Relatório - Atividade 3

Universidade Estadual de Campinas
Instituto de Computação
Introdução ao Processamento Digital de Imagens

Maio de 2018

Felippe Trigueiro Angelo
210479

1 Descrição geral da atividade 3

A atividade 3 da disciplina Introdução ao Processamento Digital de Imagens teve como objetivo a implementação de dois algoritmos para alinhamento automático de documentos.

Na atividade em questão, o usuário fornece uma imagem que contém um documento de texto que está desalinhado. Em seguida ele seleciona o método ao qual ele deseja executar o alinhamento e o programa retorna uma imagem com o documento alinhado, assim como exibe no terminal, o ângulo de rotação utilizado.

2 Executando o Programa

Para executar o programa de alinhamento automático, entre no terminal do linux, navegue até a pasta onde os códigos estão localizados e digite o seguinte comando:

```
sudo python3 alinhar.py [imagem-de-entrada] [algoritmo-de-alinhamento] [imagem-de-saída]
Ex: sudo python3 alinhar.py sample1.png 1 sample1_align1.png
```

2.1 Dados de Entrada

Para a correta execução do programa é necessário que alguns cuidados sejam tomados com os dados de entrada:

- As imagens de entrada, devem estar na mesma pasta em que está contido o código-fonte;
- Para que o alinhamento final não resulte em uma imagem invertida é necessário que a imagem de entrada também esteja desalinhada na posição vertical. Este não é um requisito proibitivo à utilização do programa, mas imagens desalinhadas em outras posições, não resultarão em uma imagem vertical, o que irá requerer um pós processamento.
- As imagens de entrada podem estar no padrão RGB, RGBA;
- Na seleção do algoritmo de alinhamento, convencionou-se o seguinte:
    1. Algoritmo de alinhamento por Projeção horizontal;
    2. Algoritmo de alinhamento por Transformada de Hough.
- As imagens que serão alinhadas pelo método da Trasnformada de Hough são tratadas como se estivessem com os caracteres do texto alinhados.
- O formato da imagem de saída deve ser .png.

2.2 Dados de Saída

Após o algoritmo selecionado ter executado o alinhamento, o programa exibe duas imagens, onde a primeira é a imagem fornecida como entrada e a segunda é a imagem alinhada. A imagem alinhada também é salva na pasta Output_Images que é criada no mesmo diretório do código fonte. Por fim, no terminal é exibido o ângulo de alinhamento que foi utilizado pelo algoritmo.

3 Descrevendo o Código

3.1 Algoritmos e Estruturas de Dados utilizadas

3.1.1 Algoritmo Baseado em Projeção Horizontal

O primeiro algoritmo de alinhamento que foi implementado, foi o algoritmo baseado em Projeção Horizontal. O mesmo pode ser selecionado por meio da entrada 1 no campo [algoritmo-de-alinhamento].

Esta técnica consiste em rotacionar a imagem em uma série de ângulos e verificar qual rotação otimiza uma função objetivo escolhida. A figura 1 mostra um comparativo entre a projeção do documento desalinhado e o documento alinhado. É possível ver que o caso alinhado, resulta em picos maiores do que a versão desalinhada, isso ocorre devido à diminuição de pixels pretos nas

linhas da imagem, assim os mesmos ficam juntos em uma região da imagem, resultado assim no aumento dos picos da projeção.

Figure 1: Comparação entre as projeções horizontais das imagens desalinhada e alinhada.

No trabalho em questão, foi utilizado como função objetivo a soma quadrática das diferenças entre os valores das células do adjacentes no perfil de projeção. A projeção horizontal é calculada por meio do somatório dos valores contidos em uma linha e pode ser vista na equação 1. Já a função objetivo utilizada foi calculada por meio da equação 2.

$$Proj[m] = \sum_{i=0}^{N-1} I[m][i] \tag{1}$$

$$SquareError = \sum_{i=0}^{M-2} (Proj[i+1] - Proj[i])^2 \tag{2}$$

A implementação desse algoritmo foi feita por meio da criação da função *align_projection()* que foi inserida no arquivo *align_algorithms.py*. A descrição da função *align_projection()* é feita a seguir.

```
def align_projection(input_Image):
    monImage = color.rgb2gray(input_Image)

    maxHistRotate = np.zeros(2)
    for k in range(1, 181):
        binImRotate = transform.rotate(monImage, k)
        binImRotate = binImRotate <= 0.8
        binImRotate = binImRotate*1

        projHist = np.sum(binImRotate, axis=1)

        #Cálculo das diferenças entre células adjacentes da projeção
        difProj = projHist[0:(len(projHist)-1)] - projHist[1:len(projHist)]
        squareError = np.sum(difProj*difProj)

        #Verifica qual o erro máximo
        if(squareError > maxHistRotate[0]):
            maxHistRotate[0] = squareError
            maxHistRotate[1] = k

    angle = maxHistRotate[1]
    if (angle > 90 and angle < 180):
       angle = angle + 180

    transformedImage = transform.rotate(input_Image, angle, resize=True)

    return transformedImage, angle
```

Inicialmente a imagem que está no padrão RGB é convertida para uma imagem monocromática, facilitando assim a obtenção da projeção horizontal. Após a conversão, a imagem monocromática é rotacionada em uma faixa de ângulos que varia de 0 até 180, permitindo verificar todos os possíveis ângulos de desalinhamento.

Com a obtenção da imagem rotacionada, um procedimento de binarização da imagem é realizado com a finalidade de transformar os pixels pretos no valor 1 e os pixels brancos no valor 0, facilitando assim o cálculo das projeção e o posterior cálculo de um valor correto da função objetivo.

Para o cálculo da projeção horizontal da imagem obtida ao fim de cada rotação, foi utilizada a função *np.sum()* presente na biblioteca *numpy*. A função recebe como parâmetros a matriz que representa a image e o *axis*, que indica a direção que se deseja realizar a soma, no caso em questão, somas ao longo de uma linha da matriz.

Após o cálculo da projeção, a função objetivo é calculada para cada rotação e comparada com o resultado da rotação anterior com a finalidade de se obter o ângulo que resultou no maior erro quadrático.

Ao fim das 180 rotações, é verificado se o ângulo de rotação está entre 90 e 180, o que indica que a imagem irá ser alinhada de forma invertida. Assim, para corrigir esse problema, caso o ângulo esteja nesse intervalo, ele é acrescido de 180 graus. Assim, a função retorna a imagem rotacionada e o ângulo de rotação utilizado.

É importante citar que na rotação final, a imagem é redimensionada com a finalidade de preservar o seu conteúdo.

A figura 2 demonstra o resultado da aplicação do algoritmo baseado em Projeção Horizontal. É possível ver que após a rotação, o algoritmo retornou o retângulo com as dimensões da imagem antes da rotação. Tal problema foi contornado por meio da função *border_remove()* que será descrita posteriormente.

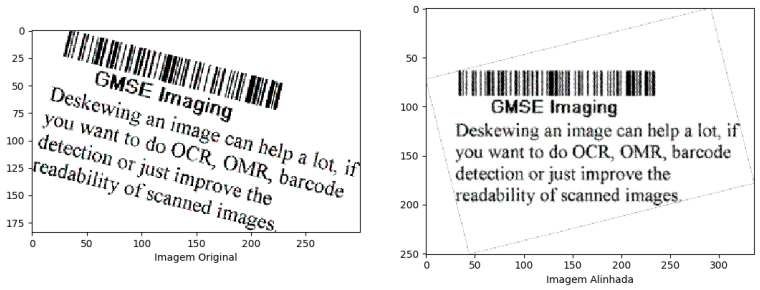

Figure 2: Exemplo da aplicação do algoritmo baseado em Projeção Horizontal.

Para a verificação da confiabilidade do algoritmo, a imagem de saída foi inserida em um Detector Óptico de Caracteres (OCR). Como resultado foi obtido o texto a seguir:

```
111111111111 111111111111
GMSE Imaging Deskewing an image can help a lot, if,, you want to do
OCR, OMR, barcode detection or just improve the readability of
scanned irnage_s.
```

Foi possível ver que mesmo com as bordas o OCR conseguiu reconhecer o texto com uma precisão bastante considerável. Entretanto, mais adiante poderá ser visto que o uso da função para retirada de bordas, melhora esse resultado.

3.1.2 Algoritmo Baseado na Transformada de Hough

O segundo algoritmo de alinhamento que foi implementado foi o algoritmo baseado na Trasnformada de Hough. Essa transformada permite a localização de pontos que pertencem a uma determinada curva, podendo assim ser utilizada para a detecção do ângulo de desalinhamento da imagem.

Pixels que são colineares podem ser vistos como uma reta, assim, caracteres que estão alinhados, quando mapeados no plano de Hough, resultam em picos e permitem, portanto, a detecção do ângulo de desalinhamento.

A implementação desse algoritmo foi feita por meio da criação da função *align_hough()* que foi inserida no arquivo *align_algorithms.py*. A descrição da função é feita a seguir.

```
def align_hough(input_Image):
    #Converte a imagem do padrão RGB para uma imagem binária
    monImage = color.rgb2gray(input_Image)
    monImage = feature.canny(monImage)

    houghTransform, angles, d = transform.hough_line(monImage)
    hTP, angles, d = transform.hough_line_peaks(houghTransform, angles, d)

    angles = np.rad2deg(angles) + 90
    angles = np.round(angles, 0)
    angles = angles.astype(int)

    angle = np.bincount(angles).argmax()

    if (angle > 90 and angle < 180):
        angle = angle + 180

    transformedImage = transform.rotate(input_Image, angle, resize=True)

    return transformedImage, angles[0]
```

Assim como no algoritmo baseado na projeção horizontal, inicialmente foi realizada um pré-processamento na imagem. Primeiro a imagem que estava no modelo RGB foi convertida para uma imagem monocromática com a finalidade de facilitar o cálculo da transformada de Hough, já que a única informação importante no presente contexto é a localização dos pixels pretos pertencentes ao texto. Após a obtenção da imagem monocromática, essa imagem é passada por um detector de bordas com a finalidade de se obter um mapa de bordas, permitindo assim à transformada de Hough encontrar as linhas do texto. O detector de bordas utilizado foi o filtro de Canny, que além de detectar bordas também permite a remoção de algum ruído presente na imagem.

Feito o pré-processamento da imagem que contém o documento, o cálculo da transformada de Hough é realizado. Para tal, é utilizada a função *hough_line()* presente no pacote *transform* da biblioteca *skimage*. A função recebe como entrada uma imagem contendo as linhas, onde os valores não nulos devem representar as bordas e retorna o plano de Hough, bem como os valores de distância e os ângulos obtidos.

Calculado o plano de Hough, se faz necessário a obtenção do valor de uma função objetivo que permita a detecção do ângulo de desalinhamento. Para o trabalho em questão foi utilizado ângulo de maior ocorrência nas linhas dominantes retornadas pela trasnformada de Hough.

Assim, para o cálculo das linhas dominantes foi utilizada a função *hough_line _peaks()* presente no pacote *transform* da biblioteca *scikitimage*. A função recebe como parâmetro os valores retornados pela função *hough_line()*, ou seja, o plano de Hough, bem como os ângulos e disâncias. Assim, a função retorna os ângulos das linhas dominantes presentes na imagem.

Calculado o ângulo de maior ocorrência nas linhas dominantes, é feita a mesma verificação que foi feita no algoritmo da projeção horizontal: se o ângulo está na faixa que vai de 90 a 180 graus, com a finalidade de evitar que a imagem fique invertida. Assim, finalmente a imagem é rotacionada e o ângulo de rotação utilizado é também retornado pela função. Assim como no caso do algoritmo baseado na projeção Horizontal, a imagem obtida aqui também possui as bordas resultantes da rotação. A figura 3 mostra o resultado da aplicação desse algoritmo. É possível ver que o resultado obtido para a imagem em questão é idêntico ao do algoritmo anterior. Portanto,

assim como na imagem anterior a imagem alinhada da figura 3 foi passada por um reconhecedor óptico de caracteres (OCR) obtendo-se um resultado idêntico ao da imagem do algoritmo anterior.

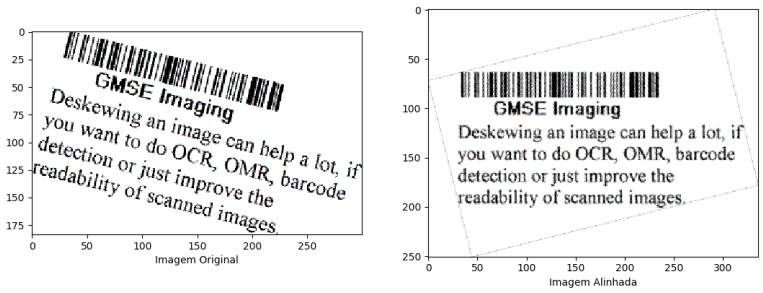

Figure 3: Exemplo da aplicação do algoritmo baseado na Transformada de Hough.

3.1.3 Algoritmo para Remoção das Bordas resultante da Rotação

Como foi visto nos algoritmos anteriores, a rotação da imagem resulta no aparecimento de bordas com as dimensões e posição da imagem no angulo anterior. Tais bordas podem prejudicar o reconhecimento automático dos caracteres, assim foi criada a função *border_remove()* para a remoção das mesmas. A função que também está presente no arquivo *align_algorithms.py* é demonstrada a seguir.

```
def border_remove (transformedImage):
    transformedImage2 = np.copy(transformedImage)
    transformedImage3 = np.copy(transformedImage)
    transformedImage2[transformedImage2 == 0] = 1

    transImGray = color.rgb2gray(transformedImage2)
    BintransImage = transImGray <= 0.8
    BintransImage = BintransImage*1

    projHori = np.sum(BintransImage, axis=1)
    projVert = np.sum(BintransImage, axis=0)

    projHori[projHori < 40 ] = 0
    projVert[projVert < 14 ] = 0

    sup = np.argmax(projHori > 0) + 1
    left = np.argmax(projVert > 0) + 1

    projHori = projHori[::-1]
    projVert = projVert[::-1]

    inf = np.argmax(projHori > 0) + 1
    right = np.argmax(projVert > 0) + 1

    transformedImage3[0:(sup-2), :] = 1
    transformedImage3[:, 0:(left-2)] = 1
    transformedImage3[(2-inf):, :] = 1
    transformedImage3[:, (4-right):] = 1

    return transformedImage3
```

A função recebe como argumento a imagem rotacionada contendo as bordas e retorna a imagem sem as bordas. Para realizar tal procedimento, o algoritmo supõe que a imagem contém apenas um texto na formatação justificada, que consiste em um texto disposto em um formato retangular. Assim, o algoritmo busca encontrar o menor retângulo que contém o texto. Isso é feito por meio dos cálculos das projeções horizontal e vertical da imagem. Nas projeções, é esperado que as regiões que não contenham texto, resultem em projeções muito menores do que o das regiões que contenham texto. Assim, por meio da utilização de um limiar para cada uma das projeções, as regiões pertencentes ao texto foram calculadas e todos os pixels que não são pertencentes a ela foram mapeados com o valor 1, tornando-os brancos.

A figura 4 demonstra a aplicação desse algoritmo à imagem da figura 3. É possível ver que para esta imagem, ainda existem resquicíos da borda devido a forma não-retangular do texto.

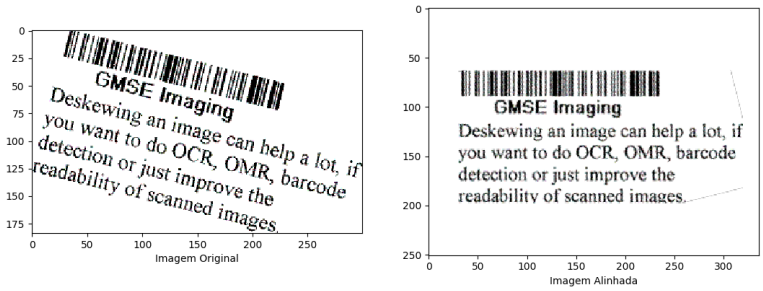

Figure 4: Aplicação do algoritmo para remoção das bordas de rotação.

A imagem alinhada da figura 4 foi passada por um OCR e foi obtido o seguinte resultado:

```
111111111111
GMSE Imaging Deskewing an image can help a lot, if you want to do OCR,
OMR, barcode detection or just improve the readability of scanned
image& -
```

É possível ver que com relação ao resultado obtido pelo OCR na sessão 3.1.1, o resultado melhorou consideravelmente.

3.2 Estruturas de dados utilizadas

No código, foram utilizadas as estruturas de dados array e lista no armazenamento das informações desejadas. No caso das imagens foram utilizados os arrays providos pela biblioteca *numpy*, gerando arrays da classe *ndarray*. Como estamos utilizando imagens de entrada no formato RGB ou RGBA, elas são representadas como arrays do tipo MxNx3 ou MxNx4, onde cada matriz MxN representa um canal do RGB ou RGBA.

As listas foram utilizadas no armazenamento das projeções horizontal e vertical necessárias no algoritmo de remoção de bordas e no algoritmo baseado em projeção horizontal. Elas também foram utilizadas no armazenamento dos ângulos referentes as linhas dominantes.

4 Testes adotados e Limitações do programa

Foram feitos testes com as imagens contidas no endereço da página da disciplina. Portanto, foram testadas imagens quadradas, retangulares no formato .png e no modelo de cores RGBA. Também foram feitos testes com imagens monocromáticas, entretanto apesar de haver um correto alinhamento, as cores mudam consideravelmente, assim preferiu-se adotar a convensão de que as imagens de entrada estivessem no modelo RGB ou RGBA.

Convém citar novamente aqui, que foi adotada a convenção de que as imagens de entrada estavam desalinhadas na posição vertical, permitindo assim que as imagens sejam alinhadas na posição correta, ao invés da posição invertida. Caso seja posto como entrada alguma imagem desalinhada em algum outro formato, o programa irá alinhá-la em algum dos eixo, fazendo assim necessária algum pós processamento para colocar a imagem na posição vertical.

A partir dos resultados dos testes é possível afirmar que ambos os algoritmos conseguem resultados similares no que se diz respeito ao alinhamento. Quanto ao detector de bordas é possível afirmar que ele produz resultados satisfatórios mesmo em imagens em um formato não-retangulares. Entretanto, para imagens com o texto no formato retangular, o resultado é bastante preciso como pode ser visto na figura 5.

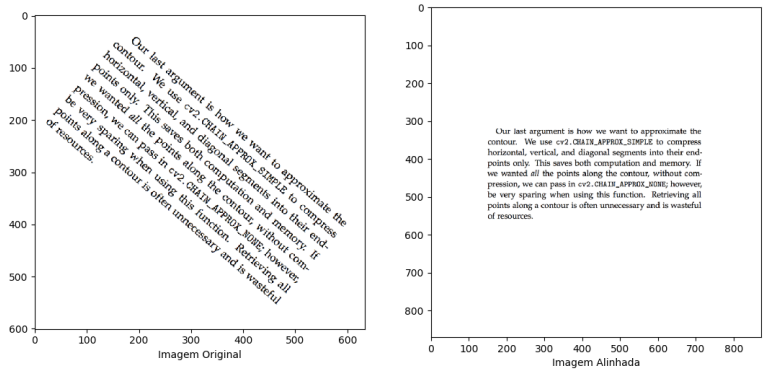

Figure 5: Aplicação do algoritmo para remoção das bordas de rotação em imagens com o texto retangular.

5 Referências Bibliográficas

1. **The Python Standard Library.** Disponível em: https://docs.python.org/3/library/index.html
2. **The Matplotlib API Pyplot.** Disponível em: https://matplotlib.org/api/pyplot_summary.html
3. **Module: Transform - Skit-image Documentation** Disponível em: http://scikit-image.org/docs/dev/api/skimage.transform.html
4. **Free Online OCR** Disponível em: https://www.onlineocr.net/