Computer Vision SBE 404B: Assignment 4

Asem Abdelaziz, Mohamed Abdallah, Taha Ali, Abdelrahman Sayed

PART1: THRESHOLDING

Implementation Policy

Python is used as a wrapper to perform matrix operations for edge detection and lines detection. *OpenCV* is only used for verifying our results, loading/showing images, and drawing lines.

Optimal Thresholding

The general overview of thresholding using *Optimal Thresholding*:

1) Extracting initial background and initial foreground.
2) Calculating the mean of both background and foreground.
3) Calculating average mean between background and foreground(*Threshold*)
4) Repeat until threshold(t+1) = threshold(t).
5) if pixel intensity $\geq$ Threshold then pixel intensity = 255 else pixel intensity = 0

GLOBAL OPTIMAL THRESHOLDING: IMPLEMENTATION

The globalOptimalThresholding function takes the input image to be thresholded using *Optimal Thresholding* and returns the thresholded image and optimal threshold.

Results

Using our implementation of global optimal thresholding we could produce the following images. see Figure 1, 2, 15, and 4.

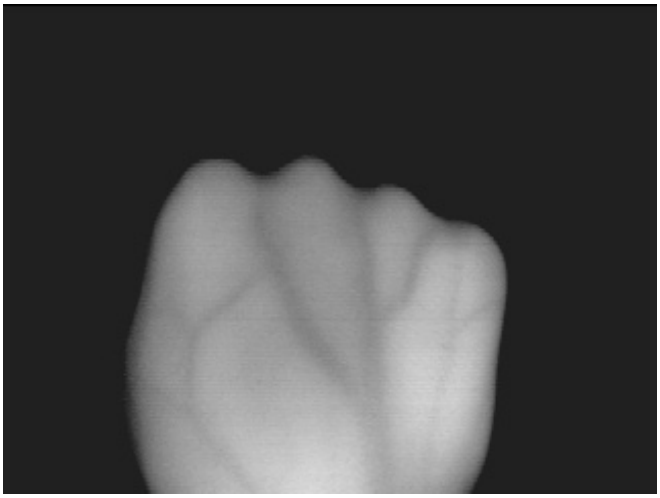

Fig. 1: Original image of interest (*0019hv3.bmp*)

Assignment is submitted to Dr. Ahmed Badawy, TA Eman Marzban, and TA Mohamed Hisham.

Fig. 2: Global thresholded image [optimal] (*Threshold = 95.51*)

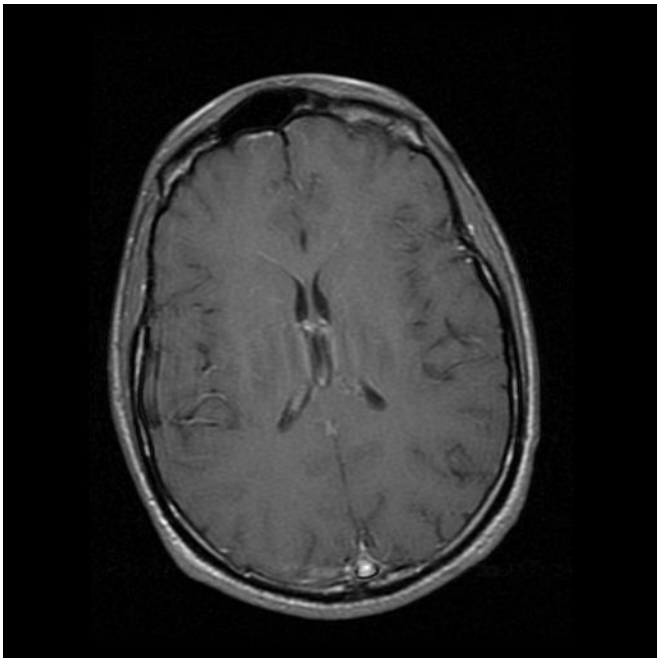

Fig. 3: Original image of interest (*MR1Brain2.jpg*)

LOCAL OPTIMAL THRESHOLDING: IMPLEMENTATION

Local optimal thresholding is based on dividing the images unit blocks and threshold each block independently by optimal method then add all the thresholded blocks to get the output thresholded image.

The localOptimalThresholding function takes the input image to be thresholded using *Optimal Thresholding* , block size returns the thresholded image.

Results

Using our implementation of local optimal thresholding we could produce the following images. see Figure 15, 6, 7, 8, 9, and 10.

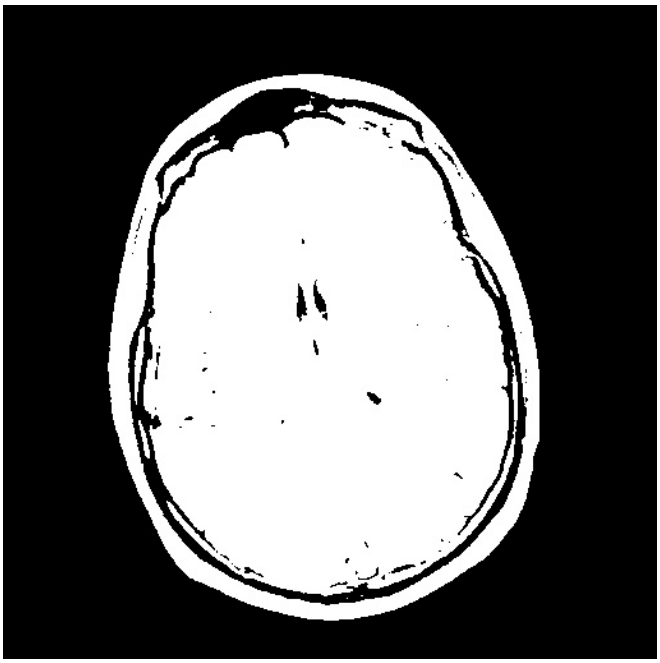

Fig. 4: Global thresholded image [optimal] (*Threshold = 48.49*)

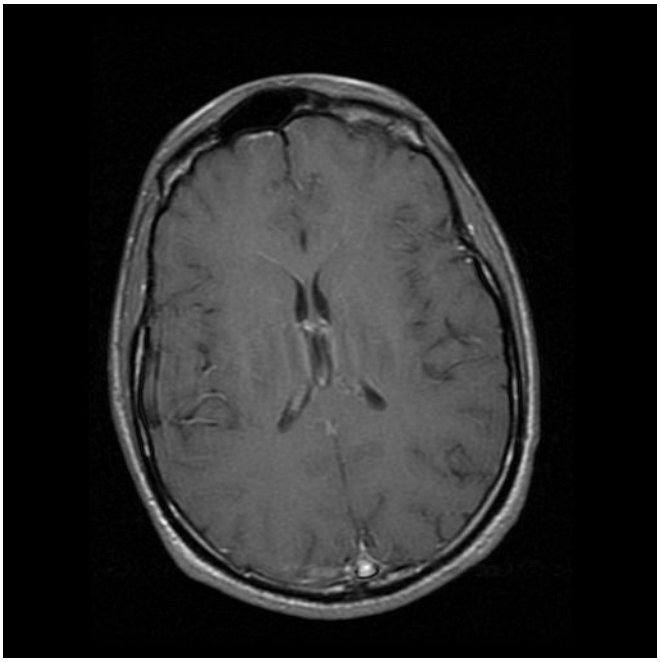

Fig. 5: Original image of interest (*MR1Brain2.jpg (512 X 512)*)

Otsu Thresholding

The general overview of thresholding using *Otsu Thresholding*: At each gray level(0:255) we do the following:

1) Calculating background weight and foreground weight.
2) Calculating background mean and foreground mean.
3) Calculating between class variance as follows :

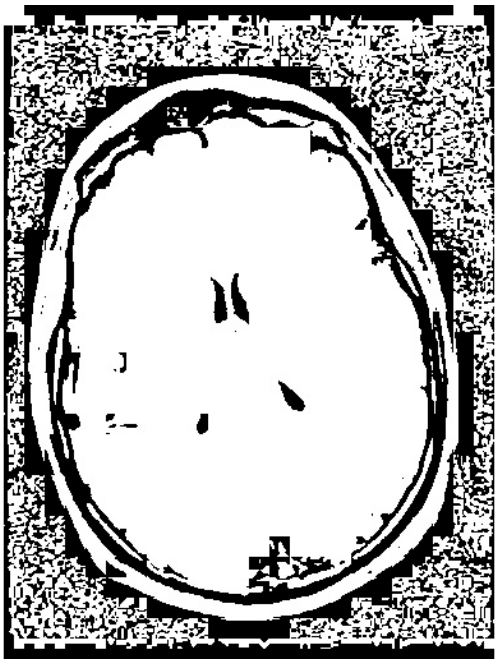

Fig. 6: local thresholded image [optimal] (*Block size 16*)

Fig. 7: local thresholded image [optimal] (*Block size 32*)

$$W_f W_b (meanf - meanb)^2 \tag{1}$$

4) Calculating maximum Between Class Variance (minimum within class variance).

The otsu threshold is the gray level corresponding to maximum Between Class Variance (Minimum Within Class Variance).

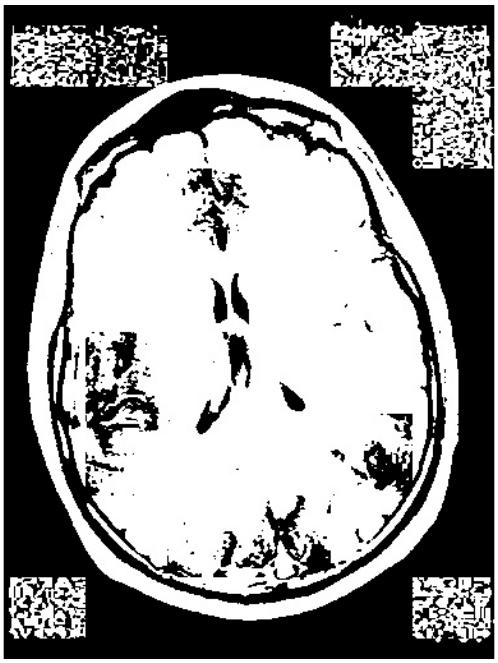

Fig. 8: local thresholded image [optimal] (*Block size 64*)

Fig. 9: local thresholded image [optimal] (*Block size 128*)

GLOBAL OTSU THRESHOLDING: IMPLEMENTATION

The globalOtsuThresholding function takes the input image to be thresholded using *Otsu Thresholding* and returns the thresholded image and otsu threshold.

Results

Using our implementation of global otsu thresholding we could produce the following images. see Figure 11, 12, 15, and 14.

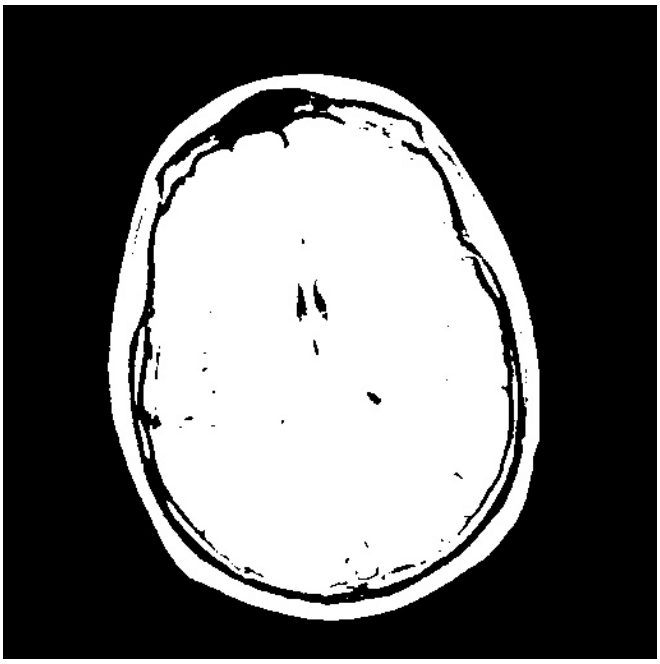

Fig. 10: local thresholded image [optimal] (*Block size 256*)

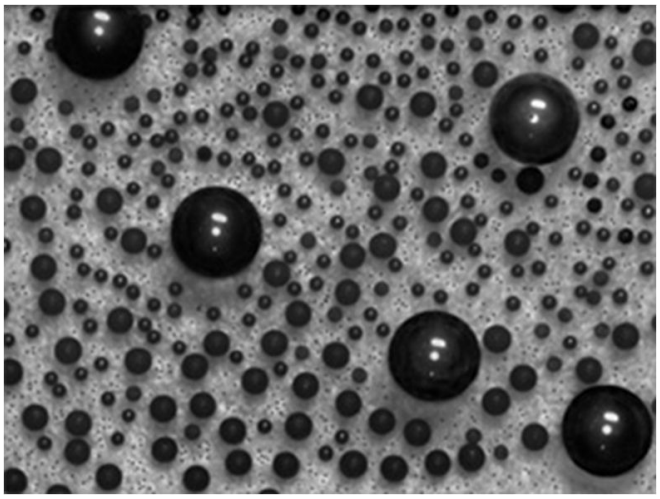

Fig. 11: Original image of interest (*Beads.jpg*)

LOCAL OPTIMAL THRESHOLDING: IMPLEMENTATION

Local Otsu thresholding is based on dividing the images unit blocks and threshold each block independently by otsu method then add all the thresholded blocks to get the output thresholded image.

The localOptimalThresholding function takes the input image to be thresholded using *Optimal Thresholding* , block size returns the thresholded image.

Results

Using our implementation of local otsu thresholding we could produce the following images. see Figure 15, 16, 17, 18, 19, and 20.

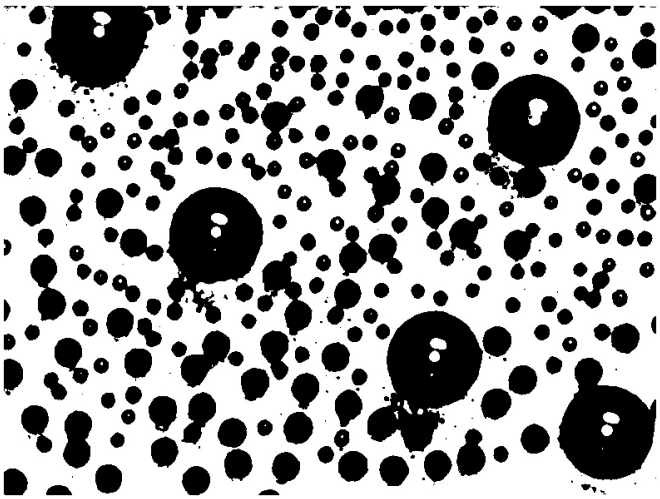

Fig. 12: Global thresholded image [otsu] (*Threshold = 90*)

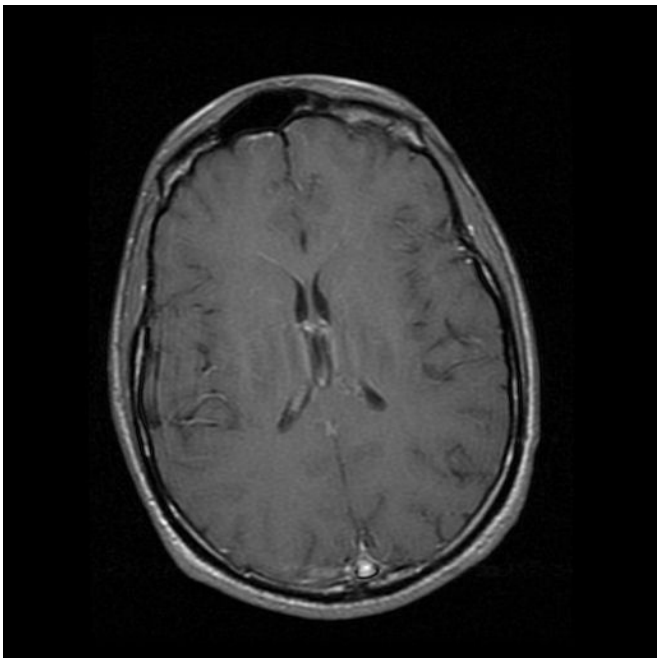

Fig. 13: Original image of interest (*MR1Brain2.jpg*)

I. Part2: Segmentation

K-means segmentation

This type of segmentation is based on the traditional k-means clustering algorithm. In this k-means implementations, we cluster the image RGB components in the RGB color-space.

***Results**:* Here are some of the results, in figure 21

Regional Growing segmentation

This type of segmentation is based on the Regional Growing segmentation algorithm. In this implementations, we cluster the image RGB components in the RGB color-space. The colors used to identify regions are chosen randomly.

***Results**:* Here are some of the results, in figure 22

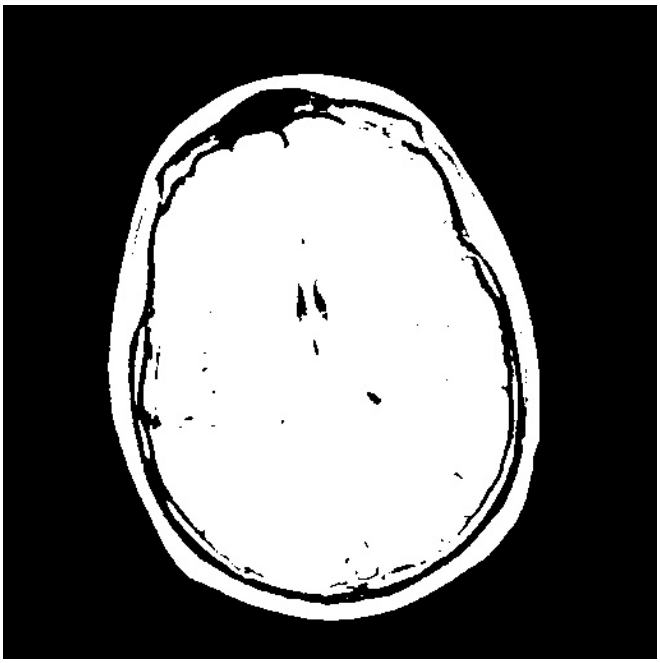

Fig. 14: Global thresholded image [otsu] (*Threshold = 49*)

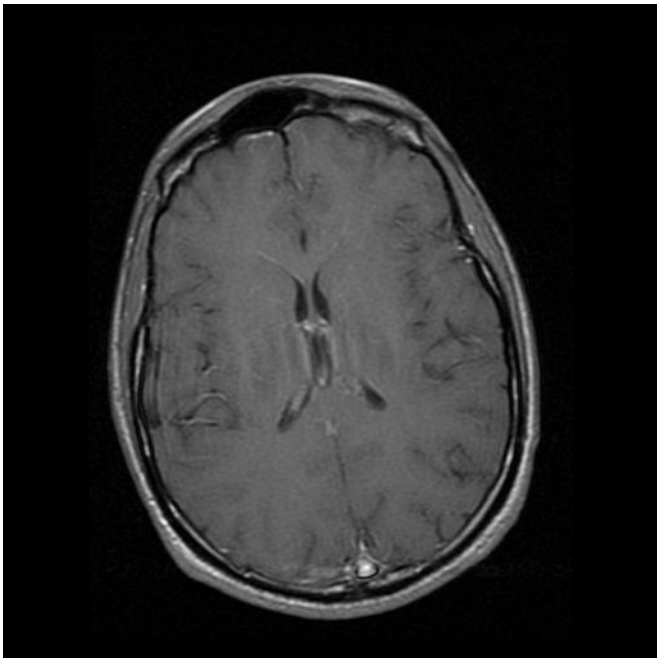

Fig. 15: Original image of interest (*MR1Brain2.jpg (512 X 512)*)

Mean Shift segmentation

This type of segmentation is based on the Regional Growing segmentation algorithm. In this implementations, we cluster the image RGB components in the UV color-space and the GB color-space. The colors used to identify regions are chosen randomly.

Results*:* Here are some of the results, in figure 24

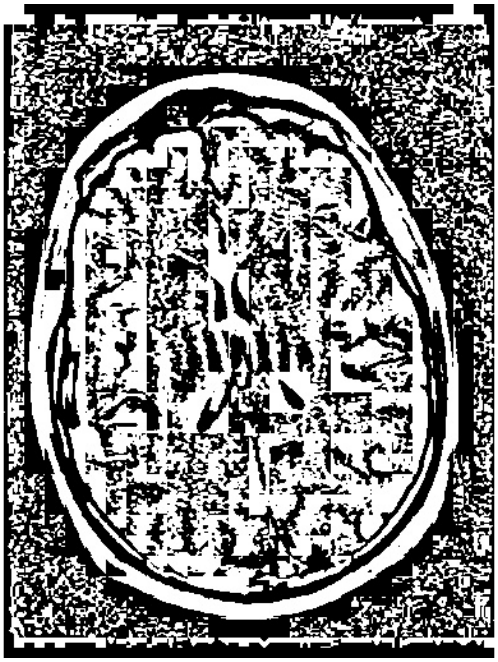

Fig. 16: local thresholded image [otsu] (*Block size 16*)

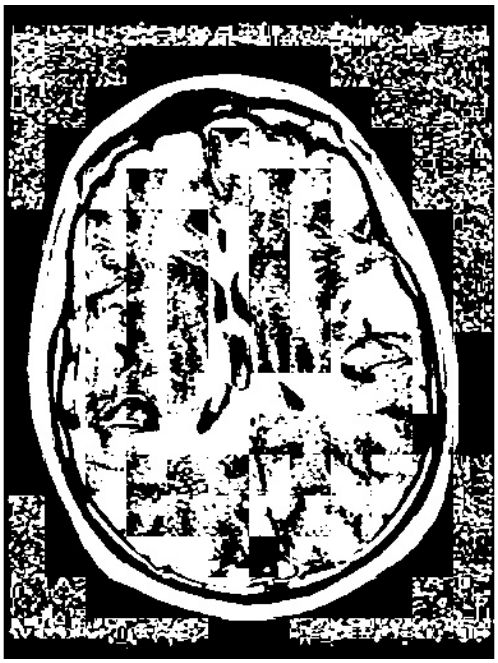

Fig. 17: local thresholded image [otsu] (*Block size 32*)

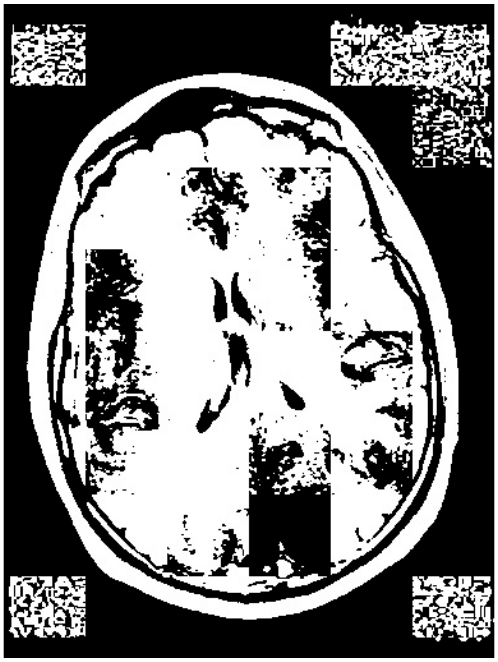

Fig. 18: local thresholded image [otsu] (*Block size 64*)

Fig. 19: local thresholded image [otsu] (*Block size 128*)

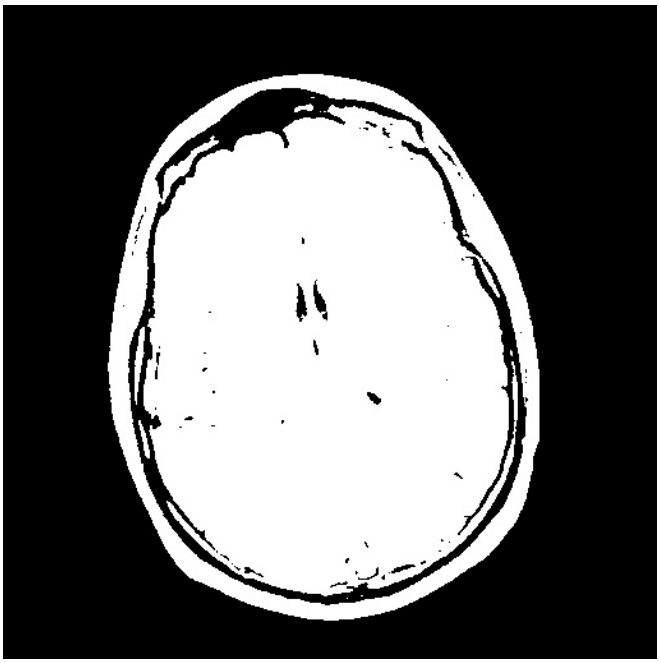

Fig. 20: local thresholded image [otsu] (*Block size 256*)

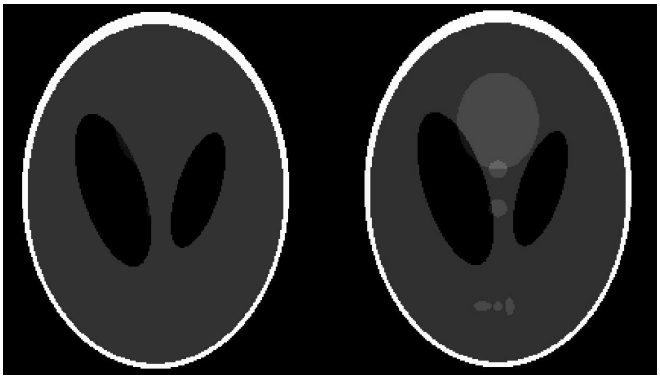

Fig. 21: K-mean clustering to 4 clusters. The one on the right is the un-clustered, the other is th clustered one

Fig. 22: Image is segmented using regional growing

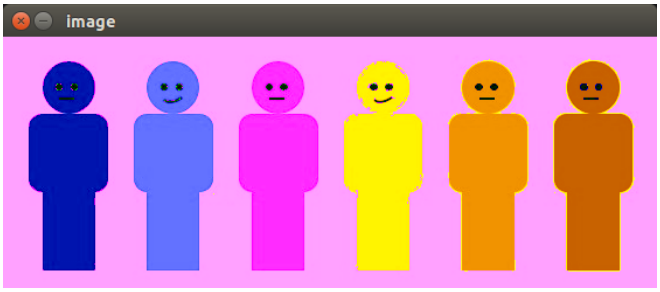

Fig. 23: Image is segmented using mean-shift , 4 clusters, UV color-space

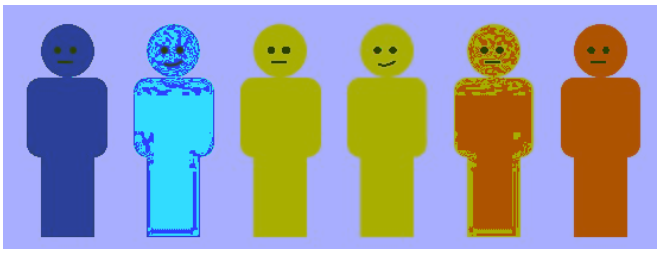

Fig. 24: Image is segmented using mean-shift , 4 clusters, GB color-space

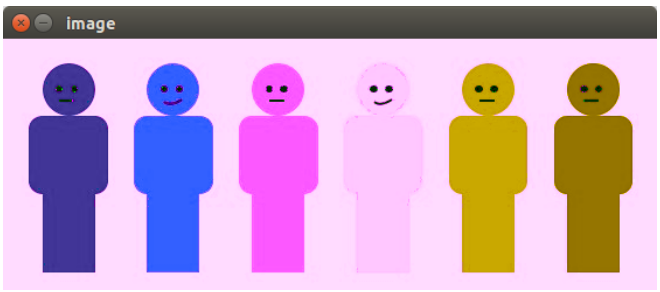

Fig. 25: Image is segmented using mean-shift, 64 clusters, LUV color-space