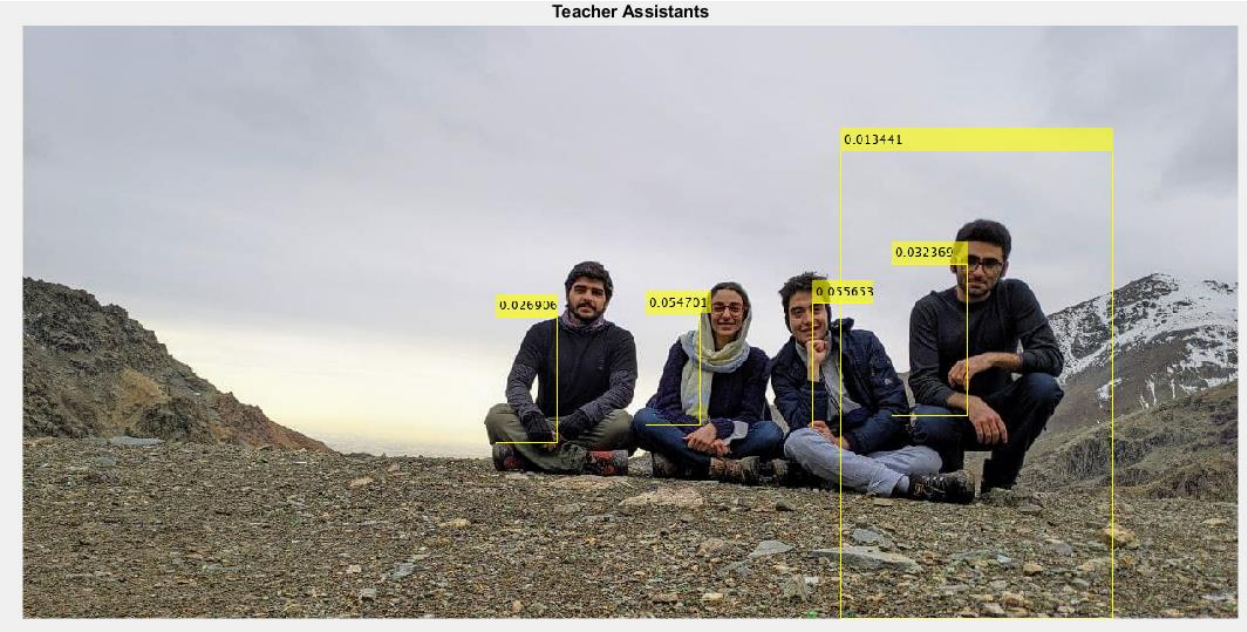
What`s good about K-means is that it`s suitable for large data sets and it uses circular clusters, So it will make changing efficient and flexible.

Another pros of the K-means is what is the cons of Otsu`s method. K-means doesn`t use the histogram for segmentation process and that`s why we can avoid the noise in the image.

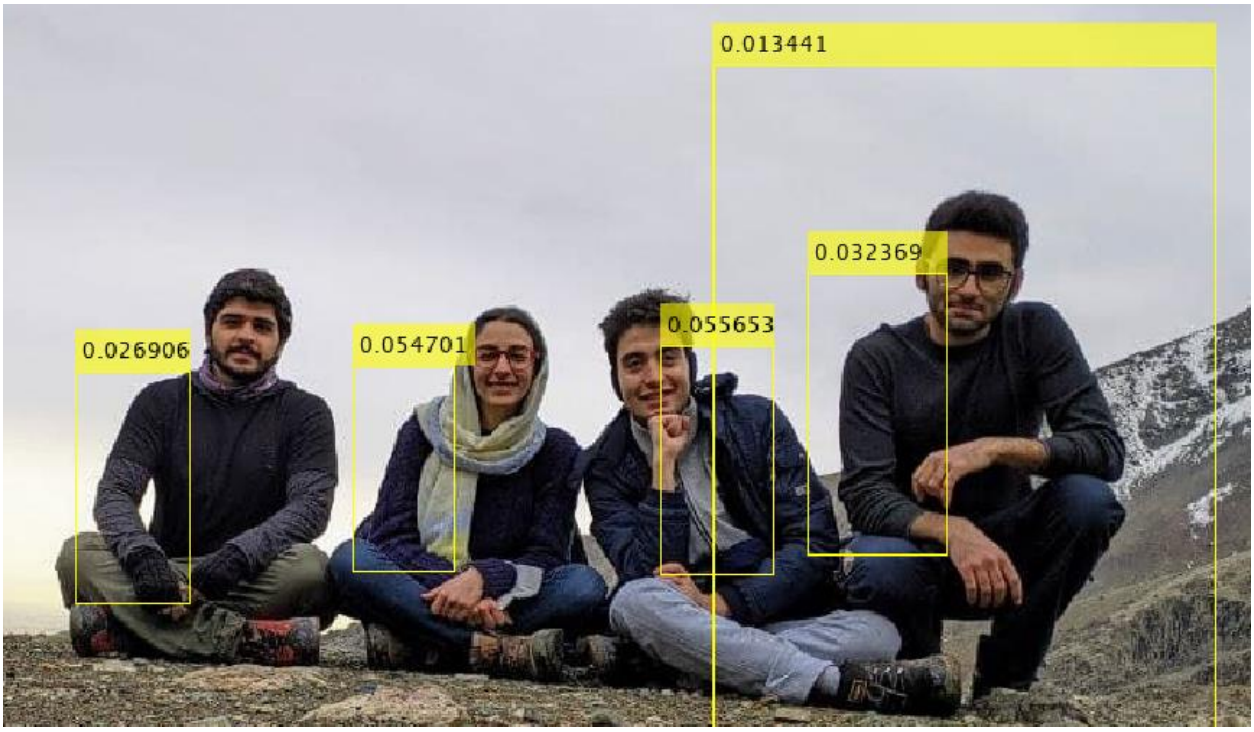
I think that's enough ☺

1.4- Object detection(Person Detection):

1.4.1 and 1.4.2-



Zoom a little bit:



As we can see all 4 TA`s are detected but we have a wrong detection which if I change the threshold It won`t be better.

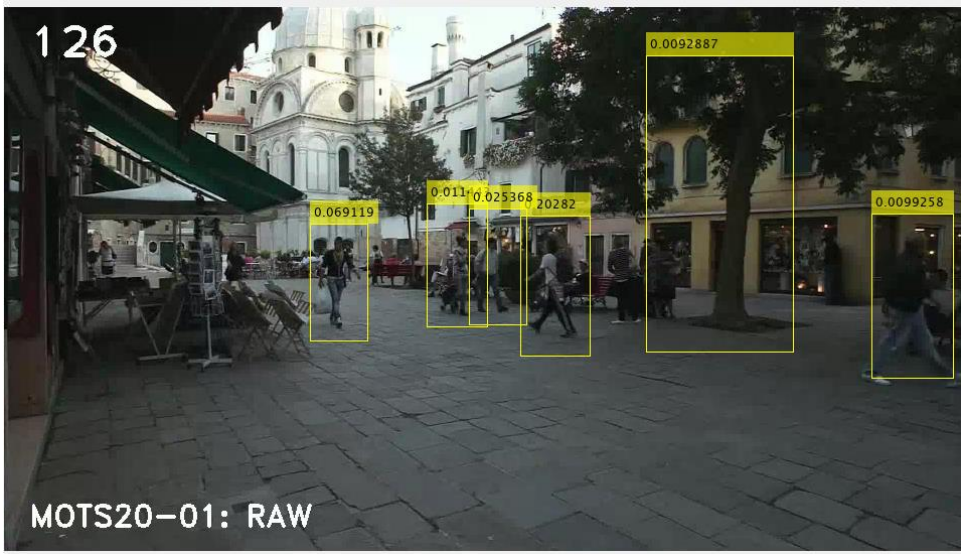
Actually, we can`t say it`s wrong but Matlab`s peopleDetector function can`t do it better.

1.4.3-

I use a rand function to peak 30 frames through out the 450 frames of the video.

The gif file has been attached to zip file named "`saved_gif.gif`".

- One screenshot:



Sometimes the tree will be detected as a person but with a very low confidence.

2- EEG signal processing:

2.1- Topography and power of channels:

2.1.1-

Remove the rest states:

```
% sampling frequency is 250Hz
% the first and the last 5 seconds is resting state and we remove them
data = data(1251:length(data)-1250,:);
time = time(2:12);
```

2.1.2-

zscore is a function for normalization the data. This function measures the distance between each data from the mean in terms of the standard deviation. This is called Standardization of the data which it has mean 0 and standard deviation 1 and it will keeps the shape of the data.(same skewness and kurtosis)

As we can see, we have a 20 seconds/ 5000 samples vector for each 19 channels that is our resting signal for each 19 channels and a 40 seconds/ 10000 samples that is our task signal for each of our 19 channels.

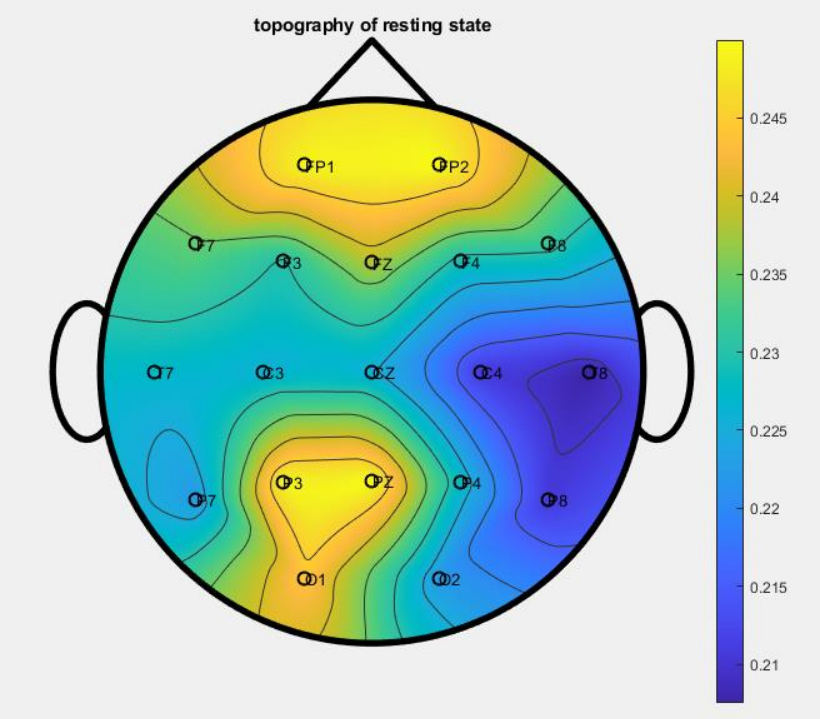
2.1.3- power of each channel when rest/task:

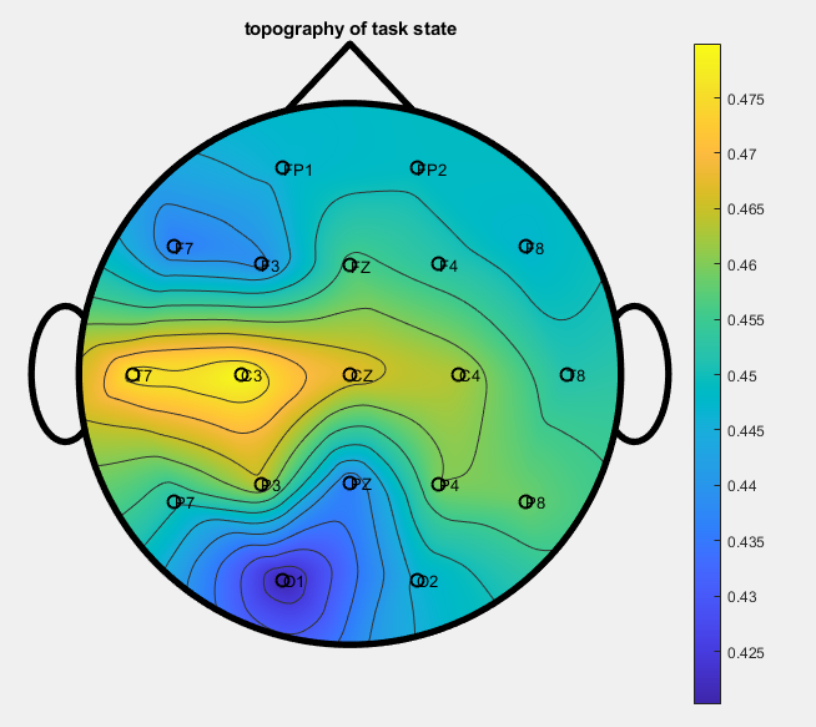
```
Prest =

  Columns 1 through 12

    0.2493    0.2525    0.2346    0.2296    0.2385    0.2286    0.2315    0.2271    0.2257    0.2254    0.2106    0.2075

  Columns 13 through 19

    0.2224    0.2493    0.2677    0.2274    0.2106    0.2444    0.2222


Ptask =

  Columns 1 through 12

    0.4468    0.4486    0.4356    0.4388    0.4563    0.4523    0.4470    0.4772    0.4853    0.4678    0.4631    0.4515

  Columns 13 through 19

    0.4510    0.4645    0.4357    0.4600    0.4588    0.4202    0.4461
```

We can see power of each channel in two rest and task states calculated by the equation below:

$$P = \frac{1}{N}\sum_{i=1}^{N} x[i]^2, N = length(x)$$

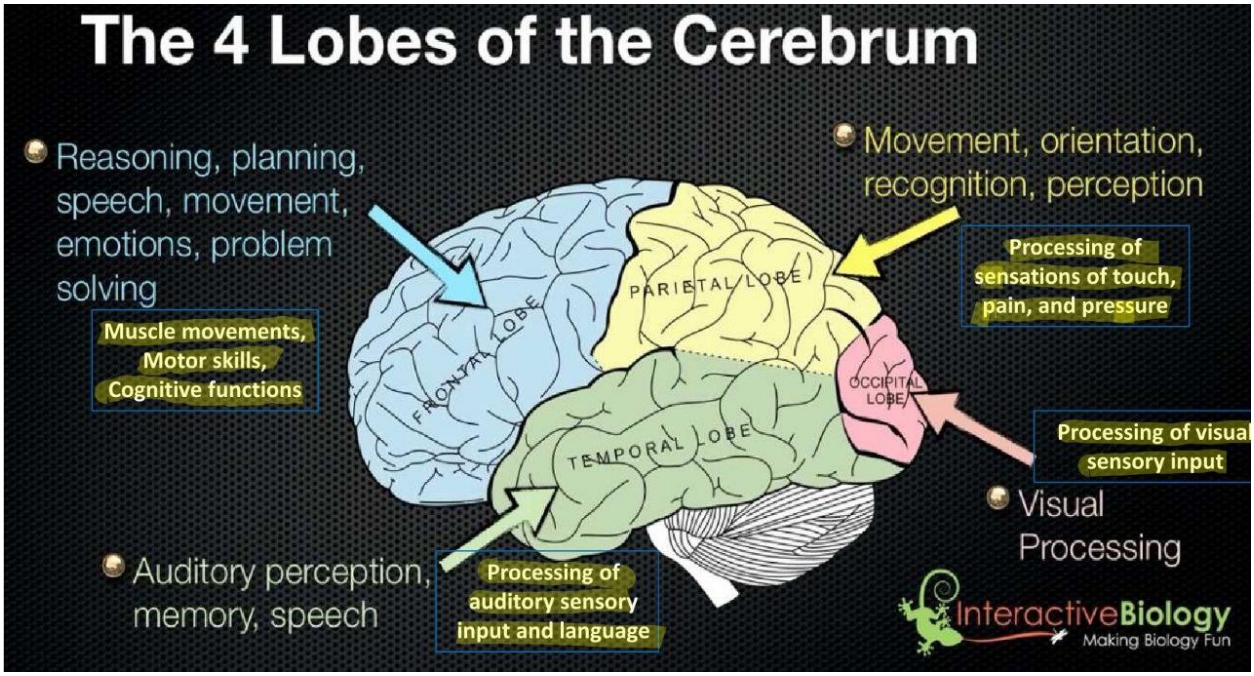
٢,١.٤-Topographies:





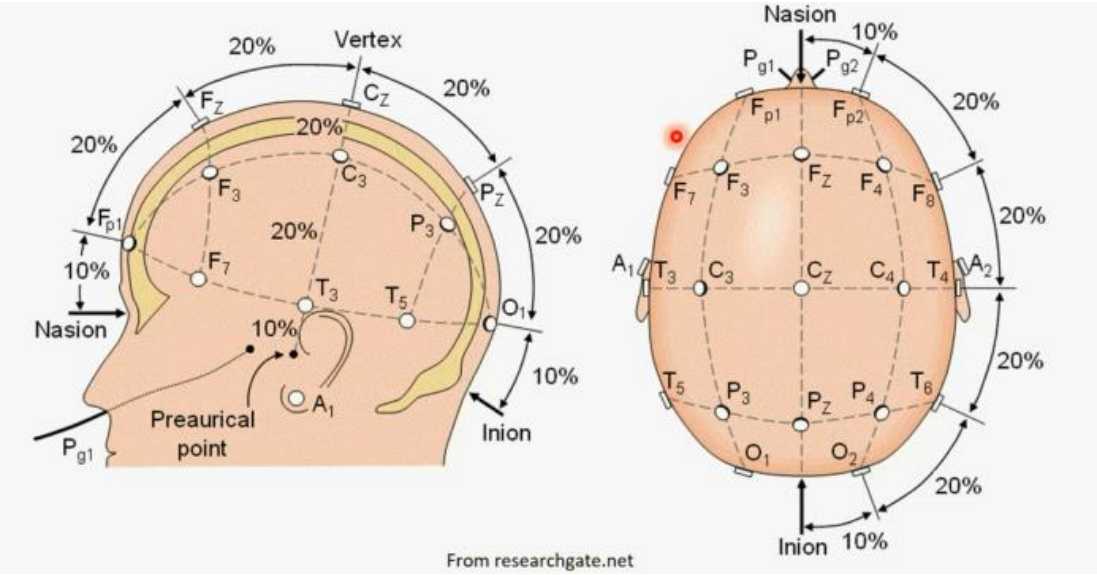
2.1.5-

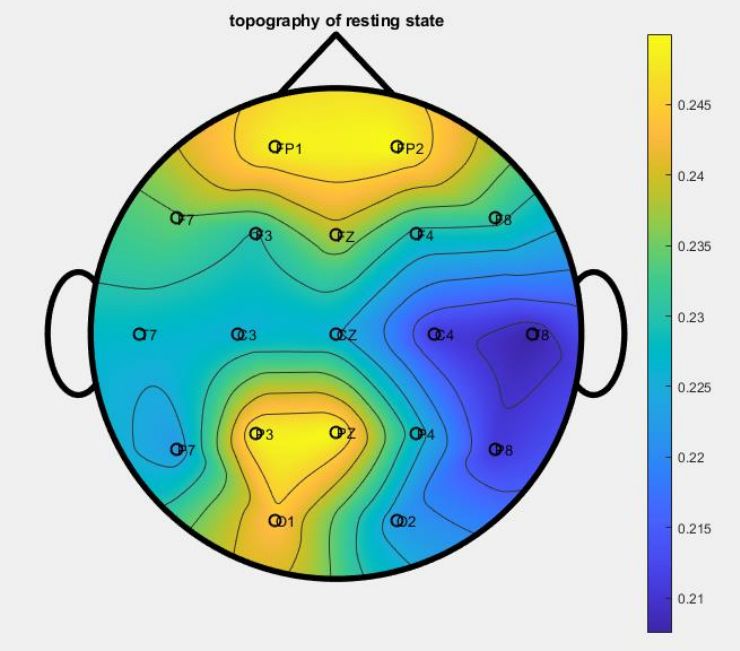
As we know, our task is an auditory task. When we`re at Rest state, we do nothing but just seeing and watching with our eyes. So at rest state we`re doing visual task.



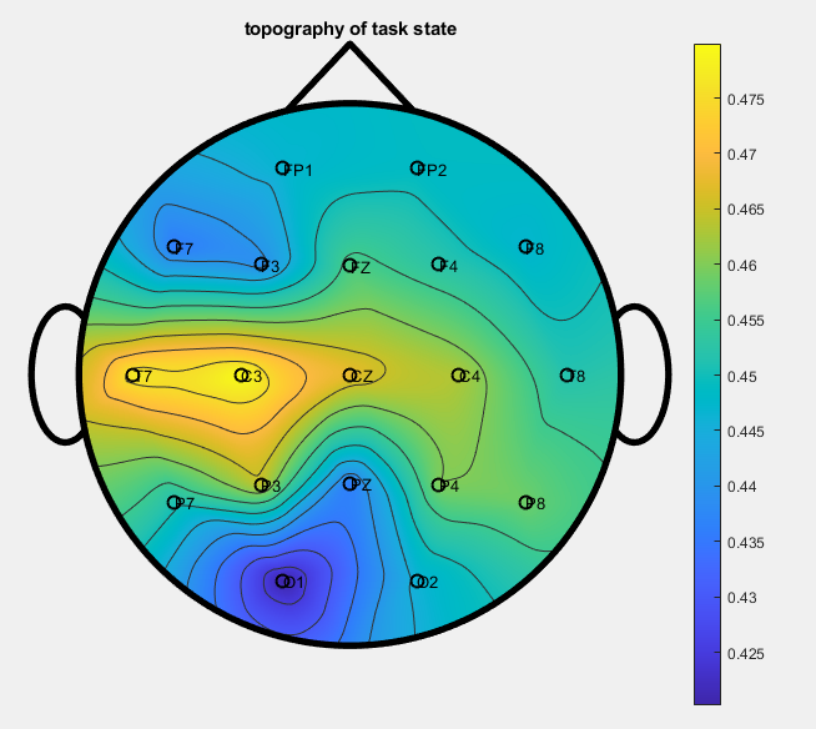
As we know our brain has 4 different lobes for different tasks.

We said when we are at resting state, we`re doing visual task because our eye is open and it`s watching unconsciously.
So we expect that the **Occipital lobe** which is related to visual processing and **Frontal lobe** which is related to muscle movements(here movement of eye muscles and muscles on top of the eyes), to be active. The topography is exactly showing us the high activity of FP1 and FP2 which are close to the Frontal lobe and O1 and O2 which are close the Occipital lobe.





When we`re at task state, we know the task is some auditory task and we`re watching around with our eyes simultaneously. So we expect the **Temporal lobe** to be very active because it is related to processing of auditory sensory input. So T channels have to be active. In addition to this, we have a visual task, too but not as much as the auditory task, because the focus of the person was probably a lot on the listening.



As we can see T7 and C3 are very active because of the auditory task. Probably the person was listening mostly with his/her left ear. Probably the sound was coming to that person from the left. But we can see the right ear is active too. As we can see we have visual task too but not as much as auditory task. We can see FP1, Fp2 and O1 that are active and this is because of the open eyes of the person.

2.2- Eye blinks counter:

2.2.1-

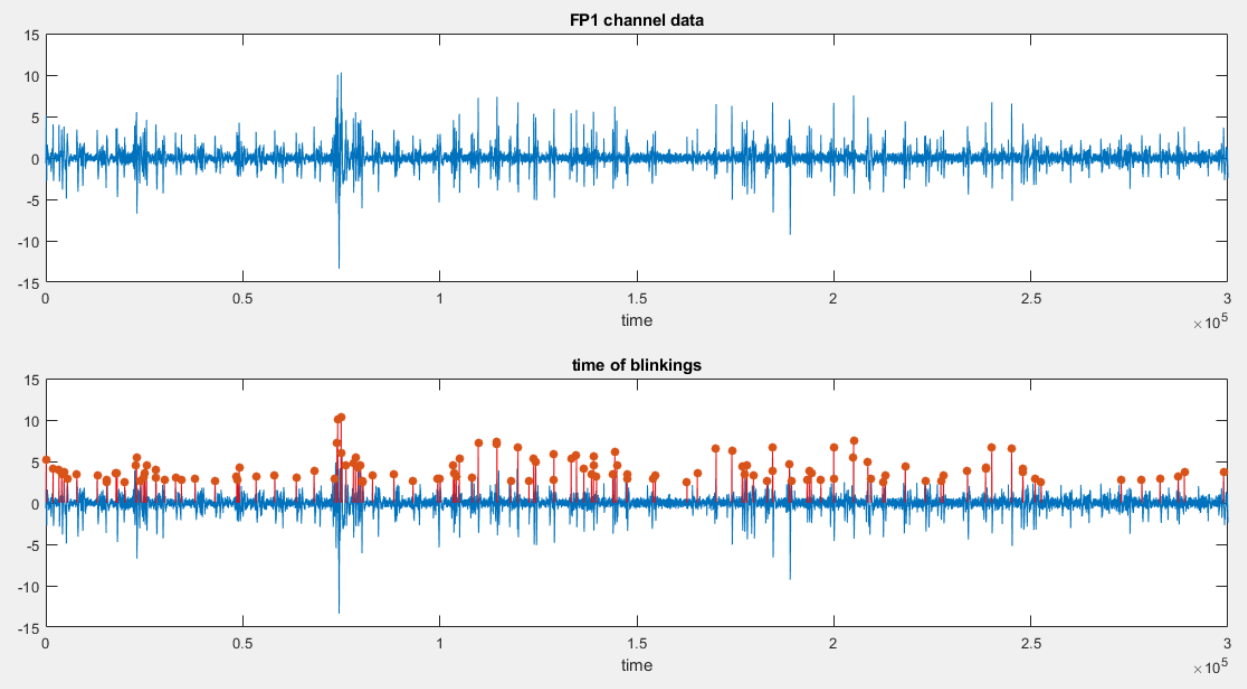
Done in the code.

2.2.2-

If the threshold is 2.5, we would count 123 blinks.

```
blinkNum =

   123
```

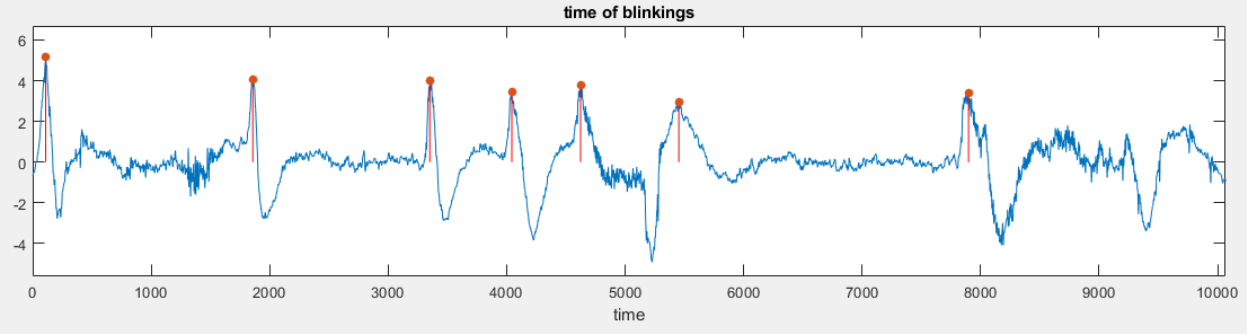
2.2.3-



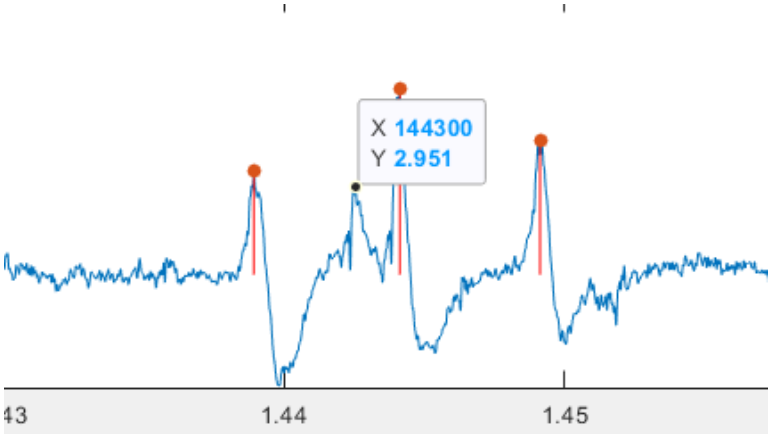
Why the window moving method gives us the number of blinks?

Actually answering this question would be easier if we had sample rate of the experiment but probably in this experiment, a window with length of 500 samples contains a very small time which a person can just blink once in it. So if we break up the data using a moving window with length of 500samples we can check if the person blinks or not by checking the maximum value of the data on that window and if it is more than the threshold, the person has blinked.

A closer look to the data:



As we can see, there is a blink in each window(Length=500). A question might come on your mind which the image below is a blink or not?



Actually before this, there is a blink and this rise is maybe because of the previous blink. So the accuarcy of counting is very good. For the best confidance of the experiment, we need to know the sample rate to find the best window length.