Universidade de São Paulo - Ribeirão Preto

Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto

Trabalho de Processamento de Imagens Médicas

Leonardo do Nascimento 7695815
Luciana Trento Raineri 9266984

Disciplina: Processamento de Imagens Médicas (IBM 1044)
Professor: Luiz Otávio Murta Júnior

Junho de 2016
Ribeirão Preto - SP

1. Proposta

A proposta neste trabalho é a utilização de técnicas de processamento de imagens como uma etapa de pré-processamento para a análise de faces humanas. Essa etapa de pré-processamento consiste em identificar o rosto humano em uma dada imagem e aplicar transformações, de modo a produzir uma imagem final contendo:

- apenas o rosto do indivíduo;
- corrigindo-se uma possível inclinação da cabeça para a direita ou esquerda;
- e o rosto reescalonado para um tamanho de imagem final desejado.

Ao realizar esse pré-processamento em um conjunto de imagens, é gerado um conjunto imagens resultantes contendo apenas faces humanas, alinhadas e de mesmo tamanho (largura e altura da imagem). Para facilitar a implementação dessas tarefas de pré-processamento, é requerido ao usuário informar, para cada imagem, quatro pontos, correspondem:

1. ao ponto do rosto próximo à orelha direita do indivíduo (na altura média da orelha);
2. ao ponto do rosto próximo à orelha esquerda do indivíduo (na altura média da orelha);
3. ao ponto correspondente ao centro da testa do indivíduo, na borda superior da face;
4. ao ponto correspondente ao centro do queixo do indivíduo, na borda inferior da face.

Dessa forma, o rosto não é identificado automaticamente, e sim a partir dos pontos fornecidos pelo usuário. Para produção da imagem resultante contendo apenas a face alinhada do indivíduo, são aplicadas transformações geométricas de translação, rotação e reescalonamento, utilizando interpolação 2D. A implementação do plugin considera imagens coloridas e trabalha com os componentes RGB. A seguir, são descritos os detalhes de implementação e decisões de projeto adotadas neste trabalho.

2. Implementação

O trabalho foi implementado na linguagem Java, na forma de Plugin para o ImageJ. O Plugin foi implementado no arquivo **FaceFitting_.java**, enviado em anexo, junto a este relatório.

2.1 Interface do plugin FaceFitting

A classe **FaceFitting_** estende a classe **Dialog**, e quando acessada pelo menu Plugins do ImageJ, cria e mostra um novo **Frame**, com as instruções necessárias para aplicação das transformações na imagem. A janela é ilustrada na Figura 1.

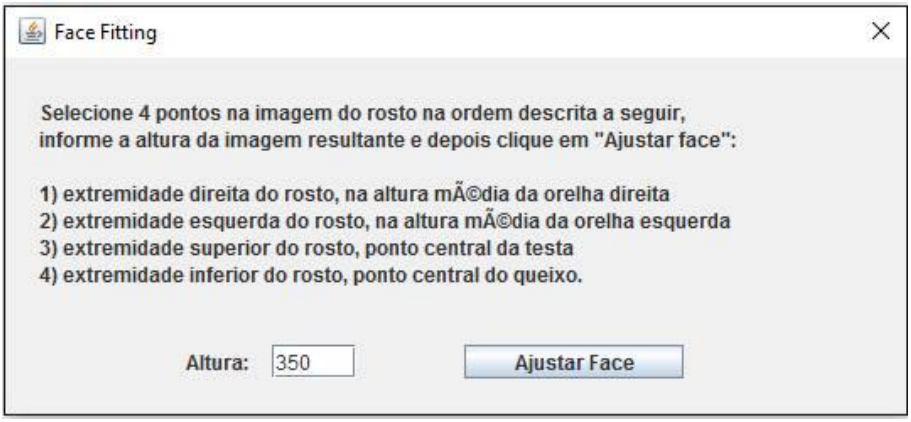


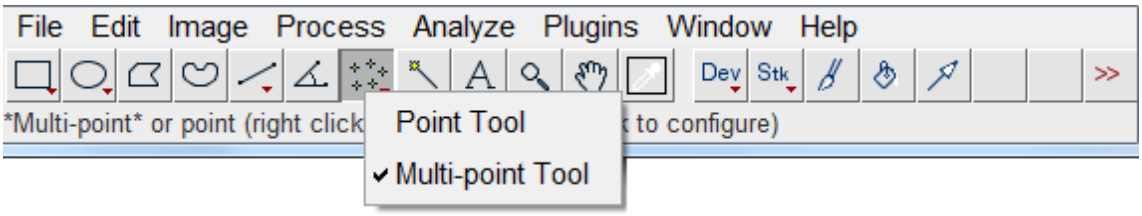
Figura 1. Janela do plugin FaceFitting

Nas instruções, são indicadas as localizações dos 4 pontos que devem ser selecionados na imagem pelo usuário. Essa janela também possui um campo de texto, no qual se deve informar a altura desejada para a imagem a ser produzida após a aplicação do plugin. A largura da imagem final é definida de modo a ser mantida a proporção do rosto da pessoa, definido pelos 4 pontos fornecidos pelo usuário. Por fim, o botão **Ajustar Face** realiza as transformações na imagem de entrada, para produzir a imagem resultante contendo a face do indivíduo.

Antes de clicar no botão Ajustar Face, devem ser atendidas as seguintes condições:

- Deve haver uma imagem aberta no ImageJ.
- Devem ser selecionados 4 pontos na imagem, correspondentes (**nessa ordem**):
    1. à extremidade **direita** do rosto, na altura média da orelha direita;
    2. à extremidade **esquerda** do rosto, na altura média da orelha esquerda;
    3. à extremidade **superior** do rosto, no ponto central da testa;
    4. à extremidade **inferior** do rosto, no ponto central do queixo.
- Deve ser informado valor válido para a **altura** da imagem final.

Para seleção dos 4 pontos, deve-se usar a ferramenta de seleção multi-ponto:



Caso ao menos uma das 3 condições não tenham sidos atendidas, são informadas mensagens de erro.

Um exemplo de seleção dos 4 pontos necessários é ilustrado na Figura 1. É importante notar que a ordem dos pontos deve ser respeitada, sendo 1 - orelha direita, 2 - orelha esquerda, 3 - testa e 4 - queixo. Os pontos 1, 2, 3 e 4 são armazenados nas variáveis de classe **rightP**, **leftSP**, **topSP** e **bottomSP**, respectivamente. O uso dessas variáveis será descrito em detalhes na Seção 2.2.

A classe **FaceFitting_**, além de estender **Dialog**, implementa a interface **WindowListener**, para permitir o fechamento do plugin ao clicar no 'X', localizado na barra superior da janela, através da implementação do método **windowClosing(WindowEvent e)**:

```
public void windowClosing(WindowEvent e) {
    dispose();
}
```

2.2 Implementação da classe FaceFitting

A classe **FaceFitting_** possui 7 variáveis de classe, como mostrado na Figura 2. A função de cada uma dessas variáveis é apresentada a seguir:

```
public class FaceFitting_ extends Dialog implements WindowListener {
    private Point rightSP;  // right selected point - orelha direita
    private Point leftSP;  // left selected point - orelha esquerda
    private Point topSP;    // top selected point - testa
    private Point bottomSP; // bottom selected point - queixo

    private ImagePlus imp;

    private static int newImageWidth;
    private static int newImageHeight;
```

Figura 2. Variáveis da classe FaceFitting_

- **leftSP**, **rightSP**, **topSP**, **bottomSP**: são variáveis de classe **Point**, utilizadas para armazenar as coordendas dos 4 pontos selecionados na imagem, em que as letras **SP** significam "Selection Point". Na Figura 3, são ilustrados os pontos associados a cada uma dessas variáveis na imagem.

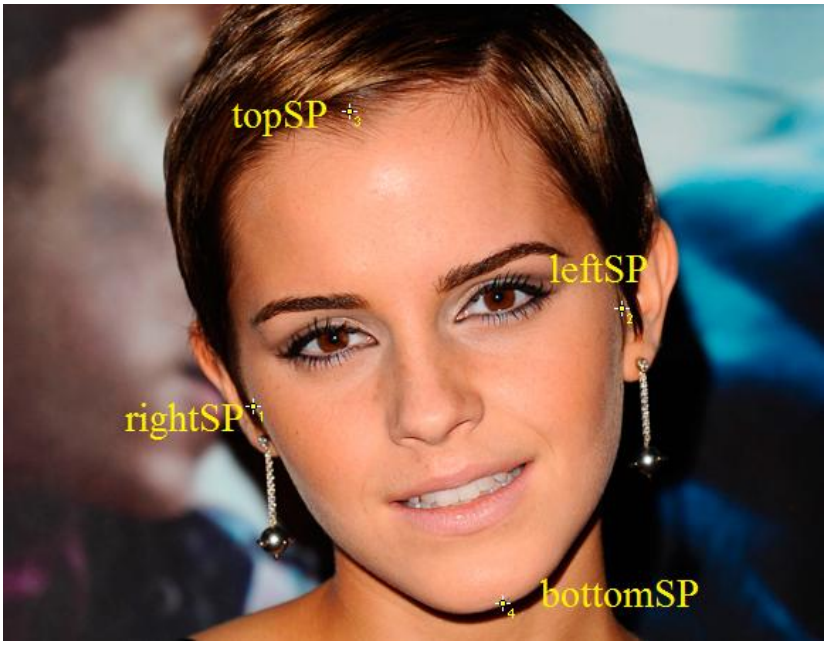


Figura 3. Variáveis associadas a cada ponto de seleção.

- **imp** corresponde a uma variável do tipo **ImagePlus** e é utilizada para armazenar a imagem a ser analisada pelo plugin, que está aberta no ImageJ.
- **newImageWidth** e **newImageHeight** correspondem aos valores de largura e altura da imagem resultante, respectivamente. Vale lembrar que apenas a altura é fornecida pelo usuário na interface do plugin. O valor da altura é calculado posteriormente para manter a proporção das dimensões do rosto.

Apresentadas as variáveis de classe, pode-se detalhar a execução do plugin **FaceFitting**. No construtor da classe, são criados os componentes, como as caixas de texto o botão. Após a instanciação do botão "Ajustar Face", é associada a ele uma ação, de modo que ao ser clicado, ele chama dois métodos, **getSelectedPixels()** e **runTransformations**.

```
JButton button = new JButton("Ajustar Face");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(getSelectedPixels()) runTransformations();
    }
});
```

O método **getSelectedPixels()** é responsável por verificar se os 4 pontos foram corretamente selecionados na imagem. Se não foram, ele retorna **false** e o método **runTransformations()** não é executado. Caso contrário, ele associa cada ponto às varíaveis **leftSP**, **rightSP**, **topSP** e **bottomSP** e retorna **true**, permitindo a executando do método **runTransformations()**.

O método **runTransformations()** é o principal método do plugin, e é onde são feitas as transformações geométricas na imagem original e produção da imagem resultante. Inicialmente, é declarado um **ColorProcessor cp**, que corresponde ao **ImageProcessor** correspondente à imagem aberta, obtido a partir do **ImagePlus imp**.

```
ColorProcessor cp = (ColorProcessor) imp.getProcessor();

double horizontalM = (leftSP.y - rightSP.y) / (double) (leftSP.x - rightSP.x);
double angle = Math.atan(horizontalM);

double sinAngle = Math.sin(angle);
double cosAngle = Math.cos(angle);
```

Em seguida, é criada uma variável **double horizontalM**, que corresponde ao valor de inclinação da reta formada pelos pontos representados por **leftSP** e **rightSP**. Como se sabe que a inclinação da reta corresponde à tan(**angle**), em que **angle** é o ângulo formado entre a reta e o eixo X, conforme ilustrado na Figura 4. Na implementação, **angle** é obtido pelo cálculo do arctang(**horizontalM**).

O valor da variável **angle** é importante para as demais transformações, pois ele indica o ângulo pelo qual a imagem deve ser rotacionada para que os pontos **leftSP** e **rightSP**, após a rotação, localizem-se na mesma reta horizontal (mesmo valor da coordenada **y**). As coordenadas dos pontos **leftSP** e **rightSP** após a rotação terem o mesmo valor de **y** indica que o rosto do indivíduo foi corretamente alinhado após a rotação (assumindo-se que a seleção de pontos pelo usuário foi realizada corretamente). Para a posterior aplicação da transformação de rotação, são armazenados os valores de seno(**angle**) e cosseno(**angle**) nas variáveis **sinAngle** e **cosAngle**, respectivamente.

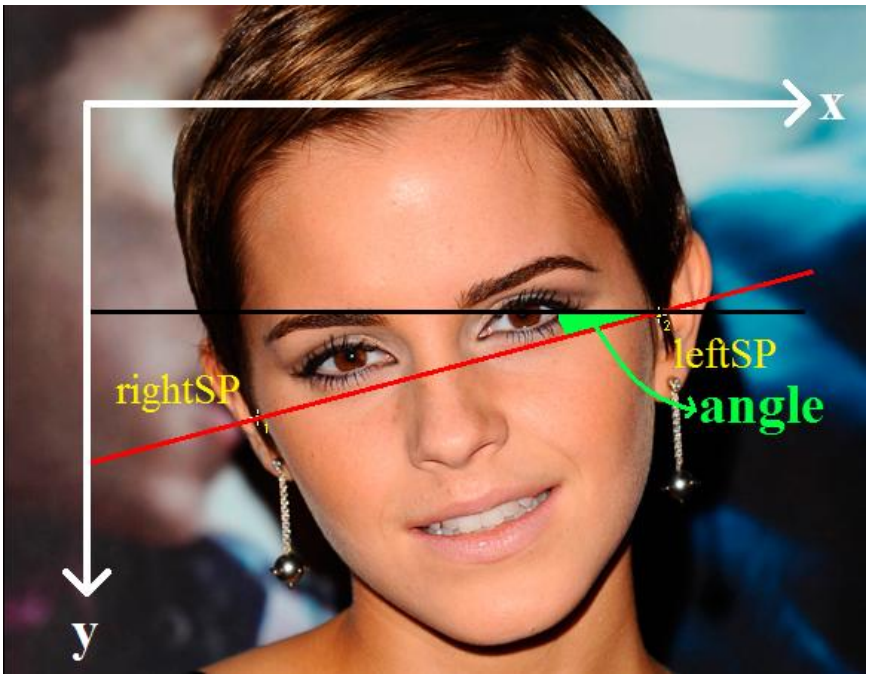


Figura 4. Cálculo do ângulo de inclinação da reta entre rightSP e leftSP

O próximo passo é calcular as coordenadas do ponto **upperLeftCorner**, que corresponderá ao ponto extremo esquerdo superior (x = 0, y = 0) da imagem após a rotação e a translação. O ponto **upperLeftCorner** é ilustrado na Figura 5.

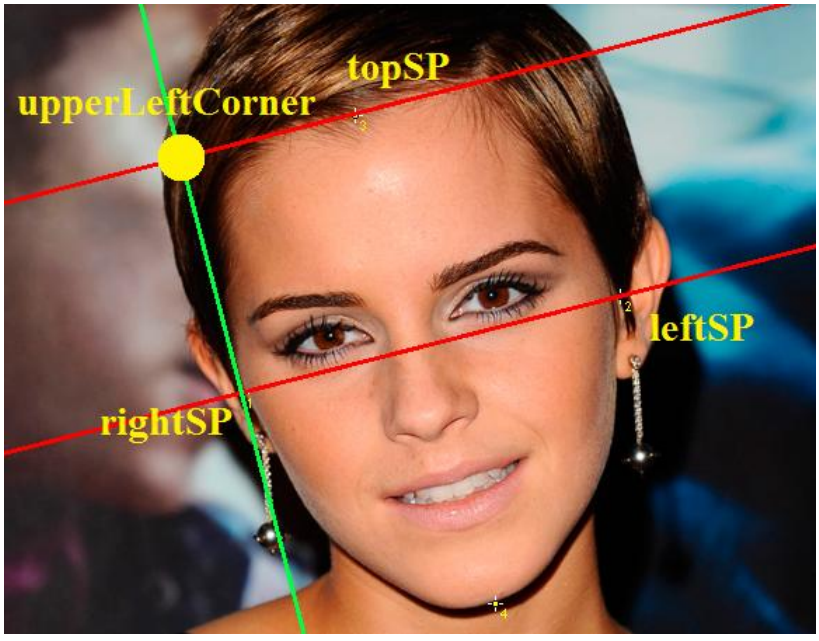


Figura 5. Localização do upperLeftCorner

O ponto **upperLeftCorner** corresponde ao ponto comum entre duas retas:

- $y - topSP.y = horizontalM * (x - topSP.x)$

  corresponde à reta que passa pelo ponto **topSP** e tem inclinação **horizontalM**, a mesma inclinação da reta que passa pelos pontos **rightSP** e **leftSP**.

- $y - rightSP.y = (\frac{-1}{horizontalM}) * (x - rightSP.x)$

  corresponde à reta que passa por **rightSP** e é perpendicular à reta que passa pelos pontos **rightSP** e **leftSP**, e portanto tem inclinação -1/**horizontalM**.

O código correspondente ao cálculo de **upperLeftCorner** foi implementado no método **calculateUpperLeftCorner**:

```
private Point calculateUpperLeftCorner(double horizontalM) {
    double verticalM = - 1 / horizontalM;

    double x = (topSP.y + verticalM * rightSP.x - horizontalM * topSP.x - rightSP.y) /
                                                        (verticalM - horizontalM);
    double y = horizontalM * (x - topSP.x) + topSP.y;

    return new Point((int) Math.round(x), (int) Math.round(y));
}
```

Nesse momento, com as informações de **leftSP**, **rightSP**, **topSP**, **bottomSP** e **upperLeftCorner**, já é possível visualizar a região de interesse que fará parte da imagem final, que está inserida no retângulo com bordas azuis, como ilustrado na Figura X. Pode-se afirmar que esses quatro segmentos de reta formam um retângulo, pois os segmentos dos lados do rosto são perpendiculares aos segmentos de reta da testa e queixo.

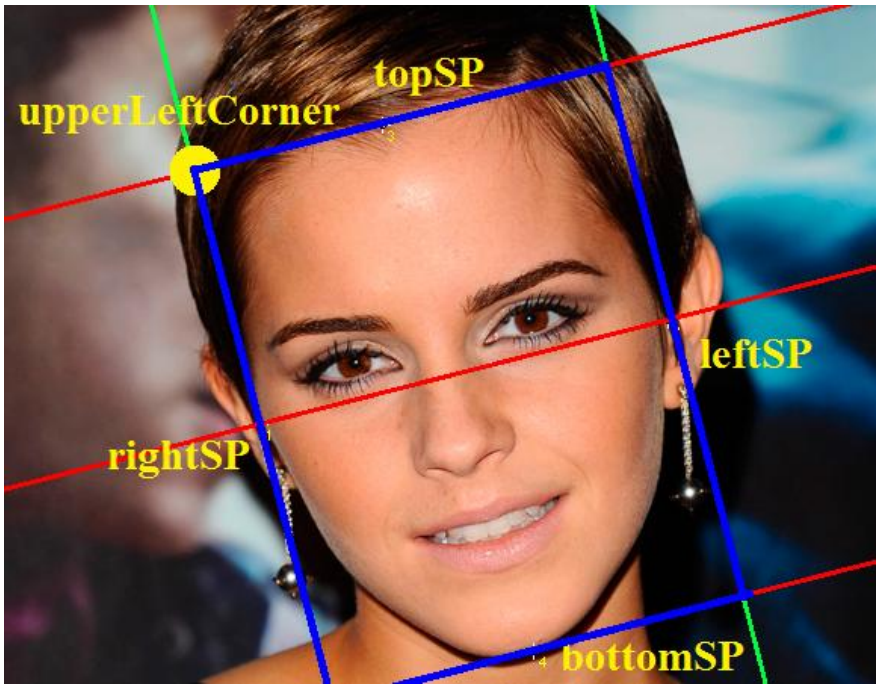


Figura 6. Região de interesse dentro do retângulo em azul.

O último passo que precede a aplicação das transformações geométricas é o cálculo da altura e largura do retângulo em azul, para que possa ser aplicada a transformação de reescalonamento. Para isso, foi feita a rotação apenas dos pontos **leftSP**, **rightSP**, **topSP** e **bottomSP**, e então usadas suas novas coordenadas para o cálculo da largura e altura, sendo a **largura** corresponde a diferença entre o componente **x** dos pontos **leftSP** e **rightSP**, após a rotação; e a **altura** sendo a diferença entre o componente **y** dos pontos **topSP** e **bottomSP**, após a rotação. O trecho de código correspondente a esse cálculo é mostrado a seguir:

```
int width = (int) Math.round((Math.cos(-angle) * leftSP.x - Math.sin(-angle) * leftSP.y)
                    - (Math.cos(angle) * rightSP.x - Math.sin(-angle) * rightSP.y));
int height = (int) (Math.round(Math.sin(-angle) * bottomSP.x + Math.cos(-angle) * bottomSP.y)
                    - Math.round(Math.sin(-angle) * topSP.x + Math.cos(-angle) * topSP.y));
```

Vale notar que a rotação desses pontos foi feita usando o ângulo **-angle**, que corresponde ao valor negado de **angle**, dado que a rotação de **leftSP**, **rightSP**, **topSP** e **bottomSP** para as coordenadas na imagem final corresponde à transformação inversa da que é realizada no plugin, que é da imagem alvo para a imagem fonte.

Utilizando os valores de largura e altura do rosto, e a informação da altura da imagem final, armazenada em **newImageHeight**, é calculado valor da largura final da imagem, e armazenado na variável **newImageWidth**, como mostrado a seguir:

```
double div = (double)width/height;
newImageWidth = (int)Math.round(newImageHeight*div);
```

Para aplicação das transformações geométricas, foi feita uma passagem por cada pixel da imagem, implementado como dois comandos **for** aninhados, iterando pelas colunas no **for** externo, e pelas linhas no **for** interno. Essa iteração é feita para o número de linhas e colunas da imagem de saída, que foram informados pelo usuário na interface do plugin. Para gerar a nova imagem, é instanciada uma variável **newCp**, da classe **ColorProcessor**, dados que as imagens analisadas são coloridas.

```
ColorProcessor newCp = new ColorProcessor(newImageWidth, newImageHeight);
for (int x = 0; x < newImageWidth; x++) {
    for (int y = 0; y < newImageHeight; y++) {
```

Então, para cada pixel **(x, y)** da nova imagem a ser gerada, são calculadas suas coordenadas na imagem de entrada, considerando as seguintes transformações geométricas:

- translação;
- rotação;
- redimensionamento.

Para a translação, o objetivo é deslocar a imagem original de maneira que o pixel de coordenada (0, 0) na imagem resultante, corresponda ao pixel **upperLeftCorner** na

imagem original. Para isso, as variáveis **dx** e **dy** armazenam os valores a serem deslocados nos eixos **x** e **y**, respectivamente. O valor de **dx** equivale à coordenada **x** de **upperLeftCorner**, e **dy** corresponde à coordenada **y** do mesmo pixel. É importante ressaltar que o objetivo é realizar a transformação da imagem resultante para a original (denominado **target-to-source**, em inglês). Por isso, Ao invés de subtrair **dx** e **dy** das coordenadas dos pixels na imagem original (**source**), é feito o acréscimo desses valores nas coordenadas de cada pixel da imagem resultante (**target**), de modo a encontrar quais são as coordenadas correspondentes a ela, na imagem original.

$$x = u + dx$$

$$y = v + dy$$

em que **u** e **v** correspondem às coordenadas na imagem original e **x** e **y** às coordenadas na imagem resultante.

No caso da rotação, a transformação consiste em rotacionar as coordenadas de cada pixel da imagem resultante pelo ângulo **angle**. Para isso, essas coordenadas são calculadas da seguinte maneira:

$$x = \cos(angle) * u - \sin(angle) * v$$

$$y = \sin(angle) * u + \cos(angle) * v$$

Para o redimensionamento, são usadas duas informações:

1. largura e altura da imagem final, armazenadas em **newImageWidth** e **newImageHeight**;
2. largura e altura da área da face (área do retângulo azul na Figura 6), armazenadas em **width** e **height**.

Assim, é possível obter os fatores de redimensionamento **xScale** e **yScale**:

$$xScale = \frac{width}{newImageWidth} \qquad yScale = \frac{height}{newImageHeight}$$

Usando os fatores de redimensionamento, é possível calcular as coordenadas pela transformação:

$$x = xScale * u$$
$$y = yScale * v$$

Após definidas as três transformações, pode-se expressá-las usando a seguinte multiplicação de matrizes:

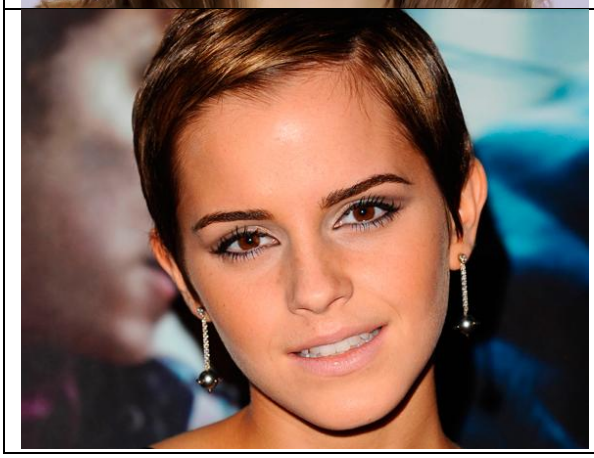
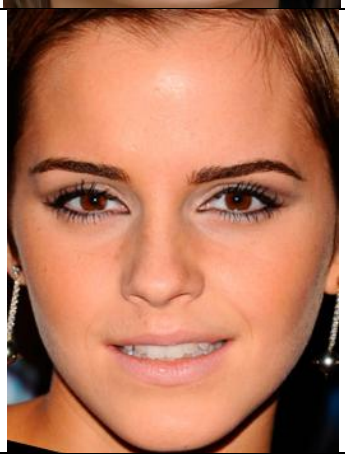
$$\begin{bmatrix} x \\ y \\ k \end{bmatrix} = \begin{bmatrix} xScale * \cos(angle) & -xScale * \sin(angle) & dx \\ yScale * \sin(angle) & yScale * \cos(angle) & dy \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Como os valores calculados **x** e **y** são reais, foi aplicada a interpolação bilinear para cada pixel (**u**, **v**). É importante ressaltar que, como as imagens analisadas são coloridas, os cálculos de interpolação são feitas para cada banda, Vermelha, Verde e Azul (RGB).

Por fim, após a interpolação bilinear o novo valor calculado é inserido na imagem resultante. Após a inserção de todos os pixels, é criada uma variável **ImagePlus newImp** e é chamado o método **show()**, para permitir a visualização da imagem resultante no ImageJ.

3. Aplicação

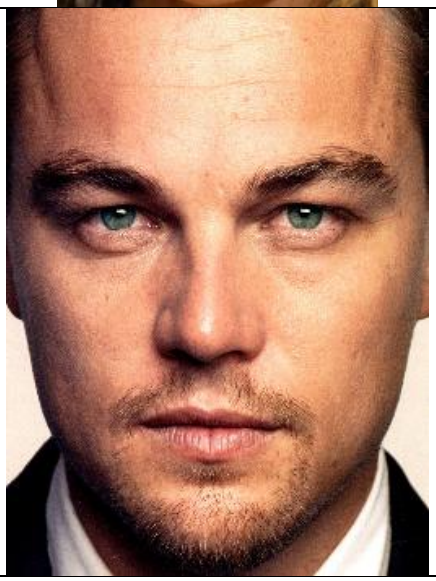
A seguir, são exibidos alguns exemplos da aplicação do plugin **FaceFitting** em imagens adquiridas na internet.

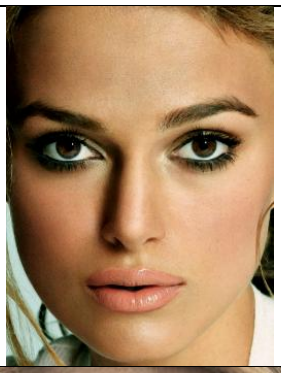
Imagem original	Imagem resultante
	
	

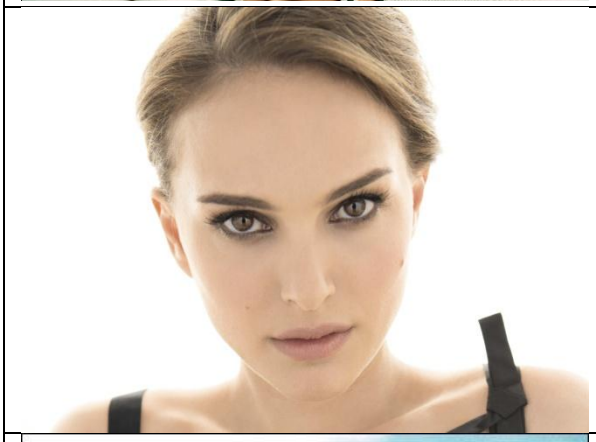


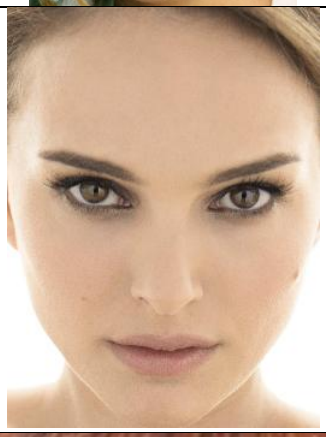


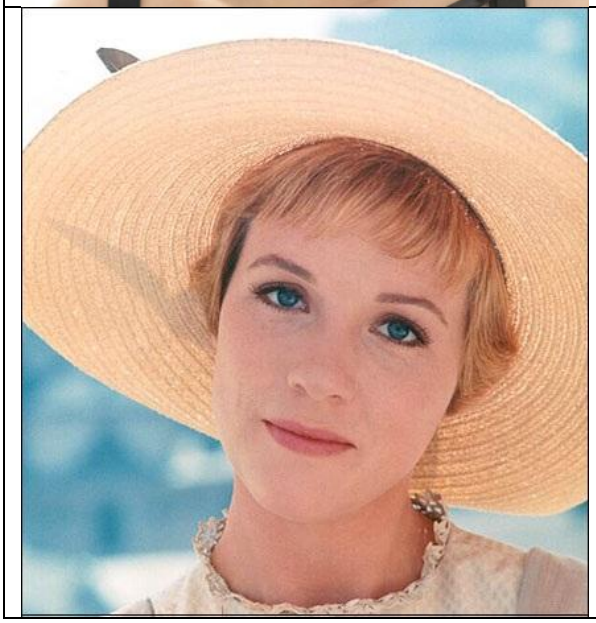


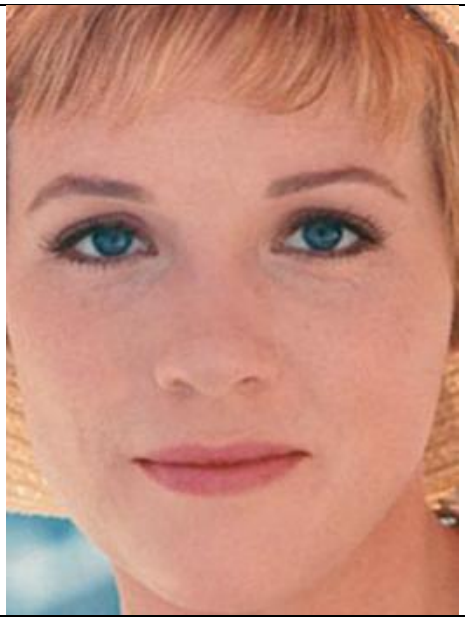



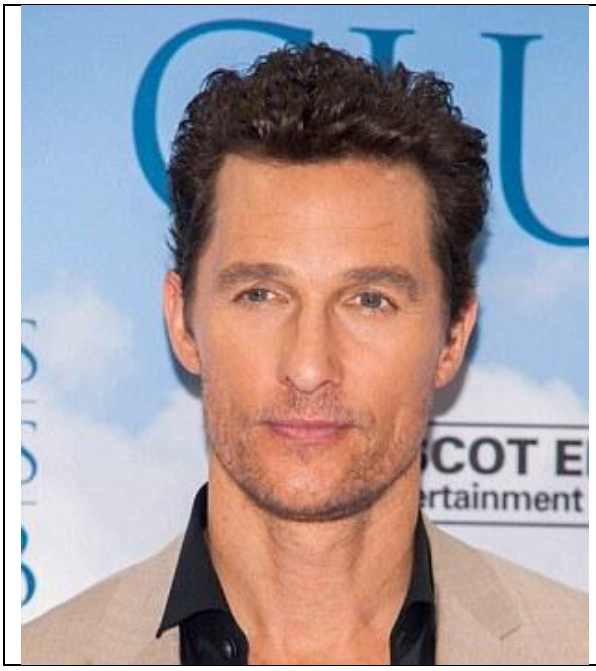











COT E
rtainment