Projeto M2 – Segmentação

PROCESSAMENTO DE IMAGEM

TEMAS

- Limiarização de Otsu e Binárioa
- Operações Morfológicas
- Skin Color Segmentation
- K-Means
- SEEDS

Limiarização de Otsu

Define um limiar com base nos pixels da imagem ao invés de deixar para o implementador escolher previamente, como no ponto a ponto.

```
59
60 def limiadorOtsu(img_in):
61   retval, img_otsu = cv.threshold(np.uint8(img_in), 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
62   return img_otsu
63
```

Processamento Morfológico

Erosão

```python

def erosao(imagem, kernel=[[1,1,1],[1,1,1],[1,1,1],[1,1,1]]):
    imagem_erodida = np.zeros(imagem.shape, dtype=np.uint8)
    kernel = np.array(kernel)
    offI = int(np.floor(kernel.shape[1]/2.0))# Centro vertical do kernel
    imparI = kernel.shape[1]%2 # Se for impar, é somado 1 pelo motivo comentado dentro dos lassos for
    offJ = int(np.floor(kernel.shape[0]/2.0))# Centro horizontal do kernel
    imparJ = kernel.shape[0]%2
    for i in range(offI, int(imagem.shape[1]) - offI):# Executa apenas aonde o Kernel não sai da imagem
        for j in range(offJ, int(imagem.shape[0]) - offJ):
            if (kernel * imagem[int(j - offJ):int(j + offJ + imparJ), int(i - offI):int(i + offI + imparI)]).all() > 0:# Se nenhum for zero, executa
                                        #Se fizer (i - offI):(i+ offI), ele retorna i-offI e I, o ponto de parada não entra no cálculo
                imagem_erodida[j, i] = 255
            else:
                imagem_erodida[j, i] = 0

    return imagem_erodida

```

Processamento Morfológico

Dilatação

```
def dilatation(img_in, struct):
    ac  = (img_in + 1) * 255
    abc = erosion(ac, struct)
    ab  = (abc    + 1) * 255
    return ab
```

(A + B)^C = A^C - B

Processamento Morfológico

Abertuda e Fechamento

```
def opening(img_in, struct):
    C = erosion(img_in, struct)
    return dilatation(C, struct)

def closure(img_in, struct):
    C = dilatation(img_in, struct)
    return erosion(C, struct)
```

Processamento Morfológico

Fluxo de Melhora de Digital

```
def treat_fingerprint(img_in, struct=np.array([[1,1,1],[1,1,1],[1,1,1]])):
    img_in = (img_in + 1) * 255
    res = thresh.closure(thresh.opening(img_in, struct), struct)
    return (res + 1) * 255
```

Aplicação

Imagem Original

Ruidos

Ruido gaussiano

Ruido Salt and Pepper

Limiarização Com Ruido Gaussiano

Limiar 112

Limiar 120

Limiar 128

Limiar 144

Limiar Otsu

Comparação Visual

Limiar 120

Otsu

OBSERVAÇÕES

- Há diferença na captação de ruido gaussiano pelas limiarizações
- O Otsu captou mais ruidos do que as limiarizações P2P < 140

Limiarização Com Sal e Pimenta

Limiar 112

Limiar 120

Limiar 128

Limiar 144

Limiar Otsu

Comparação Visual

Limiar 120

Otsu

OBSERVAÇÕES

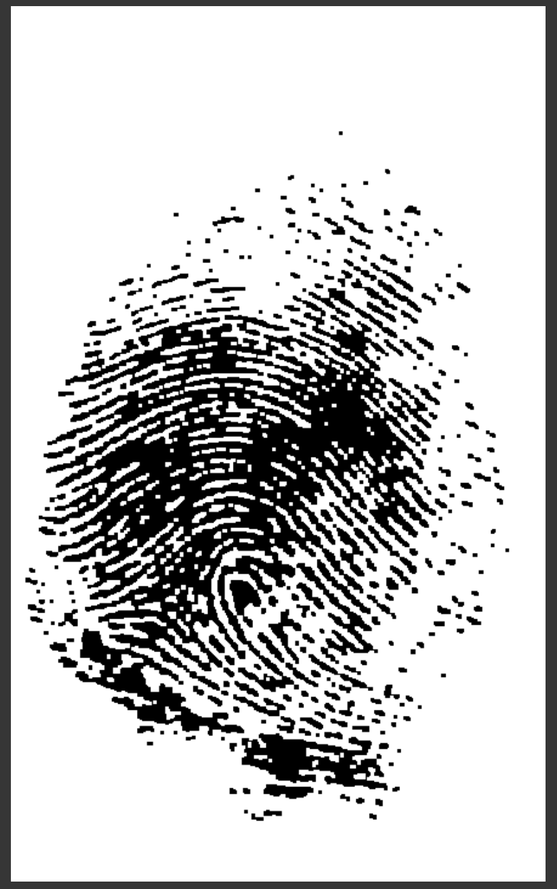
- Não há diferença entre as captações de ruidos S&P
- O limiar P2P identificoumelhor os espaçamento entre os traços do que o Otsu

Processamento Morfológico

Ruido S&P

(A · B) • B

Limiar P2P 112



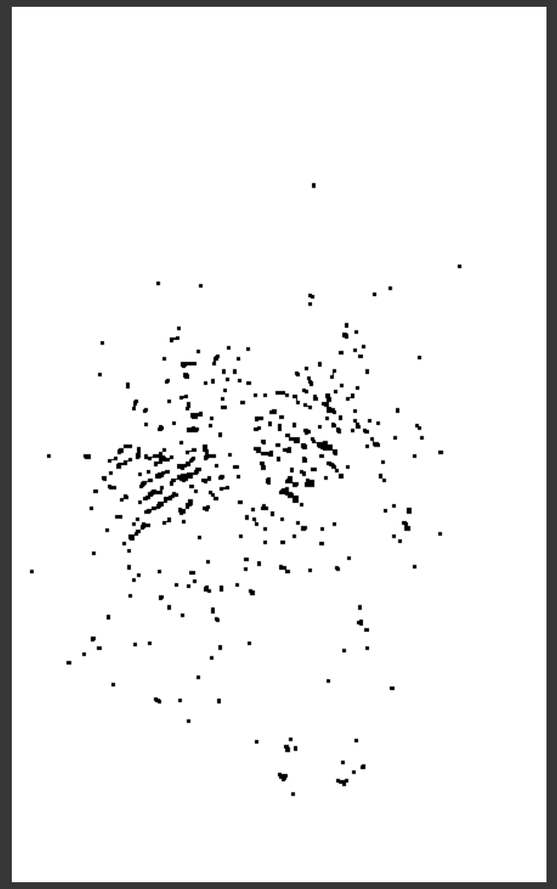
Limiar P2P 120

Otsu

Processamento Morfológico

Ruido Gaussiano

$(A \cdot B) \bullet B$



Limiar P2P 112

Limiar P2P 120

Otsu

Resultado

- Para o ruido gaussiano, o limiar Otsu deu o melhor resultado

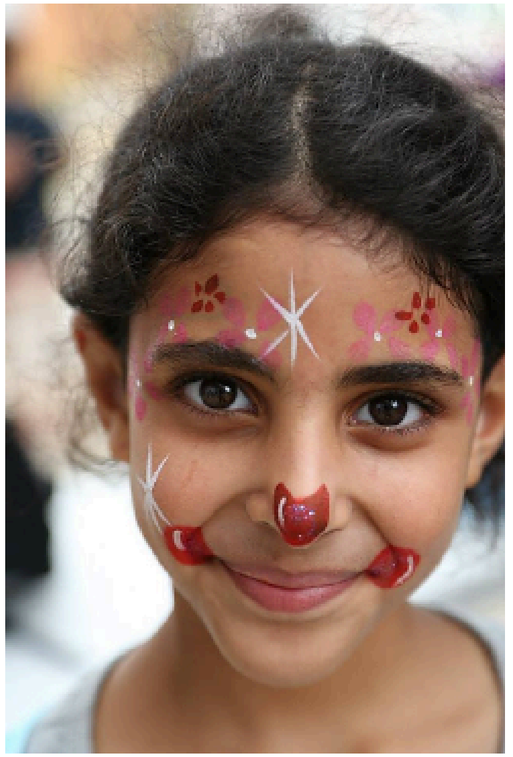
- Para o ruido S&P, o limiar P2P deu o melhor resultado

Skin Segmentation - Imagens

1

2

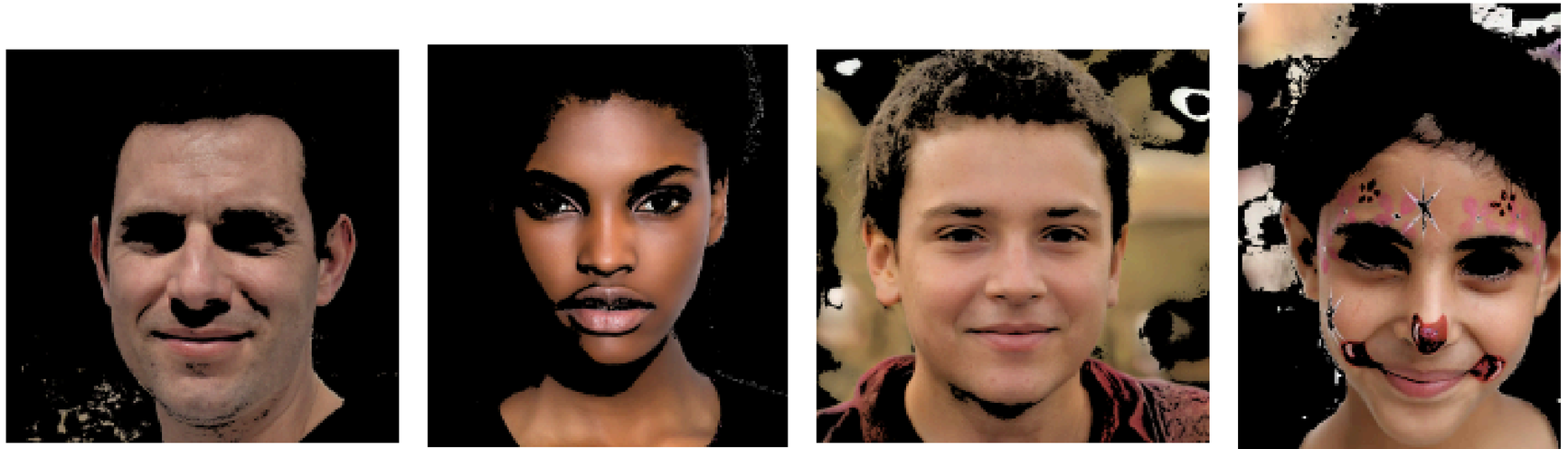
3



4

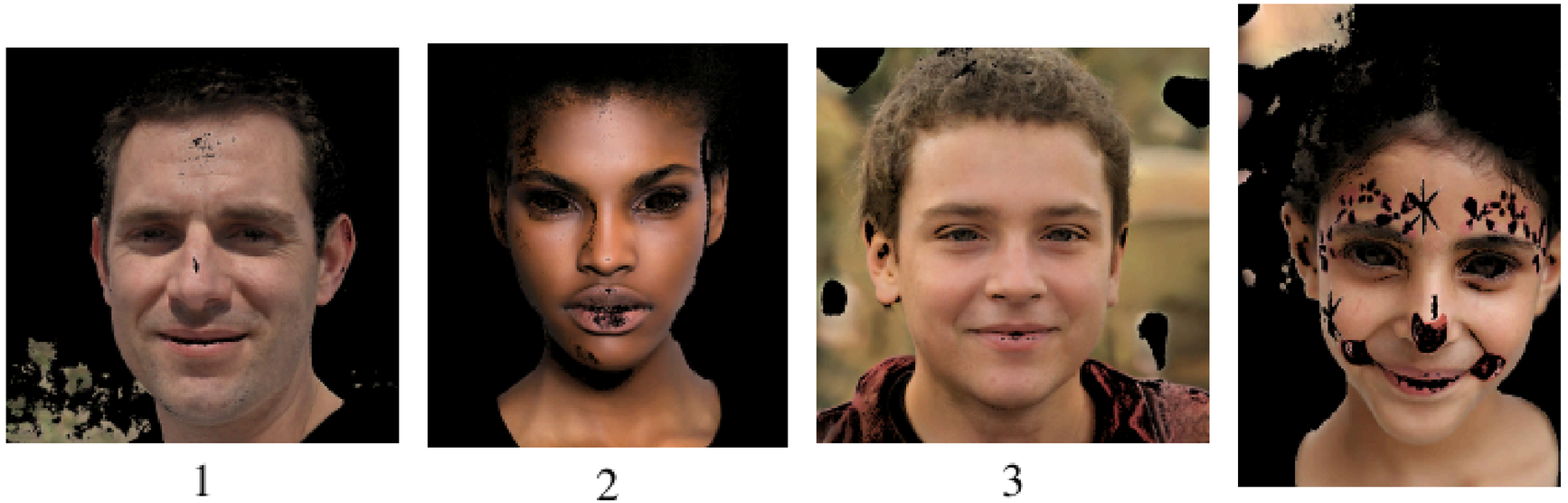
Skin Threshholding - Métodos

- Formula RGB
- Formula YCrCb
- Formula HSV
- Formula RGB+YCrCb+HSV
- Formula HSV Modificado
- Formula RGB+YCrCb+HSV Modificado



1 2 3 4

Figura 10. Faces, filtradas por RGB.

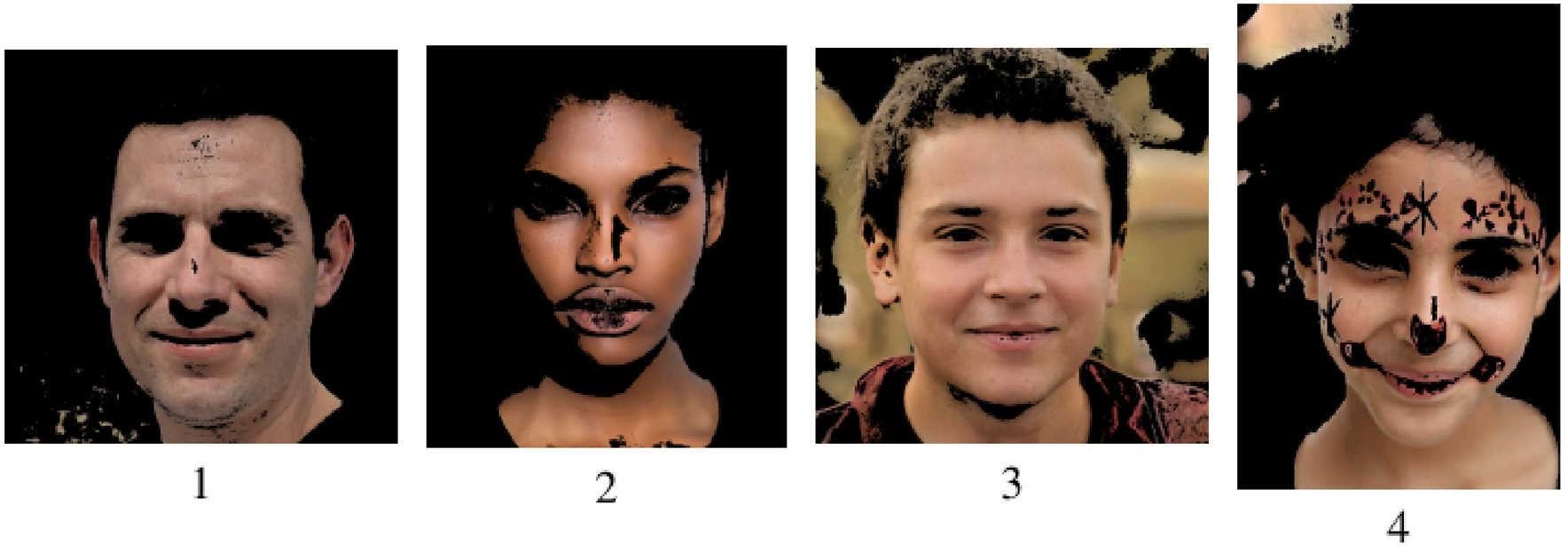


1 2 3 4

Figura 11. Faces, filtradas por HSV.

1 2 3 4

Figura 12. Faces, filtradas por YCrCb.

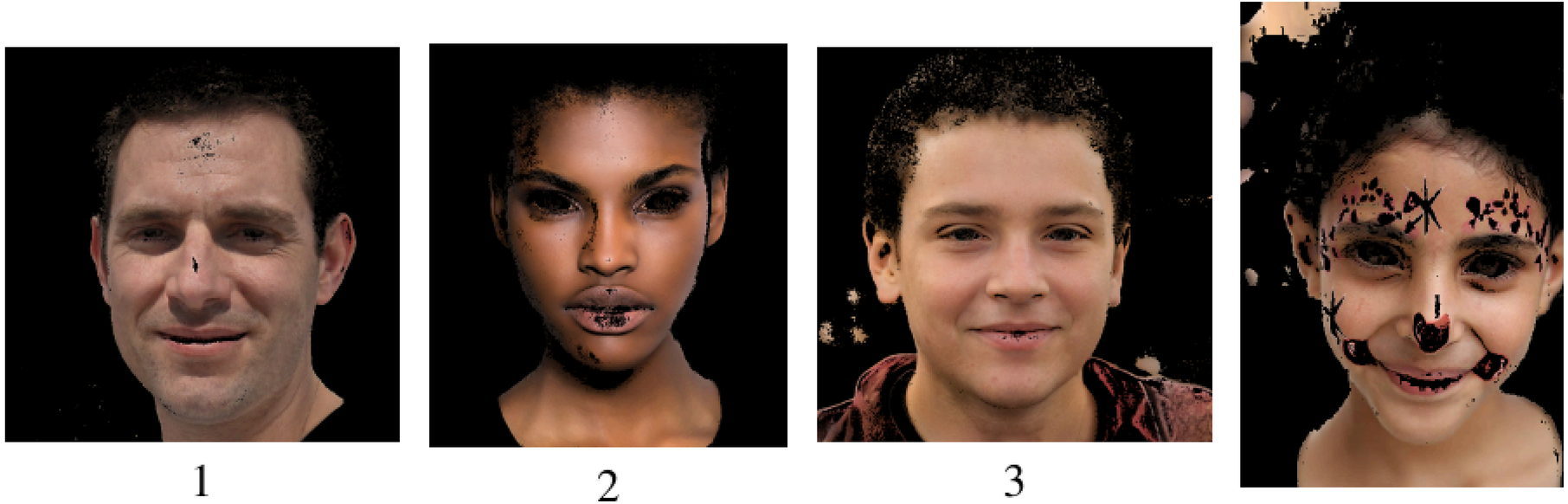


1 2 3 4

Figura 13. Faces, filtradas pela combinação de todos os métodos.

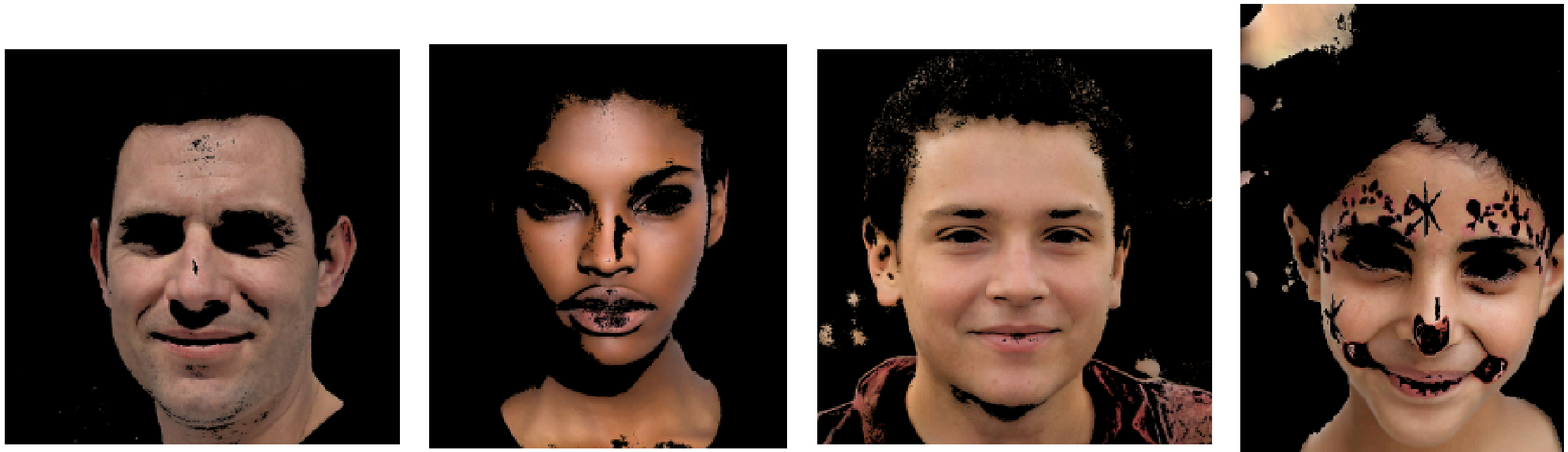
HSV - Alteração

```
def HSV_Threshold(hsv):
    return 0 <= hsv[0] and hsv[0] <= 50 and ...
# Alterado
def HSV_Threshold2(hsv):
    return 0 <= hsv[0] and hsv[0] <= 20 and ...
```



1 2 3 4

Figura 14. Faces, filtradas por HSV alterado.



1 2 3 4

Figura 15. Faces, filtradas pela combinação de todos os métodos + alteração.

K-Means - Métodos

- Várias combinações de K e Iterações
- K = {2, 3, 4, 5, 6}
- Iterações entre 10 e 100.
- A maioria das imagens parou com menos iterações.

K=2, 16 Iter

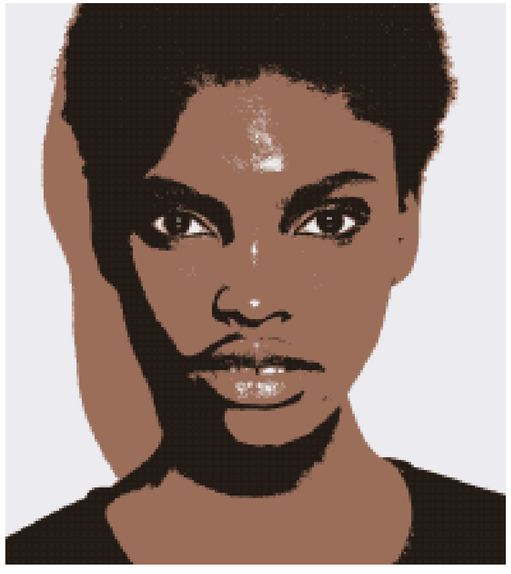
K=3, 51 Iter

K=4, 23 Iter

K=5, 24 Iter

Figura 16. Imagem um.

K=2, 22 Iter



K=3, 09 Iter

K=4, 46 Iter

K=5, 54 Iter

K=2, 21 Iter K=3, 21 Iter K=4, 15 Iter K=5, 21 Iter

Figura 18. Imagem três.

K=2, 20 Iter K=3, 23 Iter K=4, 31 Iter K=5, 49 Iter

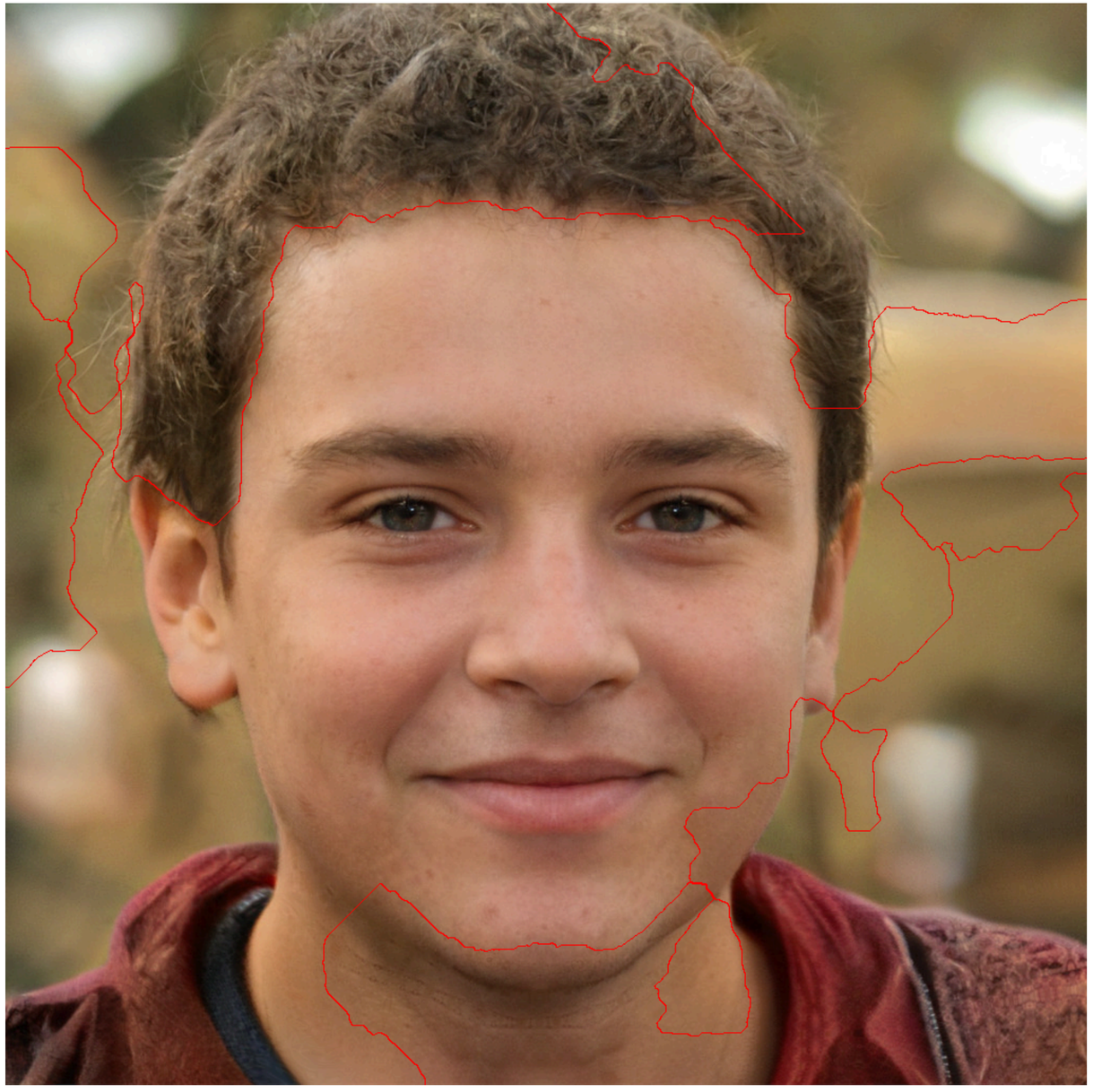
Figura 19. Imagem quatro.

SEEDS - Métodos

- Várias quantidades de superpixels
- NSP = {2, 3, 4}

```python
# https://github.com/opencv/opencv_contrib/blob/4.x/samples/python2/seeds.py
def seeds(img_or, title, nsp, prir, levels, iter):

    img_in = cv2.cvtColor(img_or, cv2.COLOR_BGR2HSV)
    height, width, channels = img_in.shape

    num_superpixels = nsp # 6
    prior = prir # 4
    num_levels = levels # 30
    num_histogram_bins = 10
    seeds = cv2.ximgproc.createSuperpixelSEEDS(width, height, channels, num_superpixels, num_levels, prior, num_histogram_bins)
    color_img    = np.zeros((height,width,3), np.uint8)
    color_img[:] = (0, 0, 255)

    seeds.iterate(img_in, iter)

    labels = seeds.getLabels()

    # labels output: use the last x bits to determine the color
    num_label_bits = 2
    labels &= (1<<num_label_bits)-1
    labels *= 1<<(16-num_label_bits)

    mask = seeds.getLabelContourMask(False)

    mask_inv = cv2.bitwise_not(mask)
    result_bg = cv2.bitwise_and(img_or, img_or, mask=mask_inv)
    result_fg = cv2.bitwise_and(color_img, color_img, mask=mask)
    result = cv2.add(result_bg, result_fg)

    cv2.imwrite(title, result)
```


StyleGAN2 (Karras et al.)



Conclusões

Para a leitura de digitais sob ruído, principalmente aquelas com baixa contraste, talvez lidas de uma folha de papel, aconselhamos o limiar básico com um threshhold de 120. Quanto ao ruído o Salt & Pepper danificou menos a imagem.
Para a segmentação de faces e pele humana, aconselhamos fortemente o método de Skin Thresholding RGB+YCbCr+HSV customizado.

Fim