CENG 466 Fundamentals of Image Processing HW3

Andaç Berkay Seval
2235521

Asrın Doğrusöz
2380301

***Abstract*—This report is prepared for the third homework of CENG 466.**

I. Introduction

This report includes methodology and rationale behind the algorithms that are implemented in the homework, difficulties and errors encountered in the design, implementation, experimentation stages and their solutions, analysis and comments on the results and the requirements of the code.

II. Requirements

This homework is done with the help of some libraires. Therefore, there are 3 requirements for the code:

- scikit-image (skimage)
- opencv (cv2)
- numpy

The details and the purposes of these libraries will be explained in the next section.

III. Implementation

A. Face Detection

Firstly, we read 2 papers about extracting skin colors in different color spaces. First one is called "Skin Detection Based on Image Color Segmentation with Histogram and K-Means Clustering". It applies a methodology to image segmentation. In the first step, it applies a threshold to RGB image to get rid off background pixels. Then, it converts RGB image to YCbCr color space and apply K-Means clustering to that image. Moreover, with experiments, it extracts skin colors based on YCbCr color space. Second paper is called "Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space". It has a 2 methodologies for skin color detection. First one is starting with converting RGB image to HSV color space. Then, histograms are computed for H, S and V components. With the help of histograms, threshold values are determined. After that, masking is applied for skin pixels. Then, threshold is applied to masked image. Threshold image is smoothened and filtered. Finally, output image only contains skin pixels. Second methodoly purely based on threshold values. Firstly, it converts RGB image to YCbCr color space. Then, with the data obtained from experiments, a threshold is applied to image such that if Cb component of the pixel $100 < Cb < 150$ keep that value, otherwise discard it and, if Cr component of the pixel $150 < Cr < 200$ keep that value, otherwise discard it. These cluster values correspond to skin colors in most the the images. The second methodoly in the second paper looks very promising. Thus, we decided to implement and test it.

Detect faces function is implemented with the help of opencv and skimage libraries. The hardest faces to capture is in image 1 probably due to the color values of the photograph. It is an old picture and it is hard to cluster. In order to cluster it for face detection more easily, histogram equalization is applied to image. Thus, RGB color values of the image is flattened. Then, histogram equalized image is converted to YCbCr color space since color clustering is easier in that color space rather than RGB color space. After that, in the

Fig. 1. Image 1 in YCbCr color space

image, if Cb component of the pixel $99 < Cb < 151$ and Cr component of the pixel $152 < Cr < 171$ keep the values of the image, otherwise paint pixel with (16,128,128) which corresponds to black in RGB color space. After that, image is converted back to RGB space, which becomes a masked image. Then, convert RGB image to gray scale and apply threshold to obtain a binary image from gray scale image.

Lastly, find contours in binary image and draw rectangles to corresponding regions if the region size is big enough for the faces.

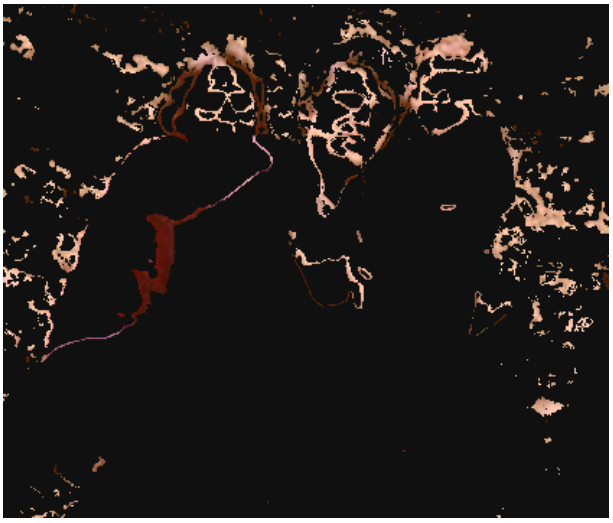

Fig. 2. Masked Image 1

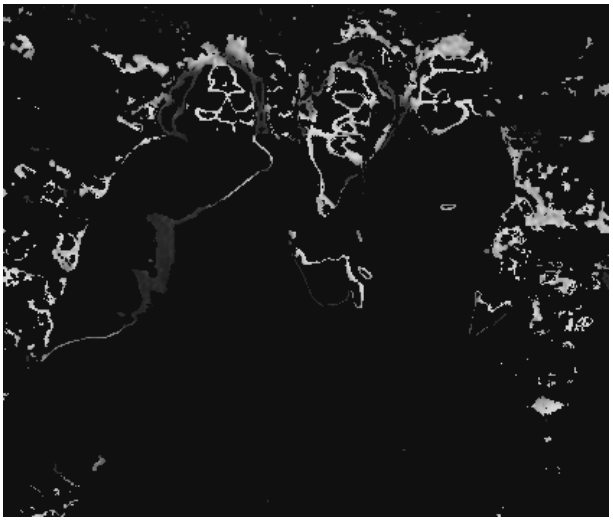

Fig. 3. Gray Scale Image 1

For the image 2 and image 3, all the process is the same with image 1 except histogram equalization is not applied to image 2 and image 3 since the faces in those images could be found easily without a need for external intervention. Furthermore, for both image 2 and image 3, Cb component of the pixel should be $100 < \text{Cb} < 150$ and Cr component of the pixel $152 < \text{Cr} < 190$ for skin colors. These values for skin colors are different from image 1 and found with experiments and paper results.

We managed to detect faces in all 3 images with our implementation. Actually, we obtained a pretty good result with our function. Our implementation could find faces in images if the skin colors are in the range of our experiment results as papers suggested. However, of course improvements can be done in the function.

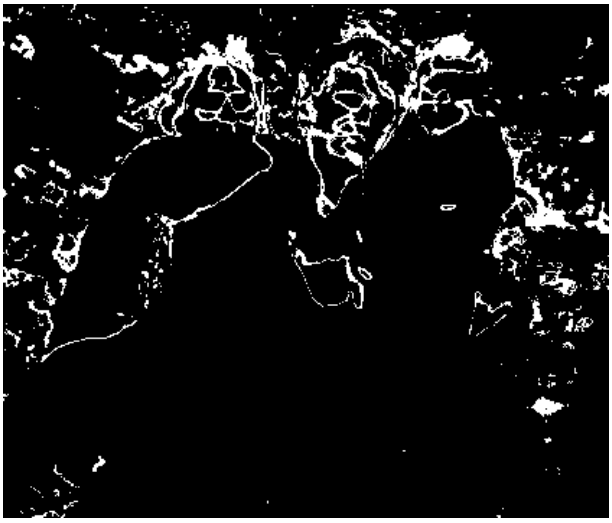

Fig. 4. Binary Image 1

Fig. 5. Image 2 in YCbCr color space

B. Pseudo-coloring

Pseudo-coloring is a method that maps colors obtained from a colorful source image to every gray value of a gray scale image. Therefore, gray scale image turns into a colorful image. Hence, this new colored image can make identification of certain features of image easier for observers. Thus, let's propose an algorithm for pseudo-coloring. Pseudo-code for pseudo-coloring:

Fig. 6. Masked Image 2

Fig. 7. Gray Scale Image 2

```
gray image = read(gray image)
list = []
-for i = 0:gray image dimension 1
–    for j = 0:gray image dimension 2
–        if gray image[i,j] not in list
–            list.append(gray image[i,j])
–    end
-end
list.sort() (sort gray values)
K = n (number of color clusters in the colorful source image)
list.partition(n) (partition list to n different lists to color mapping)
source image = read(source image)
labels, centers = K-meansClustering(K, source image) (Apply K-means clustering for K clusters to colorful source image to get center colors)
colored image = allocateSpace(dimension = (gray image dimensions, 3))
-for i = 0:gray image dimension 1
–    for j = 0:gray image dimension 2
–        if gray image[i,j] in listn (nth partition of the list)
–            colored image[i,j] = centers[n] (map center colors obtained after K-means clustering to source image to gray values in gray scale image)
–    end
-end
writeImage(colored image)
```

Therefore, after that pseudo-code, algorithm is implemented in python. Color images algorithm is implemented with the help of opencv, skimage and numpy libraries. For images 1,2 and 3, algorithm is implemented exactly as pseudo-code. Firstly, gray scale image is read. Then an empty list is created. After that, for image dimensions, if a certain pixel value is not in the list, we add to the list. Then, we sort the gray values in the list to get ready them for color mapping. After that, we partition the list to n lists where n is the cluster number that we will apply to source image. Then, we read source image and apply K-means clustering with n clusters. After that, for out image which is colored version of the gray scale image, for image dimensions, if particular pixel value of gray scale image is in nth partition of the list, that particular pixel in the out image is equal to nth center value (RGB) of the source image. Thus, color mapping is done. Hence, gray scale image becomes a colored image after this process. Furthermore, thanks to pseudo-coloring, we can examine different details of the gray scale image that we could not appreciate before coloring process. In fact, all the gray scale image came to life after pseudo-coloring. We could realize lots of different features of the original images after this process. Also, our algorithm that is implemented outputs good results for all the images. Applying K-means clustering to source image and mapping center color values to gray scale image rather than mapping actual colors of the source image

Fig. 8. Binary Image 2

Fig. 9. Image 3 in YCbCr color space

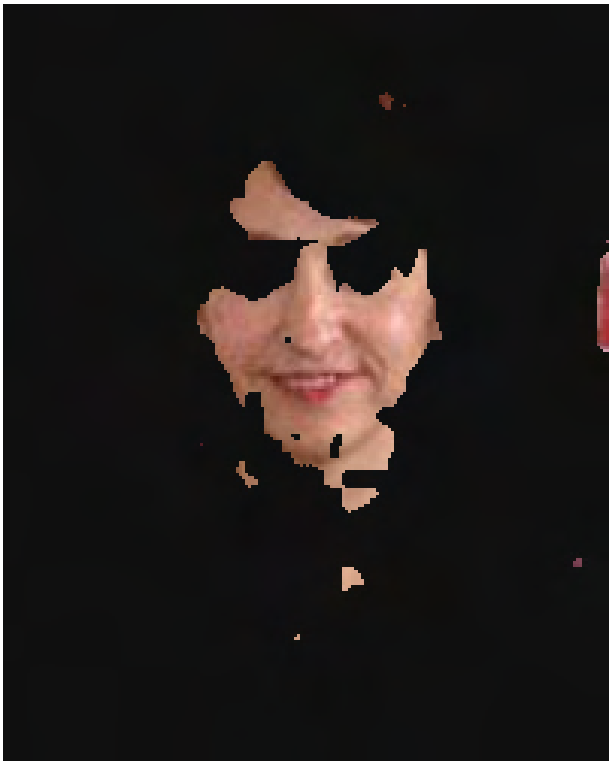

Fig. 10. Masked Image 3

Fig. 11. Gray Scale Image 3

is a clever choice since it would get cumbersome depending on the images. For example, for image 1, it has 239 different

Fig. 12. Binary Image 3

gray values and, for source image 1, it has 109744 different RGB values. Therefore, mapping between these values could be hard. Thus, applying K-means clustering is a good choice to source image. However, partitioning gray values is done assuming uniform distribution of the gray values. Therefore, a more clever approach could be done for partitioning gray values of the original images. For image 4, since the shapes of the gray scale and source images are the same, direct color mapping is used to color gray scale image.

Fig. 13. K-means clustered source image 1

Fig. 14. K-means clustered source image 2

Fig. 15. K-means clustered source image 3

Finally, let's compare RGB and HSI channels of the pseudo-colored images.

Fig. 16. Red channel of pseudo-colored image 1

Fig. 17. Green channel of pseudo-colored image 1

Fig. 18. Blue channel of pseudo-colored image 1

Fig. 19. Hue channel of pseudo-colored image 1

Fig. 20. Saturation channel of pseudo-colored image 1

Fig. 21. Intensity channel of pseudo-colored image 1

For images, when we look at RGB channels and HSI channels, RGB channels represent the coordinates of three primary colors of red, green and blue, HSI representation decomposes the image into Hue, saturation and intensity components.

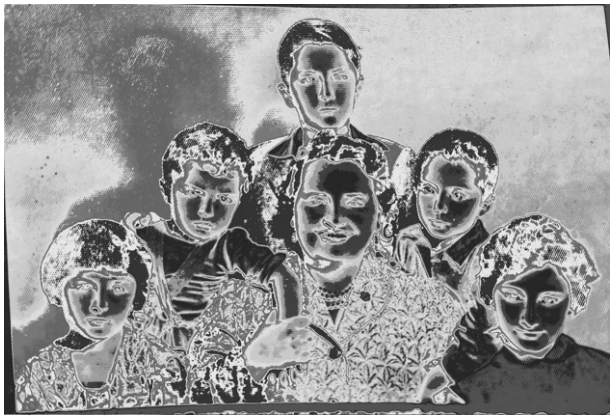

Fig. 22. Red channel of pseudo-colored image 2

Fig. 23. Green channel of pseudo-colored image 2

Fig. 24. Blue channel of pseudo-colored image 2

Also, intensity channel carries most of the information about the image in HSI representation. Moreover, HSI generates a perceptual space, which is more consistent to the human visual system than RGB color space. Thus, image processing algorithms can be designed more suitable with HSI color system.

Fig. 25. Hue channel of pseudo-colored image 2

Fig. 26. Saturation channel of pseudo-colored image 2

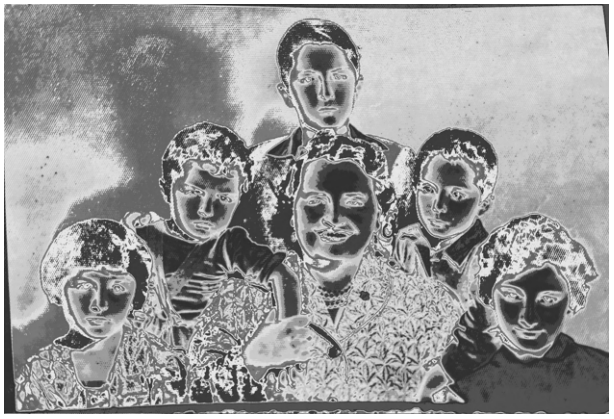

Fig. 27. Intensity channel of pseudo-colored image 2

C. Edge Detection

Detect edges function is implemented with the help of numpy and opencv libraries. Firstly, Sobel filters for vertical and horizontal edges are implemented for gradients. Then, for all images 1,2 and 3, they are splitted to R, G and B channels and for each channel, convolution is applied with horizontal and vertical Sobel filters separately. After that, each channel is merged to obtain a colored edge filtered images.

Fig. 28. Red channel of pseudo-colored image 3

Fig. 29. Green channel of pseudo-colored image 3

Secondly, images are converted to HSV color space from RGB. Hence, H, S and V channels are splitted and each channel is convoluted with Sobel filters separately. Moreover, channels are merged to obtain a HSV colored edge filtered image. For comparing the result, we thought that in RGB space, detected edges are more meaningful to human eyes. In HSV color space, detected edeges are more hard to catch for human visual system. However, the reason behind that may be edge detection in colored image. We did not convert these images to gray scale.Instead, we applied gradient filters to each channel separately and merge them to obtain a colorful edge detected images. Furthermore, application areas for edge detected RGB ans HSV images can be different, but RGB images look nicer for human eyes.

REFERENCES

[1] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453.

[2] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[3] Harris, Charles R. and Millman, K. Jarrod. "Array programming with NumPy". Nature. 2020. doi: 10.1038/s41586-020-2649-2

[4] Bradski, G. Dr. Dobb's Journal of Software Tools. "The OpenCV Library". 2000.

[5] Emir Buza, Amila Akagic, Samir Omanovic, "Skin Detection Based on Image Color Segmentation with Histogram and K-Means Clustering". University of Sarajevo, Faculty of Electrical Engineering, Department for Computer Science and Informatics, Zmaja od Bosne bb, Kampus Univerziteta, 71000 Sarajevo

[6] Khamar Basha Shaik, P. Ganesan, V. Kalist, B.S. Sathish, J. Merlin Mary Jenitha, "Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space", Procedia Computer Science, Volume 57, 2015, Pages 41-48, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.07.362, (https://www.sciencedirect.com/science/article/pii/S1877050915018918)

Fig. 30. Blue channel of pseudo-colored image 3

Fig. 31. Hue channel of pseudo-colored image 3

Fig. 32. Saturation channel of pseudo-colored image 3

Fig. 33. Intensity channel of pseudo-colored image 3

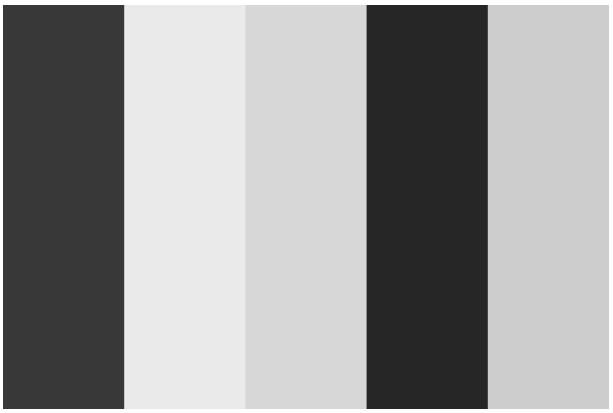

Fig. 34. Red channel of pseudo-colored image 4

Fig. 35. Green channel of pseudo-colored image 4

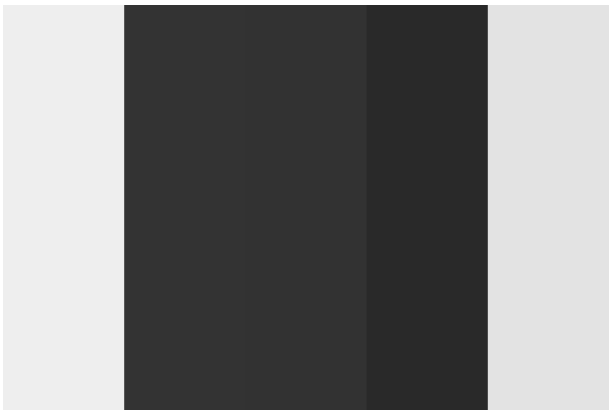

Fig. 36. Blue channel of pseudo-colored image 4

Fig. 37. Hue channel of pseudo-colored image 4

Fig. 38. Saturation channel of pseudo-colored image 4

Fig. 39. Intensity channel of pseudo-colored image 4