Substituição facial utilizando algoritmos de *landmark* e djikstra

Projeto final de IPI

Rafael Lourenço de Lima Chehab, Igor Coutinho Soriano Lousada, Gabriel Nazareno Halabi
Departamento de Ciência da Computação
Universidade de Brasília, UnB
Brasília, Brasil
Matrículas: 15/0045123, 15/0051531

***Resumo*— O objetivo desse artigo é construir um software que sobreponha as regiões do olho, baca e nariz de uma face em outra, para isso deve-se considerar rotação, deslocamento, tom de pele**

Keywords*— *processamento de imagens, facial recognition, image blending

I. Introdução

O objetivo do trabalho foi a manipulação de imagens digitais através do software matlab, utilizando conceitos aprendidos durante a disciplina. O tema do trabalho foi realizar um troca facial entre 2 pessoas, realizando a sobreposição de uma face em relação a outra, mas não por completo, apenas as regiões dos olhos, nariz e boca. A face que irá ser usada para sobrepor deve ser adaptada em relação a face que irá ser sobreposta, tanto em relação a rotação quanto em relação a pigmentação de pele.

Fig. 1. Resultado do processamento usando uma imagem espelhada da monalisa e a lena

II. Introdução Teórica

A. Detecção de faces

O primeiro passo do software é a detecção de rostos, para isso utiliza-se o método Viola-Jones através de um objeto já definido pelo aplicativo Matlab, que foi utilizado na criação desse projeto.

B. Resultado final do processamento, utilizando uma imagem previamente espelhada da monalisa e a lenaLandmarks

Mais importante que a detecção da face é a detecção de *landmarks*.

Segundo o dicionário Cambridge, *landmark* é um prédio ou lugar que é facilmente reconhecido, especialmente que sirva como ponto de referência. Similarmente, no reconhecimento de faces, *landmarks* são pontos importantes da imagem que permitem ter uma noção espacial da face.

No nosso projeto se usam estes *landmarks*: ponta do nariz, cantos dos olhos esquerdo e direito, ponta da boca e centro do rosto. A partir deles, também se calculam os centros dos olhos.

Para a detecção de *landmarks,* se utilizou o software flandmarks (http://cmp.felk.cvut.cz/~uricamic/flandmark/), que é escrito em c++ e utiliza uma interface para matlab.

C. Inclinação e rotação

Para se medir a inclinação de uma imagem, se utilizam os centros dos olhos, que são calculados como a média dos cantos. A partir de dois centros, esquerdo e direito, tem-se que:

$$\alpha = \text{arctg}(\Delta y / \Delta x) = \text{arctg}((y_{dir} - y_{esq}) / (x_{dir} - x_{esq}))$$

Com a inclinação das duas imagens, se inclina aquela do qual será retirado o rosto, para que ambas tenham o mesmo ângulo.

D. *Seam Search*

Uma vez detectados o rosto e os *landmarks*, deve-se utilizá-los para segmentar a imagem. O software criado permite que o usuário escolha entre uma segmentação retangular ou elíptica. Para evitar que se segmente uma região que corte alguma característica importante do rosto, se define uma região de interesse (ROI – *region of interest*). Essa região é a diferença entre duas possíveis segmentações, ou seja, a diferença entre dois retângulos ou duas elipses.

Deseja-se segmentar essa região de modo que não haja cores muito diferentes, então utiliza-se um critério tal que valores muito parecidos indiquem cores parecidas e valores diferentes indiquem cores pouco similares. Assim achar a região significa achar um caminho fechado cuja soma dos gradientes absolutos, diferença em módulo, de dois pixeis consecutivos seja a menor possível.

E. *Djikstra*

Assim, pode-se formular o problema como o menor caminho em um grafo implícito em um plano 2D. Para isso, faz-se um corte vertical a partir do centro, encontrado pelo *flandmarks*, e encontra-se o menor caminho de um pixel à esquerda até um à direita do corte. O menor caminho incluirá um ponto pertencente ao corte

Para achar o menor caminho, usa-se o algoritmo Djikstra. Para realizá-lo se criou dois programas em c++ com interface em Matlab, uma vez que a sua implementação em Matlab geraria um alto tempo de execução, além das dificuldades da criação em si.

Adjunto aos código c++, se criou um Makefile que os compila. Os dois códigos existem devido à duas possíveis maneiras de definir-se o critério de similaridade entre pixeis. A primeira, djikstra.cpp, utiliza o valor em níveis de cinza, a segunda, djikstra2.cpp, utiliza o valor do *hue*, no espaço de cores HSV (*hue, saturation, value*). O usuário pode escolher entre as duas opções.

F. *Adaptação do tamanho*

Após segmentar a face que se deseja colocar, deve-se adaptá-la ao tamanho da imagem de destino. Para isso utiliza-se a maior variação dos *landmarks* em x e em y, em relação ao centro, essa mesma característica é utilizada na segmentação do rosto, logo esses valores são conhecidos. Comparando esses valores da face e do destino, sabe-se a proporção em x e em y que deve-se aplicar no reescalonamento da face. Se utiliza interpolação bicúbica para realizar essa adaptação.

G. *Deslocamento espacial e Resultado Final*

Em suma, até agora, têm-se a face, com tamanho compatível ao destino, e o destino e seus *landmarks*. O próximo passo é encontrar *landmarks* da face que se deseja colocar - o *landmark* deve estar adaptado à mudança no tamanho. Assim, utiliza-se as coordenadas dos centros das duas imagens para definir o quanto a face deverá ser deslocada. Deseja-se que os centros estejam alinhados, seja x o deslocamento isso implica que:

$$cen_{face} + x = cen_{dest} \Rightarrow x = cen_{dest} - cen_{face}$$

Logo se desloca toda a face por x e coloca-se no destino, esse será o resultado final do software.

III. Resultados

Em seguida, seguirão imagens de todo o processo de traco de faces:

A. *Detecção de faces e Landmarks*

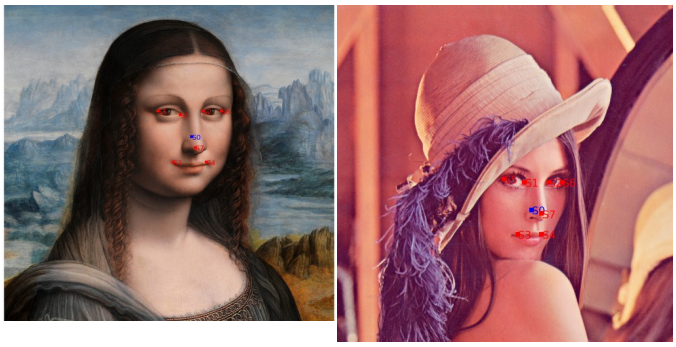


Fig. 2. Inicialmente, se acham os *landmarks* da imagem de destino (2a) e da imagem de onde se retirará a face (2b).

B. *Rotação*

Fig. 3. A imagem do rosto é rotacionada e se acham os *landmarks* da nova imagem

C. *Seam Search*

Escolheu-se segmentar usando uma elipse

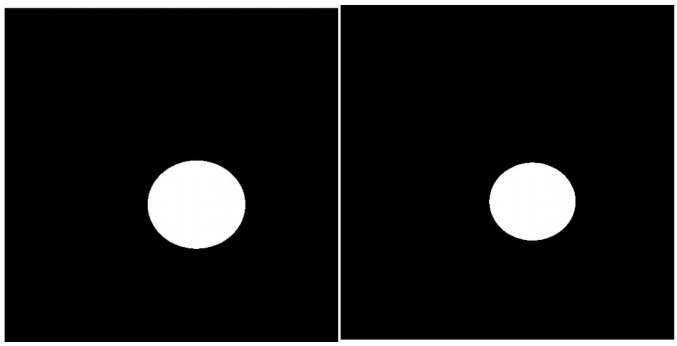


Fig. 4. Imagens de elipses centradas no centro do rosto, tem-se à esquerda (4a) uma elipse grande e à direita (4b) uma menor

Utilizando ou exclusivo:

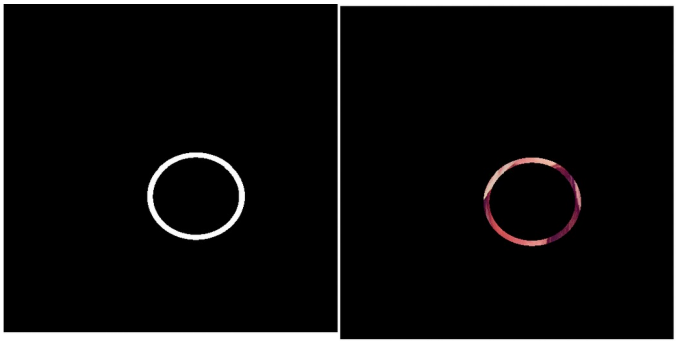


Fig. 5. Obtêm-se uma máscara para a região de interesse (5a), à direita temos à região da imagem que se analisa

D. Djikstra

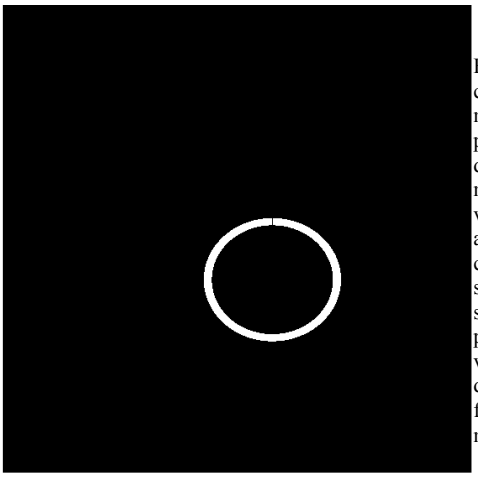


Fig. 6. Faz-se um corte vertical, a mascará resultante será passada ao algoritmo djikstra, que achará um menor caminho com vizinhança de 4 de ambos os lados do corte. Ao achar a segmentação adiciona-se o um ponto pertencente ao centro, vizinho, em vizinhança de 8 do começo e do fim do algoritmo de menor caminho.

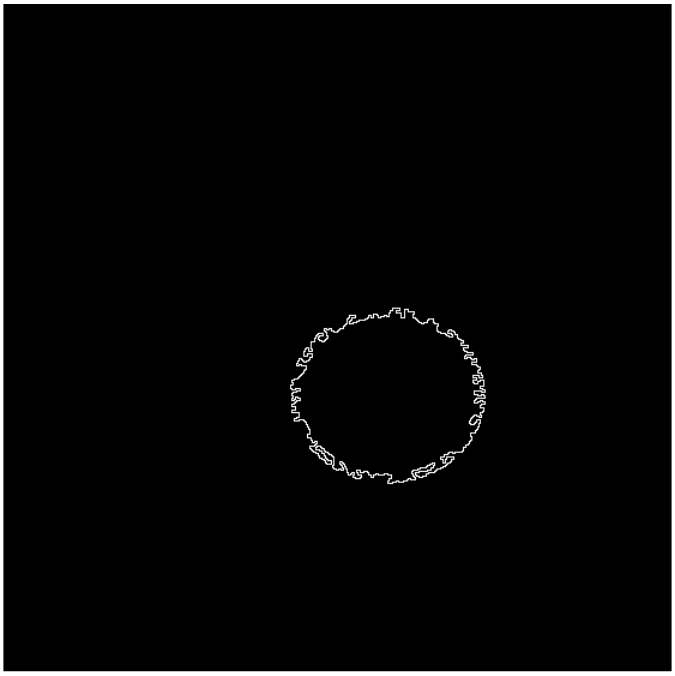
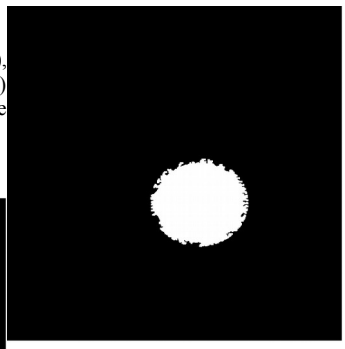


Fig. 7. Obtêm-se a segmentação (7a), na página anterior, preenchendo-a (7b) será a máscara final ao processo de seam search, à direita.



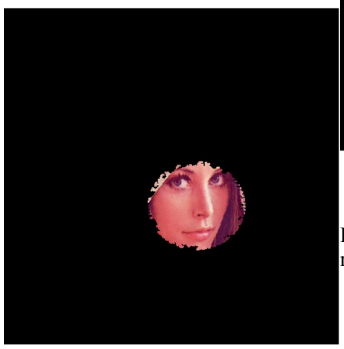
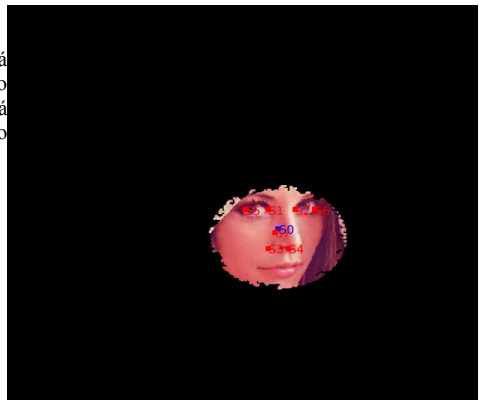


Fig. 8. À esquerda temos o rosto.

E. Adaptação de tamanho

Fig. 9. Perceba que há um erro na detecção do *landmark*, isso causará um deslocamento não tão bom



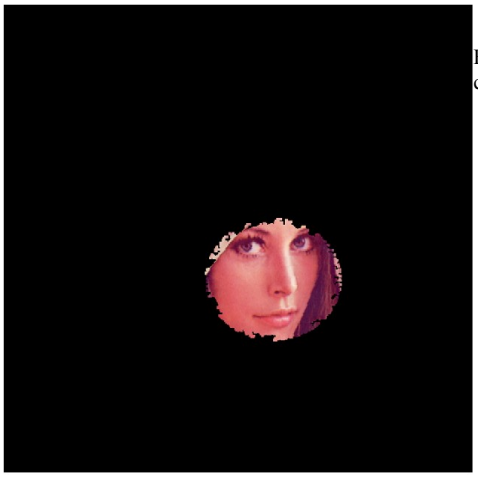


Fig. 10. Figura com tamanho adaptado

F. Deslocamento espacial e resultado Final

Assim obtêm-se o resultado da figura (1)

IV. Uso do software

O software disponibiliza duas funções para uso do usuário que fazem o pedido

A. *auto*

auto(fig1, fig2, opc1, opc2)

- fig1 é o nome do arquivo que contem a figura na qual o rosto será colocado
- fig2 é o nome do arquivo do qual será retirado o rosto
- opc1 é um inteiro que pode ser 1 ou 2, 1 significa uso de segmentação retangular, 2 é o uso de segmentação elíptica. O padrão é o uso da elíptica, ou seja, qualquer outra número colocado usará segmentação elíptica
- opc2 é um inteiro que pode ser 0 ou 1, 0 significa uso de níveis de cinza no algoritmo de menor caminho, 1 é do *hue* da imagem no algoritmo djikstra. O padrão é o uso do *hue*, ou seja, qualquer outra número colocado usará o *hue*

B. *auto_devel*

auto(fig1, fig2, opc1, opc2, op_a, op_b)

- Os quatro primeiros parâmetros são iguais à auto
- op_a permite dimensionar o/a retângulo/elipse maior
- op_b permite dimensionar o/a retângulo/elipse menor
- Essa versão é direcionada à usuários que desejem um maior controle do troca facial

V. Problemas e melhoras

A maioria dos problemas que surgiram foram referentes à erros na detecção de *landmarks*. Logo uma possível melhora é a melhora do algoritmo utilizado, possivelmente usando outra ferramenta diferente do *flandmark*. A detecção de faces inclinadas sendo um bom exemplo desses erros.

Ademais, pode-se utilizar um algoritmo para realizar *color blending*, e assim deixar menos evidente a troca de rostos. Pode-se implementar outra opção de segmentação, visto que a elipse e o quadrado por vezes incluem o fundo de uma imagem.

O uso da melhor opção dentro do banco de dados também é uma enorme melhora. Possivelmente pode-se criar uma versão na qual se testam todas as possíveis segmentações para todos os rostos e comparam-se elas segundo um critério bem definido.

VI. Conclusão

Foi implementado um algoritmo que dada duas imagens, realiza a troca das faces de forma automática. Esse algoritmo é bastante complexo e apresenta muitas limitações, principalmente em relação aos pré-requisitos das imagens para realizar trocas faciais, como não possuir objetos na face e exigir fotos frontais. O algoritmo possui alguns bugs para determinadas faces, devido principalmente o erros de detecção de *landmarks* ou problemas no djikstra divido à imagens muito grandes. Porém é possível obter bons resultados de uma maneira geral para faces não muito diferentes uma das outras.

VII. Referências

[1] M. Uřičář, V. Franc, and V. Hlaváč. "Detector of facial landmarks learned by the structured output SVM." VISAPP (2012): 547-556.

[2] Feng, Yifei; Huang, Wenxun; Wu, Tony. “Face Swapping”