PD4 - Segmentação de cor de pele

Frederico Guth (18/0081641)
Tópicos em Sistemas de Computação, ,
Turma TC - Visão Computacional (PPGI)
Universidade de Brasília
Brasília, Brasil
fredguth@fredguth.com

***Resumo*—Detecção de pele lida com o reconhecimento dos pixels que representam pele em uma dada imagem. Cor é frequentemente usada por ser invariante a orientacao, tamanho e ser fácil de processar. Neste projeto, detectamos cor de pele usando tres tipos de classificadores: por limiarizacao, bayesiano simples e k-nn. Os resultados comparam qualidade da segmentação em relação ao tempo de execução dos algoritmos.**

***Index Terms*—deteção de pele, espaço de cores, K-NN, naive bayesian, threshold-based**

I. Introdução

Deteção de pele é o processo de classificação binária de pixels em imagens como sendo *pele* ou *não-pele*. Diversas aplicações de Visão Computacional lidam com interações com humanos e a deteção de pele é tipicamente usada no pré-processamento de imagens para encontrar regiões de interesse. [1]–[3]

Cor é o principal fator na deteção de peles; é invariante à orientação e tamanho e rápida de processar. Apesar da variação de tonalidades de peles entre pessoas de diferentes origens e da variação causada pelas condições de luz na obtenção de imagens [2], já foi observado que a maior variação é de luminosidade, não de tonalidade [1]. Em imagens com normalização de brilho, a pele humana se agrupa em pequenas regiões de um espaço de cores [4]. Assim, o problema de detecção de pele pode ser abordado como um problema de classificação de cores pixel a pixel.

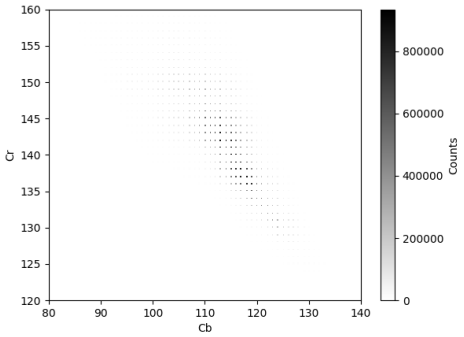


Figura 1: Cluster representando cor de pele na base SFA completa no espaço de cores CbCr

Há diversos algoritmos de classificação que podem ser usados, dos mais ingênuos e simplistas como por Limiarização até os mais sofisticados usado Redes Neurais, com diferentes custos-benefícios em termos de qualidade da segmentação versus tempo de execução.

A. Objetivos

Neste projeto apresentaremos o resultado comparativo de custo-benefício de 3 métodos de deteção de pele: Por Limiarização, Classificador Bayesiano Simples e K vizinhos mais próximos. A base de dados utilizada é a SFA [5] criada especificamente para o problema de segementação de pele.

Mais especificamente deseja-se fazer:

1) segmentação usando subconjunto da SFA (10 imagens);
2) segmentação usando toda a base de imagens SFA (1118 imagens);
3) análise dos resultados.

II. Revisão Teórica

A. Espaço de Cores

Espaço de Cores é um modelo matemático que representa a informação de cor na forma de componentes, geralmente três ou quatro vetores. Por exemplo, o modelo RGB representa diferentes cores como uma composição de vermelho (R), verde (G) e azul (B). Diferentes modelos servem diferentes aplicações como computação gráfica, processamento de imagens, Transmissão de sinal de TV, entre outras. Alguns desses modelos são baseados em tonalidade (HSI, HSV e HSL) outros em Brilho (YCbCr, YIQ e YUV) [2].

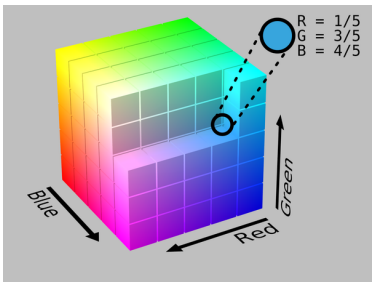
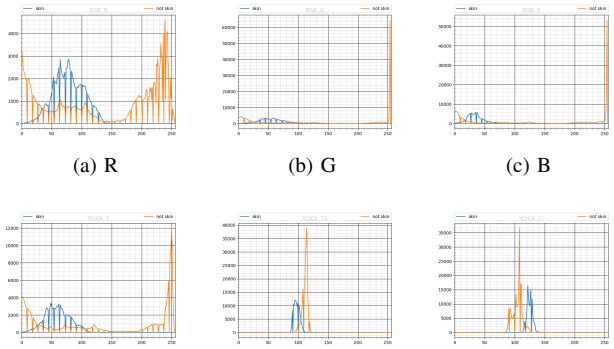


Figura 2: Modelo simplificado do espaço RGB [6]

Apesar de nenhum modelo de espaço de cores ter sido criado especialmente para a aplicação de deteção de pele, os modelos baseados em brilho, como o YCbCr, facilitam a normalização proposta por [4].



(a) R (b) G (c) B

(d) Y (e) Cb (f) Cr

Figura 3: Histograma por canal dos pixels da base de treinamento SFA reduzida

B. Classificador baseado em Limiarização

A classificação por limiarização (*Threshold-based*) é um algoritmo muito simples e comumente encontrado em tutoriais de deteção de pele [7]. Em um determinado espaço de cor, definem-se limites mínimos e máximos da cor de pele, por exemplo:

$$\begin{aligned}(R > 110) \cup (R < 210) \\ \cup (G > 40) \cup (G < 100) \\ \cup (B > 20) \cup (B < 60)\end{aligned} \tag{1}$$

O pixel que atende essa condição é classificado como *pele*, os que não atendem como *não-pele*. Apesar de simples, esse algoritmo tem a vantagem de ser potencialmente muito rápido, principalmente se não houver necessidade de transformação do espaço de cor da imagem.

Para escolher os limiares, pode-se usar sugestões publicadas em artigos ou gerar histogramas**??** para sua base de amostras e escolhê-los.

C. Classificador Bayesiano Simples

Em um Classificador Bayesiano Simples (ou Ingênuo), consideramos um conjunto de amostras classificadas em m classes distintas $C_1, C_2, ..., C_m$, e X a tupla que se deseja classificar (não fazendo parte das amostras), o classificador encontra a classe C_i para o qual $P(C_i \mid X)$ é a maior entre as classes existentes.

No nosso caso, temos só duas classes: *pele* ou *não-pele* (usaremos também os termos *skin* ou *background(bg)*); e, como usamos o espaço de cores YCbCr e normalizamos as imagens em relação ao brilho, simplesmente removendo o componente Y, X é a tupla $CbCr$, que aqui chamaremos de uv.

Assim:

$$P(uv) = \frac{c[uv]}{T} \tag{2}$$

onde $c[uv]$ é a contagem de quantas vezes a tupla uv aparece nas amostras de treinamento e T é o total de pixels nas amostras.

Para deteção de cor, usamos histogramas para determinar uma determinada probabilidade $P(uv \mid skin)$ de um determinado valor $CbCr$ ser da classe *pele*. Tal probabilidade é obtida através do Teorema de Bayes [1], onde a classe *skin* é definida por:

$$P(skin \mid uv) = \frac{P(uv \mid skin)P(skin)}{P(uv \mid skin)P(skin) + P(uv \mid bg)P(bg)}$$

$$P(skin) = \frac{T_s}{T}$$

$$P(uv \mid skin) = \frac{c_{skin}[uv]}{T_s} \tag{3}$$

onde $c_{skin}[uv]$ é a contagem de quantas vezes uv é classificado como *skin* na base de treinamento e T_s é o total de pixels classificados como *skin*. Os valores de $P(bg)$ e $P(uv \mid bg)$ podem ser obtidos da mesma forma.

Uma nova tupla /(uv) é classificada como *pele* se:

$$P(skin \mid uv) > P(bg \mid uv)$$

que pode ser simplificado [1]:

$$c_{skin}[uv] > c_{bg}[uv]$$

,

D. K vizinhos mais próximos (K-NN)

A classificação por K vizinhos mais próximos (K nearest neighbors - Knn) é o nome dado a uma família de técnicas usadas em problemas de classificação ou regressão [8]. A priori, este classificador não exige uma fase de treinamento. Considerando um conjunto de amostras classificadas em m classes distintas $C_1, C_2, ..., C_m$, e X a tupla que se deseja classificar (não fazendo parte das amostras), em tempo de execução o classificador encontra os K elementos conhecidos mais próximos de X e usa a classificação daqueles para definir a classe deste.

Para definir proximidade, diferentes funções de distância podem ser utilizadas: euclidiana, manhattan, minkowski, ou até uma função personalizada para o problema. A classificação de K pode ser obtida através da média das classificações dos K vizinhos mais próximos, uma regra de votação (contagem) ou outra função personalizada ao problema.

Por ser um algoritmo genérico com alto potencial de aplicação, há diversas implementações disponíveis do mesmo. Neste projeto, usamos a implementação sklearn.neighbors da biblioteca SciPy [9]

III. Metodologia

A. Materiais

Foram utilizados:

- computador MacBook Pro (Retina, 13-inch, Early 2015), Processador Intel Core i5 2,7 GHz, 8GB de RAM
- Python 3.6.3 :: Anaconda custom (64-bit)
- OpenCV 3.4.0

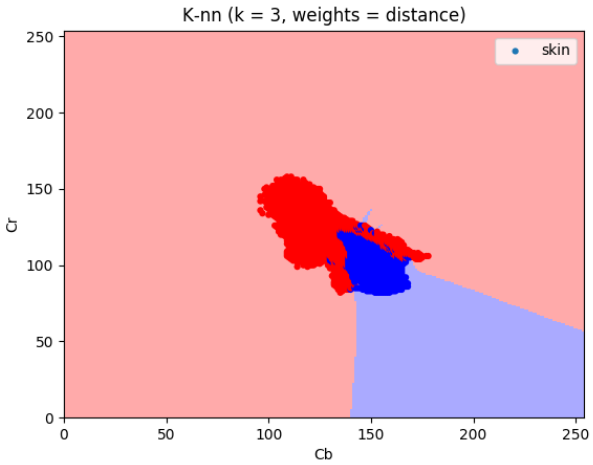


Figura 4: Classificação 3-NN dos valores de Cb e Cr na base SFA reduzida.

- dez programas em python especialmente desenvolvidos para o projeto. Todos estão disponíveis no repositório: git@github.com:fredguth/unb-cv-3183.git. Todos os tempos obtidos foram executados apenas usando apanas a CPU do computador.

B. segmentação usando subconjunto da SFA (10 imagens)

1) Executa-se o programa *r1-a.py*, que gera o diretório *results* uma série de histogramas de densidade de cores no *dataset* em questão.
2) Executa-se o programa *r1-b.py*, que calcula o baseline para os parâmetros de acuidade e índice jaccard, através de um algoritmo nulo, que apenas classifica todos os pixels como *não-pele* e anotam-se os resultados.
3) A partir dos histogramas, definem-se o espaço de cor apropriado e os limites que serão utilizados para cada canal de cor, inserindo-se no programa *r1-c.py*.
4) Executa-se o programa *r1-c.py*, que implementa um classificador por limiarização no espaço BGR muito simples, aplicando apenas um limite no canal azul e não faz nenhuma transformação no espaço de cores, e um classificador no espaço YCbCr que aplica limites aos canais Cb e Cr. Anotam-se os resultados.
5) Executa-se o programa *r1-d.py*, que implementa um classificador bayesisano simples e anotam-se os resultados.
6) Executa-se o programa *r1-e.py*, que implementa um classificador k-nn e anotam-se os resultados.

C. segmentação usando toda a base de imagens SFA (1118 imagens)

1) Executa-se o programa *r2-a.py*, que gera o diretório *results* uma série de histogramas de densidade de cores no *dataset* em questão.

Tabela I: Base SFA 10 imagens (8 treinamento, 2 teste)

Algoritmo	Qualidade		Tempo Execução	
	Acuidade (%)	Índice Jaccard (%)	Treinamento (s)	Teste (s)
Nulo	74.47	0.0	0	0
Lim (B)	44.41	11.11	0	0.06
Lim(CbCr)	81.15	34.37	0	0.04
Bayes	86.11	45.20	3.62	0.84
K-nn	84.97	45.64	3.46	8.59

2) Executa-se o programa *r2-b.py*, que calcula o baseline para os parâmetros de acuidade e índice jaccard, através do algoritmo nulo e anotam-se os resultados.
3) A partir dos histogramas, definem-se o espaço de cor apropriado e os limites que serão utilizados para cada canal de cor, inserindo-se no programa *r2-c.py*.
4) Executa-se o programa *r2-c.py*, que implementa um classificador por limiarização e anotam-se os resultados.
5) Executa-se o programa *r2-d.py*, que implementa um classificador bayesisano simples e anotam-se os resultados.
6) Executa-se o programa *r2-e.py*, que implementa um classificador k-nn e anotam-se os resultados.

IV. Resultados

Nesta seção apresentamos os resultados obtidos. Todos os dados podem ser acessados no repositório do projeto (III-A).

A. Segmentação usando subconjunto da SFA (10 imagens)

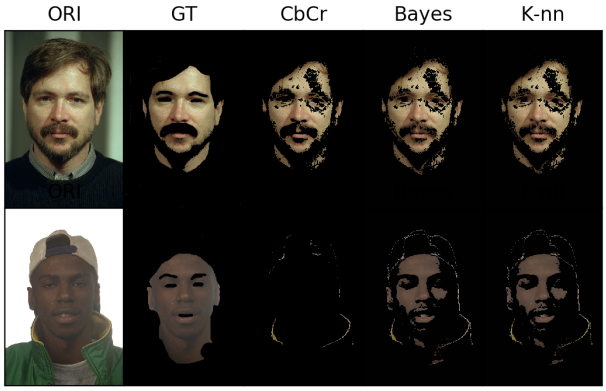


Figura 5: Resultado qualitativo dos diferentes algoritmos.

O resultado qualitativo obtido com os diferentes algoritmos pode ser apreciado em **??**.

Para uma análise quantitativa da qualidade e do tempo de execução, temos I

B. segmentação usando toda a base de imagens SFA (1118 imagens)

O resultado qualitativo obtido com os diferentes algoritmos pode ser apreciado em **??**.

Para uma análise quantitativa da qualidade e do tempo de execução, temos II

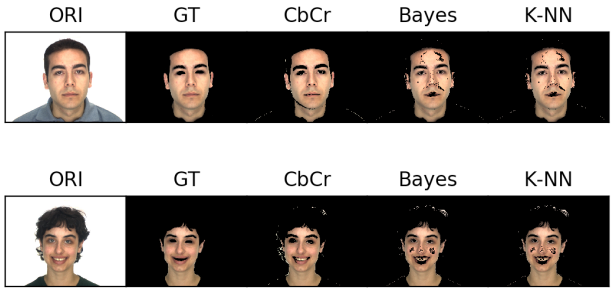


Figura 6: Resultado qualitativo dos diferentes algoritmos.

Tabela II: Base SFA 1118 imagens
(782 treinamento, 167 validação, 167 teste)

Algoritmo	Qualidade		Tempo Execução	
	Acuidade (%)	Jaccard (%)	Trein. (s)	Teste (s)
Nulo	79.59	0.0	0	0
Lim (B)	69.64	1.38	0	3.33
(CbCr)	95.03	77.78	0	3.01
Bayes	94.62	76.77	364	77
K-nn	94.49	77.00	406	1660

C. *Análise dos resultados*

Em face aos resultados obtidos, cabem algumas análises:

1) O uso de um algoritmo nulo se mostrou muito importante, uma vez que apontou que a métrica escolhida como acuidade (pixels segmentados corretamente em relação ao total de pixels da imagem) é uma métrica muito ruim, uma vez que dado que as cores que representam pele são um pequeno cluster dentro do espaço de cores, $P(bg)$ é muito maior, logo o algoritmo nulo consegue bons resultados de acuidade.
   O índice Jaccard é melhor nesse sentido, mas não discrimina falsos positivos de falsos negativos.
2) O algoritmo por limiarização CbCr apresentou um resultado quantitativo melhor do que o esperado. Já esperava-se que fosse rápido, e foi, classificando 167 imagens de teste em apenas 3 segundos. Há de se considerar o fator ”sorte”na escolha dos limites para essa base. A análise qualitativa, entretanto, mostra que para amostras pequenas é razoavelmente pior do que o classificador bayesiano e o k-nn.
3) A limirização apenas pelo canal B em RGB tinha pouca chance de ir bem e confirmou o que esperávamos. Mais interessante foi notar que a biblioteca OpenCV [10] não apresenta nenhuma diferença de tempo para carregar imagens no espaço RGB (que é o mesmo da imagem de origem) e carregar no espaço YCbCr.
4) Os algoritmo bayesiano se mostrou uma alternativa mais robusta que o de limiarização, apresentando um resultado muito melhor na base pequena e bastante rápido também, levando 77 segundos para testar 167 imagens (menos de 0.5 segundo por imagem em uma CPU antiga).

V. Discussão e Conclusões

Neste trabalho, implementamos três algoritmos para a segmentação de cor de pele em imagens e apresentamos os diferentes custos-benefícios.Demonstramos que, em conformidade com a teoria existente, a segmentação de pele por cor usando algoritmos simples de classificação atinge bons resultados que são mais do que suficiente para diversas aplicações de pré-processamento.

Referências

[1] T. de Campos, “3d hand and object tracking for intention recognition,” University of Oxford, Tech. Rep., 2003.
[2] S. Kolkur, D. Kalbande, P. Shimpi, C. Bapat, and J. Jatakia, “Human skin detection using rgb, HSV and ycbcr color models,” *CoRR*, vol. abs/1708.02694, 2017. [Online]. Available: http://arxiv.org/abs/1708.02694
[3] A. Conci, E. Nunes, J. Pantrigo, and A. Sanchez, “Comparing color and texture-based algorithms for human skin detection.” *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, pp. 166–173, 01 2008.
[4] J. Yang, W. Lu, and A. Waibel, “Skin-color modeling and adaptation,” in *Computer Vision — ACCV’98*, R. Chin and T.-C. Pong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 687–694.
[5] J. P. B. Casati, D. R. Moraes, and E. L. L. Rodrigues, “SFA: A human skin image database based on FERET and AR facial images,” in *IX Workshop de Visao Computational*, Rio de Janeiro, Brazil, 2013, dataset available at http://www.sel.eesc.usp.br/sfa/.
[6] W. Commons, “File:rgb cube show lowgamma cutout b.png,” Março 2010. [Online]. Available: https://commons.wikimedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png
[7] A. Rosebrok, “Skin detection: A step-by-step example using python and opencv,” Agosto 2014. [Online]. Available: https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/
[8] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/
[9] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–, [Online; accessed 16th May, 2018]. [Online]. Available: http://www.scipy.org/
[10] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.