Universidade Estadual de Campinas

Instituto de Computação

Introdução ao Processamento Digital de Imagem (MC920 / MO443)

Trabalho 3

Leodécio Braz - 230219

22 de maio de 2019

1 Especificação do Problema

O objetivo deste trabalho é aplicar operadores morfológicos para segmentar regiões compreendendo texto e não texto em uma imagem de entrada.

2 Script

O script foi desenvolvido em Python na versão 3.0 e foi utilizado o framework Jupyter para documentação do mesmo. As bibliotecas utilizadas foram **OpenCV** [3], **Numpy**[2] e **matplotlib**[1].

2.1 Entrada

As imagens de entrada estão no formato PBM (*Portable BitMap*) e foram disponibilizadas pelo professor no endereço que se encontra em sua página[1]

As imagens estão localizadas no diretório $./imagens\_morfologia/$, e já se encontram declaradas no código.

2.2 Saída

Após a execução, o script irá gerar tanto resultados intermediários visíveis na tela, quanto as imagens resultante dos processos que serão salvas com o formato PBM (*Portable BitMap*) no diretório **./output/**.

[1] $http://www.ic.unicamp.br/\ helio/imagens_morfologia/$

2.3 Leitura das imagens e declaração dos kernels

As imagens de entrada foram lidas através da função **imread** da biblioteca do OpenCV, que gera um *Array* do Numpy de dimensões MxN que corresponde ao tamanho da imagem.

Os elementos estruturantes (ou *kernels*), também no formato de *Array* do Numpy, foram declarados manualmente no código e são representados como: **kernel_1**, com 1 pixel de altura e 100 pixels de largura, **kernel_2**, com 200 pixels de altura e 1 pixel de largura, **kernel_3**, com 1 pixel de altura e 30 pixels de largura e **kernel_4**, com 1 pixel de altura e 5 pixels de largura.

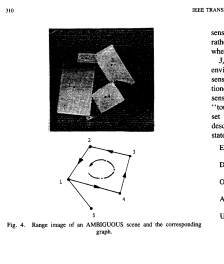
3 Resolução

3.1 Encontrar contornos na imagem

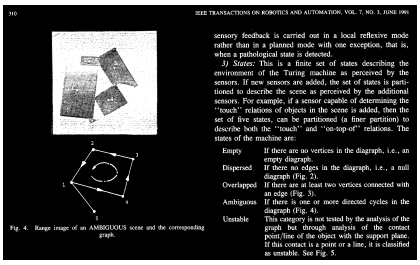
Para encontrar componentes conexos, dada uma imagem de entrada, inicialmente invertemos seus *pixels*, isto é, por meio da função **bitwise_not** do OpenCV invertemos os pixels da imagem de branco para preto e de preto para branco, logo a imagem passou a possuir fundo preto e letras brancas. A Figura 1 ilustra a imagem original e a mesma imagem após a inversão de pixels.

Figura 1: Imagem original e sua inversa

(a) Imagem original



(b) Imagem original invertida



Após este passo, realizamos operações de dilatação e erosão na imagem invertida, utilizando as funções **dilate** e **erode** do OpenCV.

Uma operação de dilatação consiste em escanear um kernel sobre uma imagem, calculando o valor máximo do pixel sobreposto pelo kernel e substituindo o pixel da imagem na posição central do kernel com esse máximo valor. Essa operação de maximização faz com que regiões claras dentro de uma imagem, "cresçam". Por isso é dado o nome de "dilatação".

A operação de erosão é similar a dilatação, porém ela calcula o mínimo local sobre a área do kernel e substitui o pixel da imagem na posição central do kernel por este valor mínimo. Essa operação faz com que regiões mais claras diminuam, e o fundo se sobressaia.

Primeiramente, aplicamos na imagem uma operação de dilatação, com um elemento estruturante correspondente ao **kernel_1**, que foi declarado anterior-

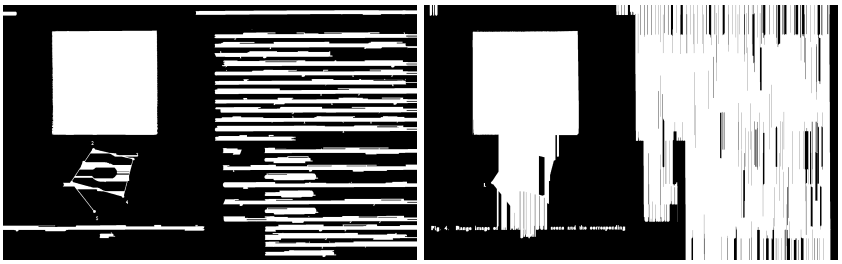
mente. Logo após, aplicamos no resultado do processo anterior uma operação de erosão com o mesmo elemento estruturante.

Em seguida, repetimos estes mesmos passos, alterando apenas o elemento estruturante. Assim, aplicamos na imagem uma operação de dilatação, com um elemento estruturante correspondente ao **kernel_2** e logo após, aplicamos no resultado obtido uma operação de erosão com o mesmo elemento estruturante.

A Figura 2 ilustra as imagens que foram obtidas após estes processos. É possível notar o efeito que as dimensões do kernel possuem sobre os resultado, na Figura 2a ocorreu uma junção de componentes na horizontal (linhas), enquanto na Figura 2b essa junção de componentes é feita na vertical.

Figura 2: Aplicação da operação de fechamento na imagem

(a) Kernel_1 (b) kernel_2

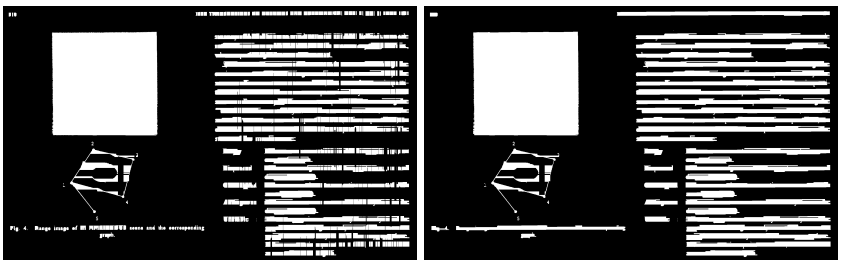


Em seguida, através da função **bitwise_and**, realizamos uma interseção entre as duas imagens geradas nos processos anteriores. E com o resultado desta interseção realizamos uma operação de fechamento com o elemento estruturante definido pelo **kernel_3**. Uma operação de fechamento é definida como uma dilatação seguida de uma erosão.

As imagens geradas após a interseção e a operação de fechamento podem ser visualizadas na figura 3. Após a interseção é possível visualizar na imagem algumas pequenas brechas entre os contornos e após a operação de fechamento estas brechas foram preenchidas.

Figura 3: Aplicação da interseção e fechamento na imagem

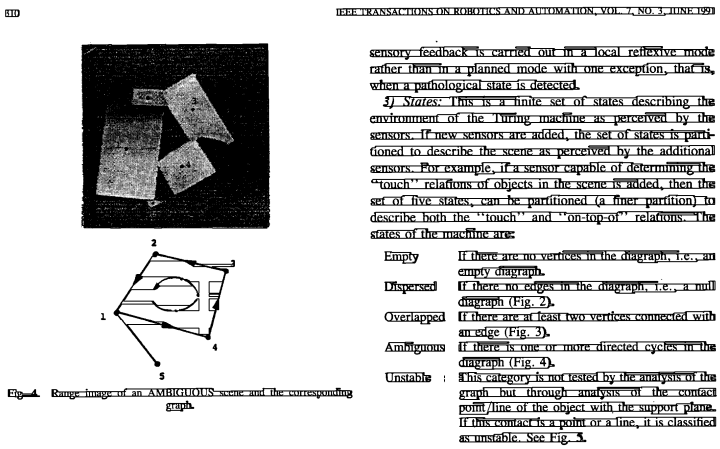
(a) Interseção (b) Fechamento



Após estes processos, utilizamos a função **findContours** do OpenCV, que encontra contornos em uma imagem binária utilizando o algoritmo "Topological structural analysis of digitized binary images by border following" [4]. Utilizamos esta função passando como parâmetro a imagem gerada após a operação de fechamento da figura 3, a função **findContours** retorna uma lista contendo as coordenadas de cada contorno encontrado na imagem.

Após obter as coordenadas podemos desenhar cada contorno na imagem original para obtermos uma visualização dos mesmos. Os contornos desenhados na imagem original podem ser visualizados na figura 4.

Figura 4: Aplicação da interseção e fechamento na imagem



Uma vez que temos os contornos de cada componente, foi necessário estabelecer uma regra para definir se um determinado contorno se caracteriza como texto ou não, para isso dois coeficientes foram considerados, chamados de coeficientes A e B, onde:

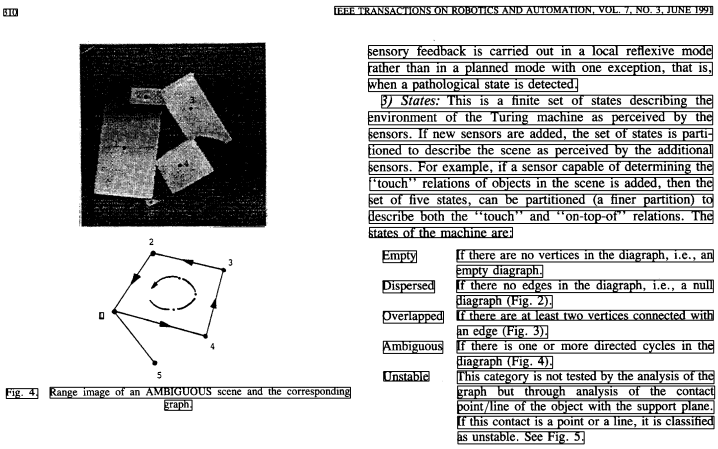
- A - é razão entre o número de pixels pretos e o número total de pixels (altura x largura);
- B - é razão entre o número de transições verticais e horizontais branco para preto e o número total de pixels pretos;

Para cada contorno na imagem, computamos e salvamos seus coeficientes A e B. Ao final, computamos a média e o desvio padrão de cada coeficiente. A regra de texto então foi definida como sendo:

$$|media_A - 2 * media_B| <= coeficiente_A <= media_A \quad (1)$$

A Figura 5 mostra o resultado obtido após desenhar na imagem apenas os contornos que respeitam a regra definida acima. Esta regra também foi utilizada para determinar a quantidade de linhas na imagem. Sendo assim, ao desenhar cada contorno, assumimos o mesmo representando também uma linha, então incrementamos um contador de linhas.

Figura 5: Aplicação da interseção e fechamento na imagem



3.2 Retângulo envolvendo cada palavra

Para a desenhar um retângulo em cada palavra e consequentemente contá-las, segui-se alguns passos como os utilizados na seção anterior.

Novamente, recebemos uma imagem de entrada e invertemos seus pixels. Após isto, utilizamos um elemento estruturante representado pelo **kernel_4**, para realizar três iterações de dilatação na imagem. Em seguida, no resultado que foi obtido, realizamos três iterações de erosão utilizando este mesmo **kernel_4**.

Logo após estes passos, assim como na seção anterior, utilizamos a função **findContours** na imagem obtida para encontrar os componentes presentes na mesma. O resultado obtido após as iterações de dilatação e erosão e os contornos obtidos desenhados na imagem, podem ser visualizados na figura 6. Notamos que as dimensões menores do **kernel_4** possibilitaram, após as dilatações e erosões, destacar melhor componentes mais próximos, que seriam as palavras, sem unificá-los.

Após estas etapas, foi necessário a criação de uma outra regra para definir o que é ou não palavra. Assim como a regra de texto definida anteriormente, esta regra também é baseada nos coeficientes A e B citados na seção anterior. Para cada contorno na imagem, calculamos novamente os coeficientes e baseado nos valores de média e desvio padrão definimos esta regra como:

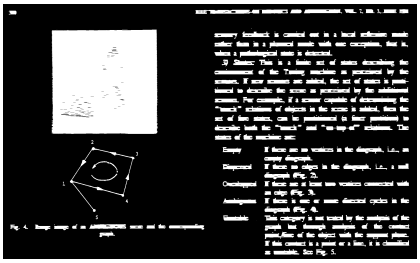
$$|\frac{1}{2} * media_A - desvio_A| <= coeficiente_A <= \frac{1}{2} * media_A + (1.7) * desvio_A \quad (2)$$

$$coeficiente_B >= media_B \quad (3)$$

A Figura 7 mostra o resultado obtido após desenhar na imagem apenas os contornos que respeitam a regra de palavra definida acima. Para cada contorno que satisfaz a regra definida, incrementamos um contador de palavras, assim ao final obtemos uma quantidade aproximada de palavras presentes na imagem.

Figura 6: Resultado após as iterações de dilatação e erosão e desenho dos contornos

(a) Iterações de dilatação e erosão



(b) Contornos na imagem

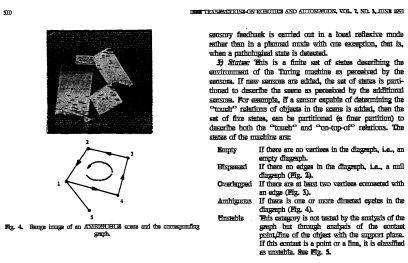
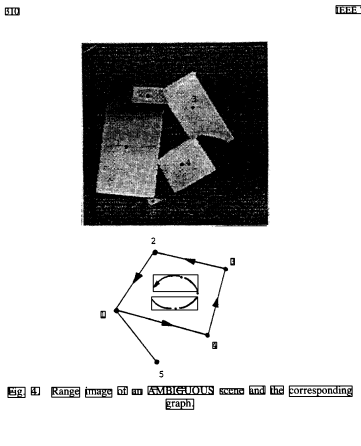


Figura 7: Resultado após desenhar contornos de palavras



4 Resultados

Ao final de todos os processos, obtivemos um retângulo envolvente em cada palavra mostrado na figura 7, o que consideramos um resultado satisfatório pois, apesar de pequenas falhas, os contornos em sua maioria abrangeram de maneira fiel as palavras. Além disso, para esta mesma imagem obtivemos um valor aproximado de linhas de texto e blocos de palavras. Estes valores estão presentes na tabela 1.

Tabela 1: Quantidades de linhas e palavras no texto

Quantidade de linhas	Quantidade de palavras
36	301

Referências

[1] Matplotlib user guide. Accesso em: 05/05/2019.

[2] Numpy user guide. Accesso em: 05/05/2019.

[3] Welcome to opencv documentation! Accesso em: 05/05/2019.

[4] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.