Nama : Fadia Ramadhana
NIM : G64170026

**LKP 6**
**PENGOLAHAN CITRA DIGITAL**

1. Buatlah fungsi konversi citra RGB to HSV dan RGB to YCrCb tanpa menggunakan fungsi OpenCV dan terapkan pada citra 'FACE DETECTION.png'. Rumus dapat dilihat dilampirkan

**##### HSV #####**

RGB $\leftrightarrow$ HSV

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-min(R,G,B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$ . On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$ .

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2(\text{to fit to } 0 \text{ to } 255)$
- 16-bit images: (currently not supported) $V < -65535V, S < -65535S, H < -H$
- 32-bit images: H, S, and V are left as is

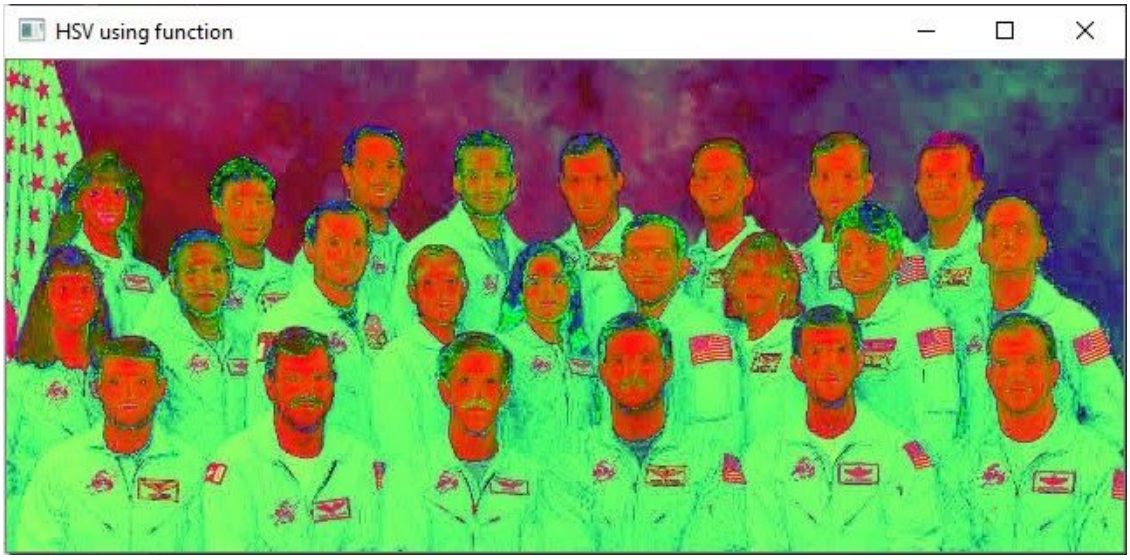
```python
import numpy as np
import cv2

img = cv2.imread("FACE DETECTION.png")

def hsv(image):
    # mengakses ukuran citra
    row,col,ch = image.shape
    # membuat canvas baru berukuran matrix 3*3
    # dan semua elemennya berisi angka "0"
    canvas_1 = np.zeros((row,col,3), np.uint8)
    for i in range(0,row):
        for j in range(0,col):
            # mengakses nilai pixel RGB
            blue, green, red = image[i,j]
            # mengubah nilai pixel menjadi integer
            b = int(blue)
            g = int(green)
            r = int(red)

            # mengakses nilai pixel maksimal dari masing-masing channel
            maksimal = max(red, green, blue)
            # mengakses nilai pixel minimal dari masing-masing channel
            minimal = min(red, green, blue)
```

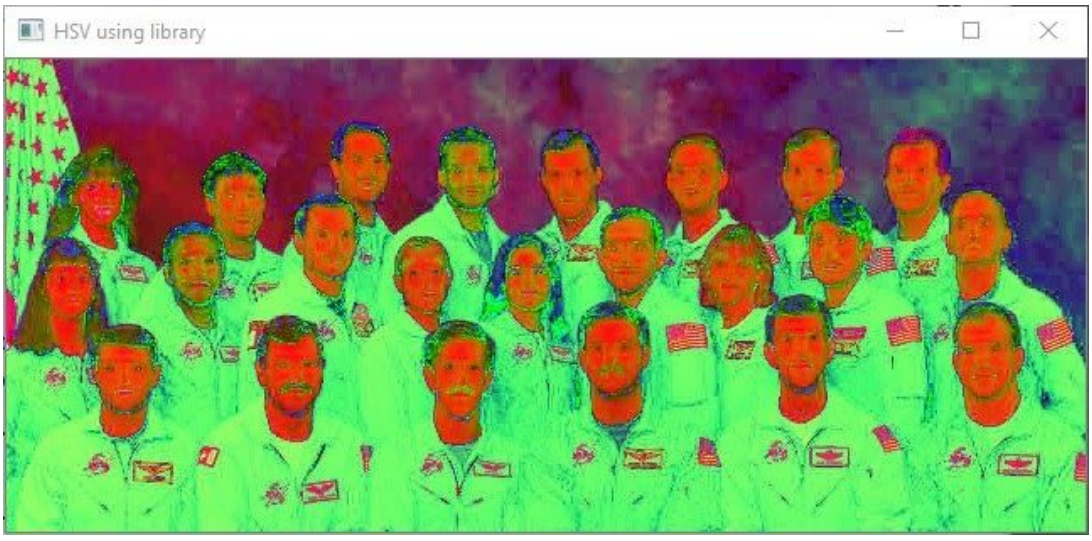
```
            # nilai V (Value)
            v = maksimal

            # nilai S (Saturation)
            if v != 0:
                s = (v-minimal)/v
            else:
                v = 0

            # nilai H (Hue)
            if v == minimal:
                h = 0
            elif v == r:
                h = (60*(g-b))/(v-minimal)
            elif v == g:
                h = 120+(60*(b-r))/(v-minimal)
            elif v == b:
                h = 240+(60*(r-g))/(v-minimal)

            # konversi nilai H, S, dan V ke tipe data tujuan
            # dalam hal ini adalah 8-bit images
            if h<0:
                h = h+360
            h = int(h/2)
            s = int(s*255)
```

```
            # memasukkan masing-masing nilai h, s, dan v ke dalam canvas_1
            canvas_1.itemset((i, j, 0), h)
            canvas_1.itemset((i, j, 1), s)
            canvas_1.itemset((i, j, 2), v)
    return canvas_1

final_hsv = hsv(img)
lib_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
(h,s,v) = cv2.split(lib_hsv)

cv2.imshow("HSV using library", lib_hsv)
cv2.imshow("HSV using function", final_hsv)
```

**Input image :**

**Output image :**



###### YCrCb ######

## RGB ↔ YCrCb JPEG (or YCC)

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$Cr \leftarrow (R - Y) \cdot 0.713 + delta$$
$$Cb \leftarrow (B - Y) \cdot 0.564 + delta$$
$$R \leftarrow Y + 1.403 \cdot (Cr - delta)$$
$$G \leftarrow Y - 0.714 \cdot (Cr - delta) - 0.344 \cdot (Cb - delta)$$
$$B \leftarrow Y + 1.773 \cdot (Cb - delta)$$

where

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating-point images} \end{cases}$$

Y, Cr, and Cb cover the whole value range.

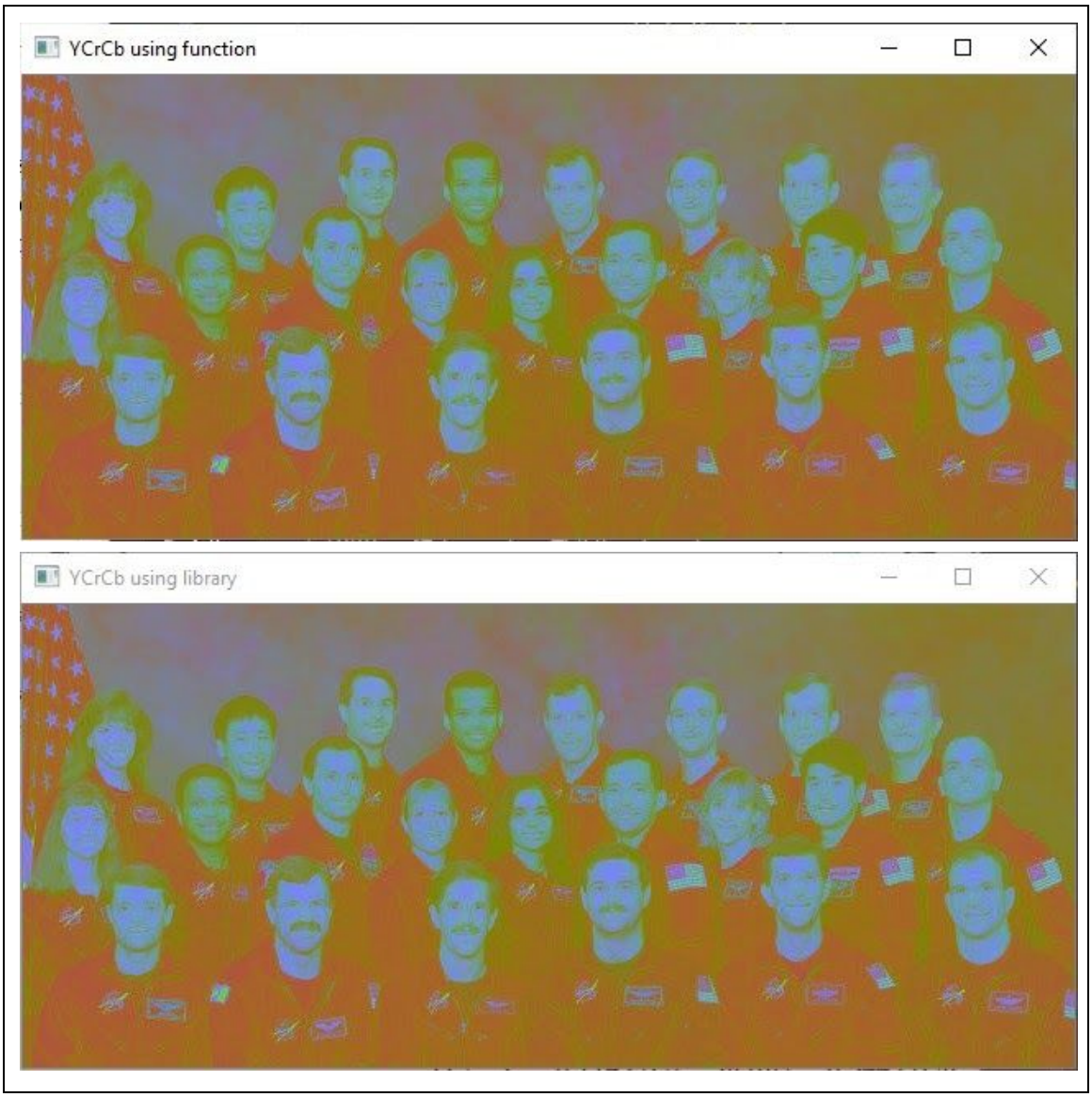
```
65  def YCrCb(image):
66      # mengakses ukuran citra
67      row,col,ch = image.shape
68      # membuat canvas baru berukuran matrix 3*3
69      # dan semua elemennya berisi angka "0"
70      canvas_2 = np.zeros((row,col,3),np.uint8)
71      # deklarasi nilai delta untuk 8-bit image
72      delta = 128
73      for i in range(0,row):
74          for j in range(0,col):
75              # mengakses nilai pixel RGB
76              blue, green, red = image[i,j]
77              # mengubah nilai pixel menjadi integer
78              b = int(blue)
79              g = int(green)
80              r = int(red)
```

```
                # nilai Y (Luminance)
                Y = 0.299*r+0.587*g+0.114*b
                # nilai Cr (Crominance red)
                Cr = (r-Y)*0.713 + delta
                # nilai Cb (Criminance blue)
                Cb = (b-Y)*0.564 + delta

                # nilai R (red)
                R = Y + 1.403*(Cr-delta)
                # nilai G (green)
                G = Y - 0.714*(Cr-delta) - 0.344*(Cb-delta)
                # nilai B (blue)
                B = Y + 1.773*(Cb-delta)
```

```
                # memasukkan masing-masing nilai Y, Cr, dan Cb ke dalam canvas_2
                canvas_2.itemset((i, j, 0), Y)
                canvas_2.itemset((i, j, 1), Cr)
                canvas_2.itemset((i, j, 2), Cb)
    return canvas_2

final_YCrCb = YCrCb(img)
lib_YCrCb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)

cv2.imshow("YCrCb using library", lib_YCrCb)
cv2.imshow("YCrCb using function", final_YCrCb)
```

**Input image :**

**Output image :**




2. Lakukan face detection pada citra 'FACE DETECTION.png'
   Pakai ilmu image subtraction, konvolusi, thresholding dan ilmu-ilmu lain dari pertemuan sebelumnya pada color space selain RGB sehingga menghasilkan segmentasi yang lebih bagus. Berilah alasan kenapa memilih color space tersebut, jika perlu sertakan juga paper rujukan.
   Output dari proses ini adalah citra baru yang isinya gambar wajah saja.

Makalah ini menyajikan studi perbandingan deteksi warna kulit manusia HSV dan color space YCbCr. Pendeteksian wajah merupakan pendeteksian kulit yang merupakan proses pemisahan antara piksel kulit dan non-kulit. Menurut Li dan Yihong (2018), sulit untuk mengembangkan metode yang seragam untuk segmentasi atau deteksi deteksi kulit manusia karena nada warna kulit manusia secara drastis

bervariasi untuk orang dari satu daerah ke daerah lain. Survei literatur menunjukkan bahwa ada berbagai color space yang diterapkan untuk pendeteksian warna kulit. color space RGB tidak baik untuk digunakan untuk pendeteksian berbasis warna dan analisis warna karena informasi dan karakteristik pencampuran warna (*chrominance*) dan intensitas (*luminance*) yang tidak seragam. Sementara itu, transformasi warna citra dari RGB ke HSV adalah proses yang memakan waktu. Dalam hal ini, koordinat Kartesius diubah menjadi sistem koordinat kutub. Deteksi berbasis HSV paling cocok untuk gambar sederhana dengan latar belakang seragam. Apalagi, jika ada banyak fluktuasi nilai informasi warna (Hue dan Saturation), piksel dengan intensitas kecil dan besar tidak dipertimbangkan. [1]

Hasil percobaan dari kedua paper rujukan menunjukkan efisiensi color space YCbCr untuk segmentasi dan deteksi warna kulit pada gambar berwarna. color space YCbCr dapat diterapkan untuk gambar warna yang kompleks dengan pencahayaan yang tidak rata (Li dan Yihong, 2018). Dalam pendeteksian wajah dan peningkatan warna, color space YCbCr dapat sepenuhnya memantulkan warna wajah, sehingga sistem dapat mendeteksi wajah lebih akurat (Shaik et al., 2015). Maka dari itu, saya memilih color space YCbCr untuk melakukan proses pendeteksian wajah pada citra FACE RECOGNITION.png. Namun, sebagai perbandingan, saya akan menggunakan color space HSV juga.

**Kode Program dan Penjelasan :**

### Step 1 : Mengeset range nilai piksel untuk masing-masing channel Y, Cr, dan Cb

Setelah membaca image, hal yang harus dilakukan adalah mengeset range nilai piksel sebagai threshold untuk masing-masing channel yaitu Y, Cr, dan Cb. Lower (min) range merupakan sebuah array yang menyimpan batas bawah dari masing-masing channel. Upper (max) range merupakan sebuah array yang menyimpan batas atas dari masing-masing channel. OpenCV menggunakan range YCrCb sebagaimana value aslinya yaitu :

- 0 > Y > 255
- 0 > Cr > 255
- 0 > Cb > 255

Bersamaan dengan aturan tersebut dan menggunakan paper dan beberapa website rujukan dengan beberapa tes empiris dan evaluasi visual, saya akan melakukan percobaan dengan menggunakan beberapa range untuk Y, Cr, dan Cb yaitu:

- (0 - 235, 133 - 173, 77 - 127)
- (0 - 255, 135 - 180, 85 - 135)
- (0 - 255, 151 - 199, 101 - 149)

Range ini nantinya akan menjadi kriteria untuk nilai-nilai piksel yang termasuk ke dalam *skin pixel*.

```
image = cv2.imread("FACE DETECTION.png")

#----- YCrCb -----#

# menerapkan threshold ke gambar HSV untuk menentukan range atas dan bawah
# batas bawah ke-1
min_YCrCb1 = np.array([0,133,77],np.uint8)
# batas atas ke-1
max_YCrCb1 = np.array([235,173,127],np.uint8)
# batas bawah ke-2
min_YCrCb2 = np.array([0,135,85],np.uint8)
# batas atas ke-2
max_YCrCb2 = np.array([255,180,135],np.uint8)
# batas bawah ke-3
min_YCrCb3 = np.array([0,151,101],np.uint8)
# batas atas ke-3
max_YCrCb3 = np.array([255,199,149],np.uint8)
```

**Step 2 : Mengkonversi color space image dari RGB ke YCrCb**

Citra yang telah dibaca sebelumnya akan diubah ruang warnanya dari RGB menjadi YCrCb. Untuk mengubah ruang warna ini dapat dilakukan menggunakan fungsi manual seperti yang ada pada nomor 1 atau dapat menggunakan fungsi pada library openCV seperti dibawah ini :

```
# konversi image dari RGB menjadi YCrCb
imageYCrCb = cv2.cvtColor(image,cv2.COLOR_BGR2YCR_CB)
```

**Step 3 : Membuat mask berdasarkan nilai range piksel**

Gambar yang telah diubah color space nya menjadi YCrCb, akan digunakan untuk membuat mask bersama dengan nilai range (min dan max) dari masing-masing channel. Mask untuk image didapat menggunakan fungsi dari openCV yaitu fungsi inRange dimana fungsi tersebut hanya mengembalikan MASK bernilai biner. Piksel putih (255) mewakili piksel yang termasuk ke dalam range antara min dan max dan begitu pula piksel hitam (0) mewakili piksel yang tidak termasuk ke dalam range.

```
# membuat mask dengan
# mengembalikan nilai biner image menggunakan range bawah dan atas (min, max)
skinRegionYCrCb1 = cv2.inRange(imageYCrCb,min_YCrCb1,max_YCrCb1)
skinRegionYCrCb2 = cv2.inRange(imageYCrCb,min_YCrCb2,max_YCrCb2)
skinRegionYCrCb3 = cv2.inRange(imageYCrCb,min_YCrCb3,max_YCrCb3)
```

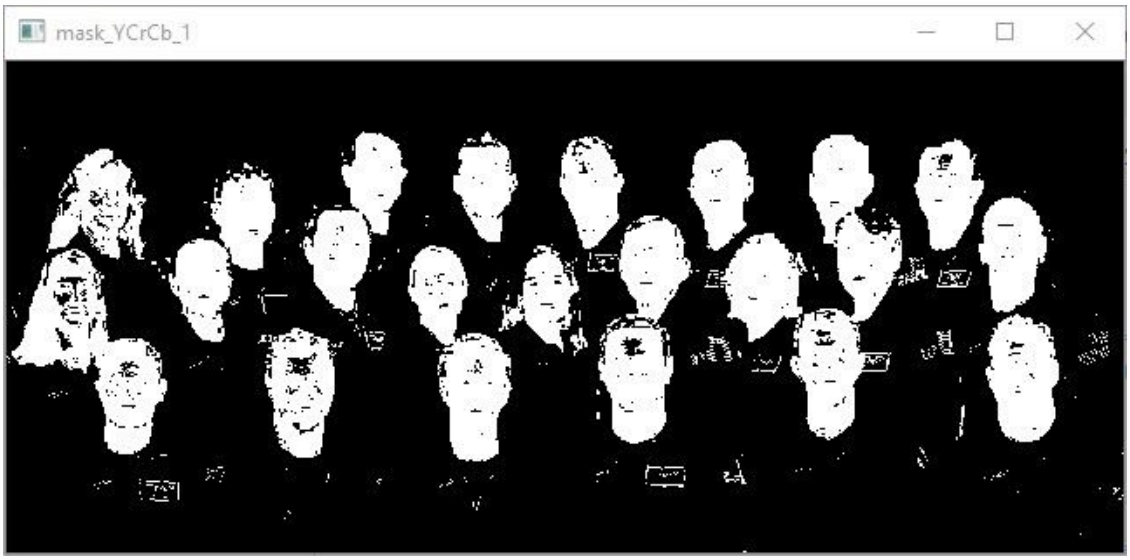
Untuk melihat bagaimana hasil dari mask gambar tersebut dapat dilakukan dengan menggunakan fungsi cv2.imshow

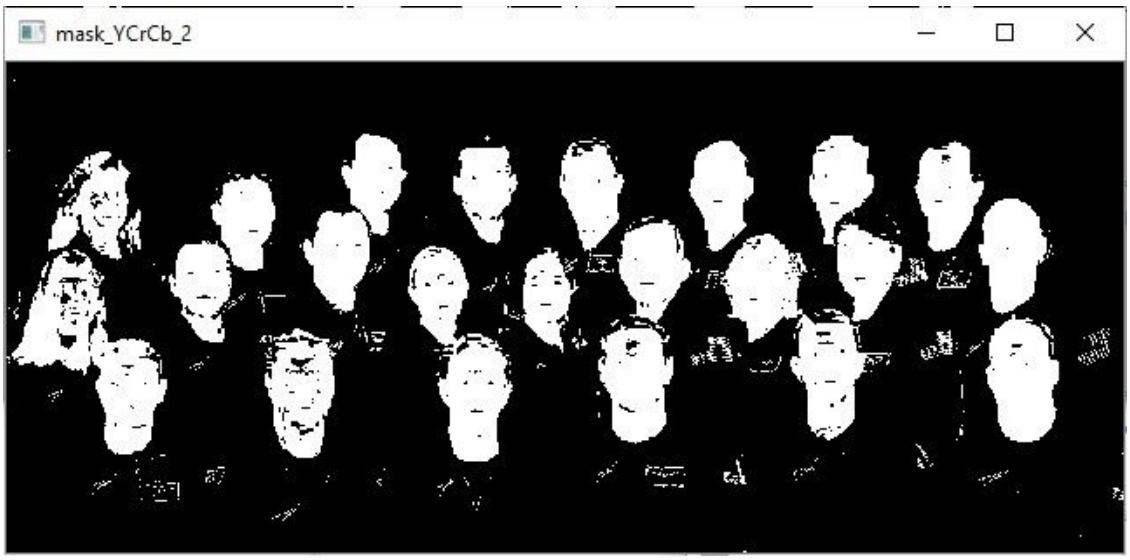
```
38      # range ke-1
39      cv2.imshow("mask_YCrCb_1", skinRegionYCrCb1)
```

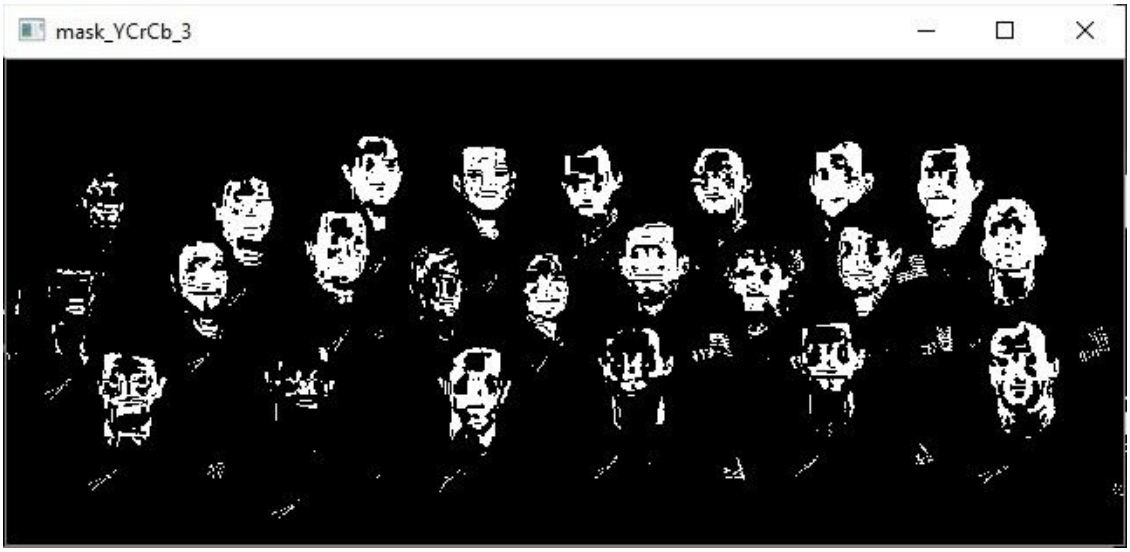
```
41      # range ke-2
42      cv2.imshow("mask_YCrCb_2", skinRegionYCrCb2)
```

```
44      # range ke-3
45      cv2.imshow("mask_YCrCb_3", skinRegionYCrCb3)
```

Dan akan menghasilkan output mask seperti berikut :






**Step 4 : Mengelompokkan wajah-wajah yang terdeteksi**

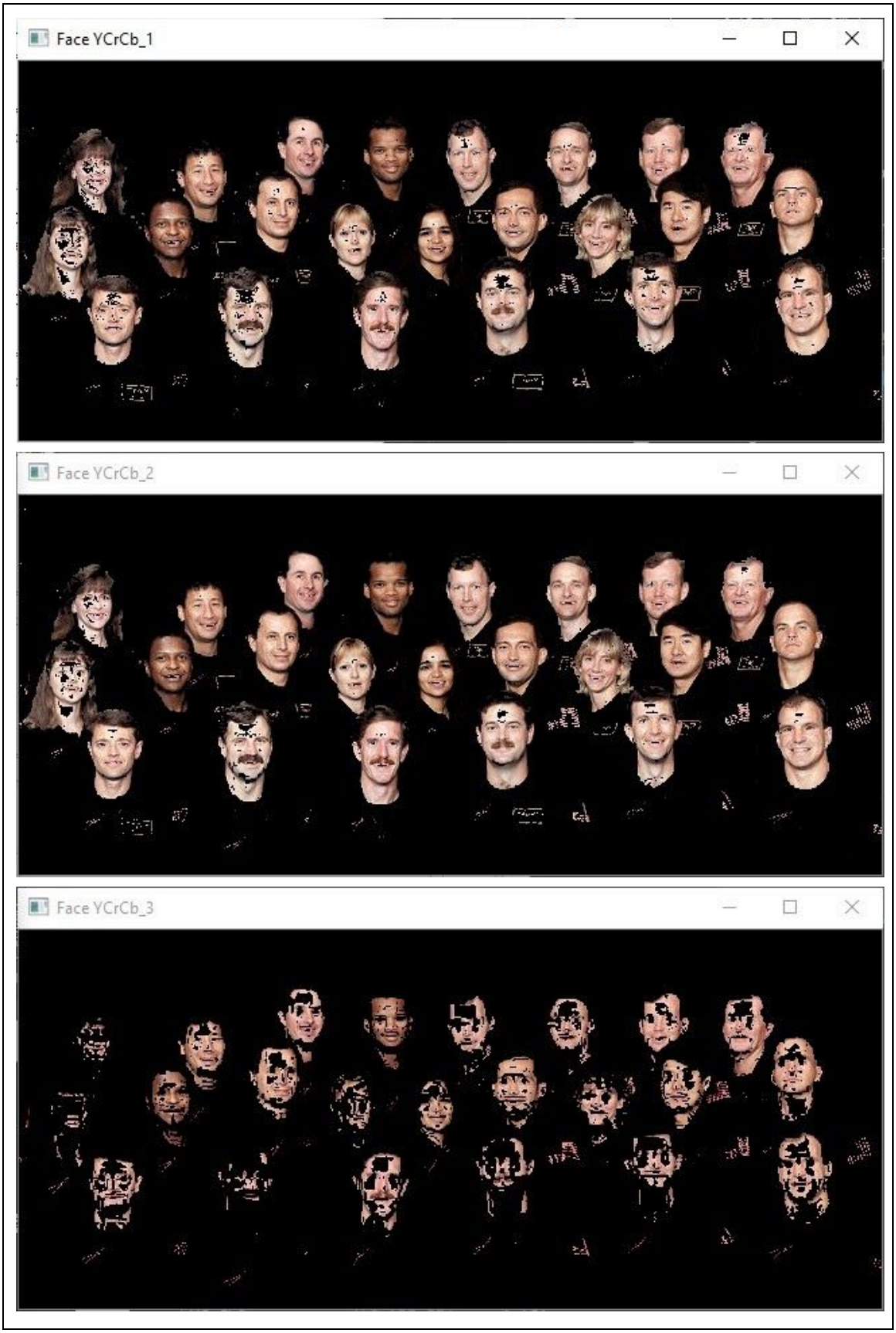
Setelah mendapatkan mask, maka langkah selanjutnya adalah melakukan segmentasi wajah pada gambar asli dengan menggunakan mask yang telah didapatkan pada step sebelumnya dengan memanfaatkan fungsi **cv2.bitwise_and()** yang menjaga setiap piksel dalam gambar yang diberikan jika nilai yang sesuai dalam mask adalah 1. Cara bekerja fungsi ini mirip seperti subtraction, dimana gambar output merupakan hasil gambar hasil irisan dari dua buah gambar.

```
32  # menggabungkan gambar asli dengan mask skinRegionYCrCb
33  # mask berada di atas gambar asli
34  faceYCrCb1 = cv2.bitwise_and(image, image, mask = skinRegionYCrCb1)
35  faceYCrCb2 = cv2.bitwise_and(image, image, mask = skinRegionYCrCb2)
36  faceYCrCb3 = cv2.bitwise_and(image, image, mask = skinRegionYCrCb3)
```

Langkah tersebut adalah langkah terakhir dari proses pengolahan gambar untuk mendeteksi wajah. Untuk melihat hasil final dapat digunakan fungsi cv2.imshow

```
38  # range ke-1
39  cv2.imshow("mask_YCrCb_1", skinRegionYCrCb1)
40  cv2.imshow("Face YCrCb_1", faceYCrCb1)
41  # range ke-2
42  cv2.imshow("mask_YCrCb_2", skinRegionYCrCb2)
43  cv2.imshow("Face YCrCb_2", faceYCrCb2)
44  # range ke-3
45  cv2.imshow("mask_YCrCb_3", skinRegionYCrCb3)
46  cv2.imshow("Face YCrCb_3", faceYCrCb3)
```

Dan output akhir adalah sebagai berikut :


Face YCrCb_1
Face YCrCb_2
Face YCrCb_3

Dari output gambar final yang dihasilkan, dapat disimpulkan bahwa perbedaan pemilihan range nilai piksel untuk mendeteksi skin pixel berpengaruh karena warna kulit manusia yang sangat beragam. Dan dari ketiga range, dapat dilihat bahwa range Y, Cr, dan Cb secara berurutan pada (0 - 255, 135 - 180, 85 - 135) adalah yang paling baik dalam mendeteksi skin pixel untuk gambar FACE RECOGNITION.png

Seperti yang telah saya katakan sebelumnya, untuk perbandingan, akan dilakukan proses pendeteksian wajah menggunakan color space HSV. OpenCV menggunakan range HSV secara berurutan tiap channel nya Hue, Saturation, dan Value adalah (0-180, 0-255, 0-255), pada aslinya range dari HSV adalah (0-360, 0-1, 0-1). Untuk lebih jelasnya adalah sebagai berikut :

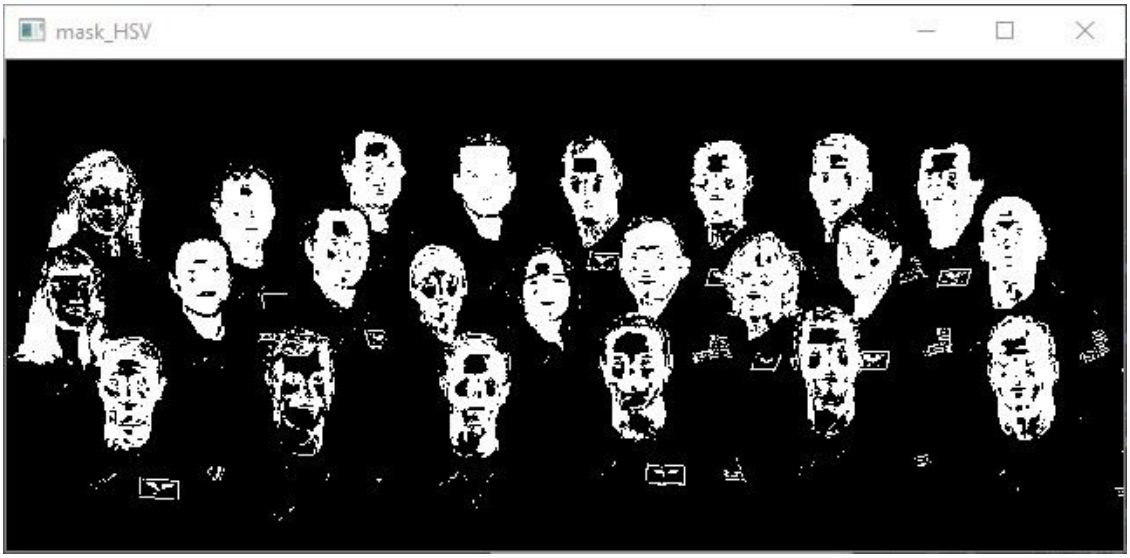
- 0 > H > 360 ⇒ OpenCV range = H/2 (0 > H > 180)
- 0 > S > 1 ⇒ OpenCV range = 255*S (0 > S > 255)
- 0 > V > 1 ⇒ OpenCV range = 255*V (0 > V > 255)

Langkah-langkah dalam pendeteksian wajah menggunakan color space HSV secara garis besar sama dengan menggunakan color space YCrCb. Hanya berbeda di bagian pendeklarasian range nilai untuk batas atas dan bawah yang sesuai untuk *skin pixel*. Berikut adalah kode program beserta output jika menggunakan color space HSV :

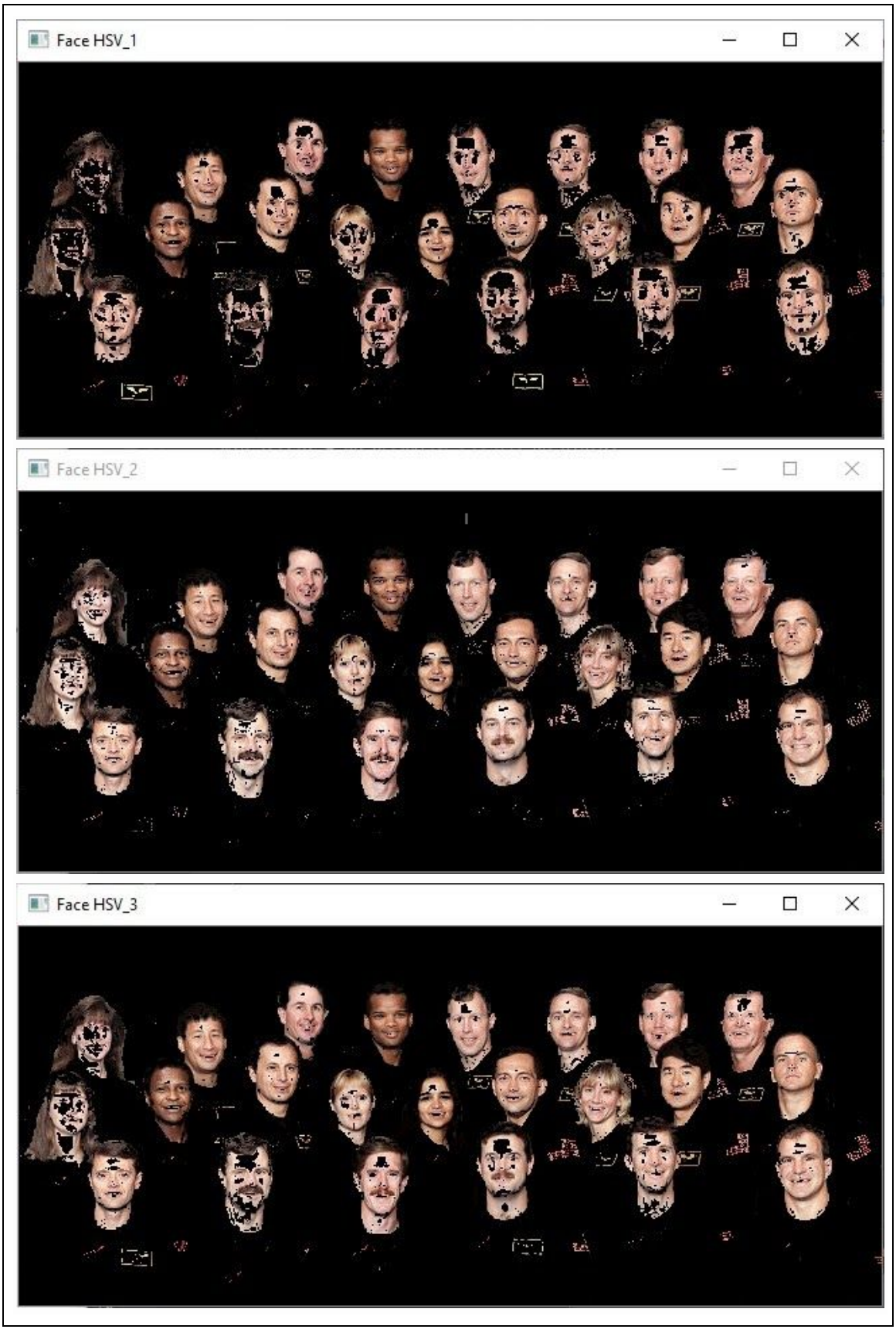
```
#----- HSV -----#
# menerapkan threshold ke gambar HSV untuk menentukan range atas dan bawah

# batas bawah ke-1
min_HSV1 = np.array([0, 58, 30],np.uint8)
# batas atas ke-1
max_HSV1 = np.array([33, 255, 255],np.uint8)
# batas bawah ke-2
min_HSV2 = np.array([0, 15, 0],np.uint8)
# batas atas ke-2
max_HSV2 = np.array([17, 170, 255],np.uint8)
# batas bawah ke-3
min_HSV3 = np.array([0, 40, 0],np.uint8)
# batas atas ke-3
max_HSV3 = np.array([25, 255, 255],np.uint8)

# konversi image dari BGR menjadi HSV
imageHSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# membuat mask dengan
# mengembalikan nilai biner image menggunakan range bawah dan atas (min, max)
skinRegionHSV1 = cv2.inRange(imageHSV, min_HSV1, max_HSV1)
skinRegionHSV2 = cv2.inRange(imageHSV, min_HSV2, max_HSV2)
skinRegionHSV3 = cv2.inRange(imageHSV, min_HSV3, max_HSV3)
```

```
74      # menggabungkan gambar asli dengan mask skinRegionHSV
75      # mask berada di atas gambar asli
76      faceHSV1 = cv2.bitwise_and(image, image, mask = skinRegionHSV1)
77      faceHSV2 = cv2.bitwise_and(image, image, mask = skinRegionHSV2)
78      faceHSV3 = cv2.bitwise_and(image, image, mask = skinRegionHSV3)
79
80      # range ke-1
81      cv2.imshow("mask_HSV_1", skinRegionHSV1)
82      cv2.imshow("Face HSV_1", faceHSV1)
83      # range ke-1
84      cv2.imshow("mask_HSV_2", skinRegionHSV2)
85      cv2.imshow("Face HSV_2", faceHSV2)
86      # range ke-1
87      cv2.imshow("mask_HSV_3", skinRegionHSV3)
88      cv2.imshow("Face HSV_3", faceHSV3)
89
90      cv2.waitKey(0)
91      cv2.destroyAllWindows()
```

**Mask HSV :**




**Output final HSV :**


Face HSV_1
Face HSV_2
Face HSV_3

Dari hasil final kedua jenis color space dapat ditarik kesimpulan jika YCrCb dapat mendeteksi skin pixel lebih baik dan efisien daripada HSV.

## Sumber

**Paper :**

[1] Li, E & Yihong, Xu. (2018). Face Detection Based on Improved Color Space of YCbCr. IOP Conference Series: Materials Science and Engineering. 439. 032075. 10.1088/1757-899X/439/3/032075.

[2] Shaik, K.B. & Packyanathan, Ganesan & Kalist, V. & B.S, Sathish & Jenitha, J.. (2015). Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space. Procedia Computer Science. 57. 41-48. 10.1016/j.procs.2015.07.362.