

# Trabalho 4

## MO443: Operadores Morfológicos

Henrique de Abreu Amitay - RA: 138501

## 1 Introdução

Este trabalho tem o objetivo de explorar o efeito de operadores morfológicos em imagens binárias e da aplicação de algoritmos de identificação de componentes conexos para segmentar regiões entre *texto* e *não texto*.

Junto deste relatório se encontram os scripts para cada etapa do trabalho junto à uma imagem de exemplo.

O trabalho foi implementado utilizando a linguagem de programação **Python** na versão **3.6.9**.

Além disso os seguintes pacotes foram utilizados para a implementação:

- OpenCv (cv2)
- Numpy

Todas as etapas deste exercício foram feitos a partir da seguinte imagem em binária:

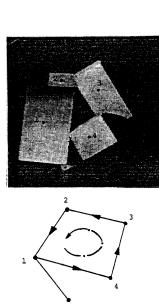


Fig. 4. Range image of an AMBIGUOUS scene and the corresponding graph.

Figura 1: Imagem binária utilizada

## 2 Exercícios

### 2.1 Operações Morfológicos e Algoritmo de componentes conexos

Para este trabalho as seguintes operações foram feitas na imagem base:

- 1) Inversão da imagem para pixels escuros se tornarem brancos, de forma a operar em cima dos textos.
- 2) Dilatação da imagem negativa com um elemento estruturante, *kernel*, de 1 pixel de altura e 100 pixels de largura.

- 3) Erosão da imagem dilatada acima com um elemento estruturante, *kernel*, de 1 pixel de altura e 100 pixels de largura.
- 4) Dilatação da imagem negativa com um elemento estruturante, *kernel*, de 200 pixels de altura e 1 pixel de largura.
- 5) Erosão da imagem dilatada acima com um elemento estruturante, *kernel*, de 200 pixels de altura e 1 pixel de largura.
- 6) Aplicação da intersecção (AND) dos resultados dos passos 3 e 5.
- 7) Fechamento do resultado obtido no passo (6) com um elemento estruturante de 1 pixel de altura e 30 pixels de largura.

O primeiro passo se deu necessário pois os elementos estruturantes utilizados pelos métodos fornecidos pelo OpenCV esperam uma imagem binária cujo preto seja um valor unitário e branco valores nulos. Como a imagem inicial era o oposto deste cenário, a imagem foi invertida a partir da seguinte operação:

```
# Negativa a imagem
img = (255 - img)
```

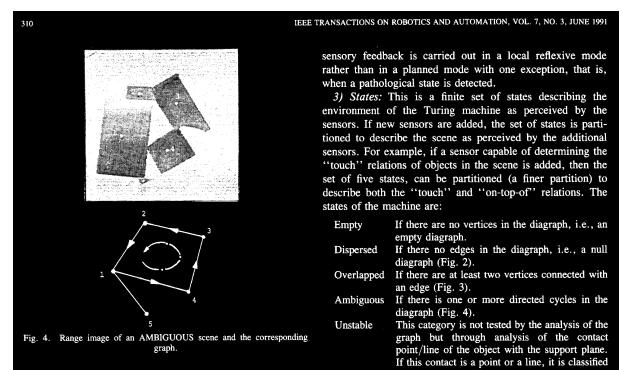


Figura 2: Imagem binária negativada

Os elementos estruturantes retangulares também são facilmente obtidos a partir de um vetor do Numpy com elementos unitários, conforme o seguinte método desenvolvido:

```

1 def createKernel(height, width):
2     return np.ones((height, width))
3

```

As operações de Dilatação e Erosão, que correspondem respectivamente ao Somatório e Subtração de Minkowski foram trivialmente aplicadas a partir dos métodos `dilate` e `erode` do OpenCV, ambos os métodos recebem a imagem e o elemento estruturante e retornam a imagem após a aplicação do operador.

```

1 def dilate(img, kernel):
2     return cv2.dilate(img, kernel, iterations
3 =1)
4
5 def erode(img, kernel):
6     return cv2.erode(img, kernel, iterations
7 =1)
8

```



Figura 3: Resultante da Primeira Dilatação

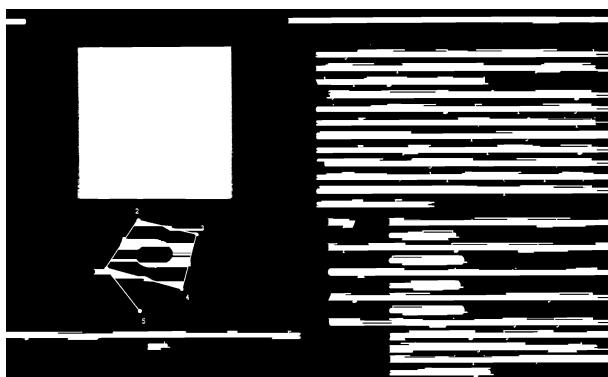


Figura 4: Resultante da Primeira Erosão



Figura 5: Resultante da Segunda Dilatação

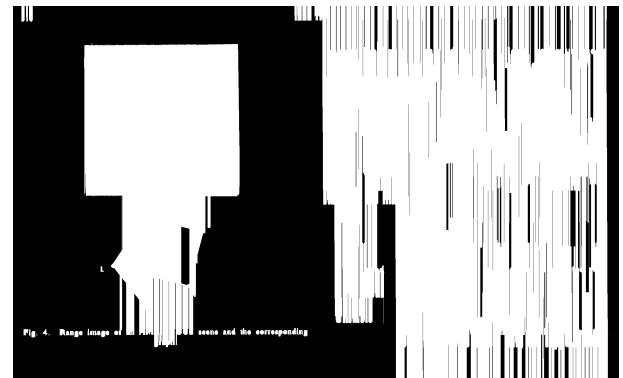


Figura 6: Resultante da Segunda Erosão

A operação AND é facilmente feita a partir do operador lógico `&`, esta operação que também é a intersecção entre os resultados nos fornece o resultado abaixo.

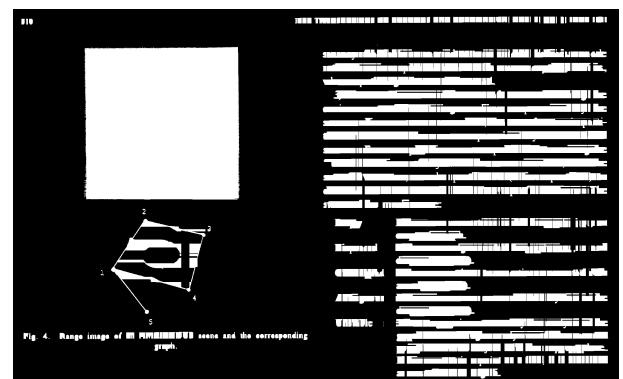


Figura 7: Resultante da Intersecção (Operação AND)

Por fim operação de fechamento, pode ser desenvolvida a partir da operação de dilatação seguida da erosão, porém para explorar mais as ferramentas do OpenCV utilizou-se o método `morphologyEx` que permite utilizar várias operações morfológicas.

```

1 def close(img, kernel):
2     return cv2.morphologyEx(img, cv2.
3     MORPH_CLOSE, kernel)

```

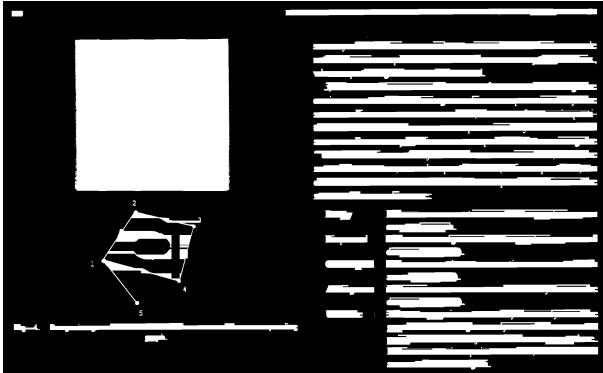


Figura 8: Resultante da Operação de Fechamento

Após estas operações morfológicas o algoritmo dos componentes conexos foi aplicado também a partir de métodos oferecidos pelo OpenCV, conforme código abaixo:

```

1 def connected_component(img):
2     return cv2.connectedComponents(img)
3

```

Este método preenche cada componente conexo com um valor diferente e retorna além da imagem processada um vetor com estes valores, que podem ser abstraídos como labels para cada componente conexo, com isso, podemos isolá-la cada componente e executar os seguintes passos:



Figura 9: Separação dos Componentes Conexos em Labels

- 8) Aplicação de algoritmo para identificação de componentes conexos sobre o resultado do passo 7.
- 9) Para cada retângulo envolvendo um objeto foi calculado: a razão entre o número de pixels brancos (dado que a imagem foi negativada) e o número total

de pixels, e a razão entre o número de transições verticais e horizontais branco para preto e número total de pixels pretos.

- 9) Com estes valores foi achada uma regra para identificar retângulos *texto* e *não texto*

Os retângulos foram obtidos a partir da descoberta dos contornos dos componentes conexos com o seguinte método oferecido pelo OpenCV.

```

1 countours = cv2.findCountours(mask, cv2.
2     RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[1]
3 (x, y, w, h) = cv2.boundingRect(countours[0])
4 rectangle = img[y:y+h, x:x+w]

```

Estes foram os resultados obtidos:

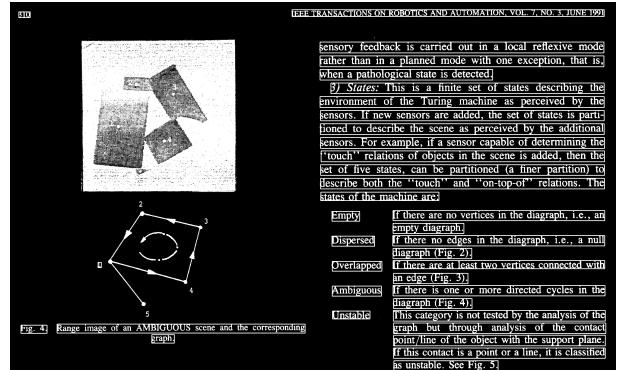


Figura 10: Descoberta dos Retângulos de Texto

Por fim, utilizamos mais duas operações morfológicas em cada um dos retângulos encontrados de forma a conseguir identificar as diferentes palavras dentro da imagem.

Podemos assumir que letras suficientemente perto uma das outras constitui uma palavra, logo uma dilatação com um elemento estruturante retangular de 10x10 seguido de um fechamento com um elemento estruturante de 2x2 serviu para agrupar cada palavra em elementos conexos, conforme exemplo abaixo:

**as unstable. See Fig. 5.**

Figura 11: Exemplo de texto não processado



Figura 12: Exemplo de texto apóis dilatação e fechamento

Com isto, o algoritmo de componentes conexos é executado em cada um destes retângulos cujas palavras foram

"aglutinadas" e com isso conseguimos as coordenadas de todos os retângulos que constituem uma palavra conforme resultado abaixo:

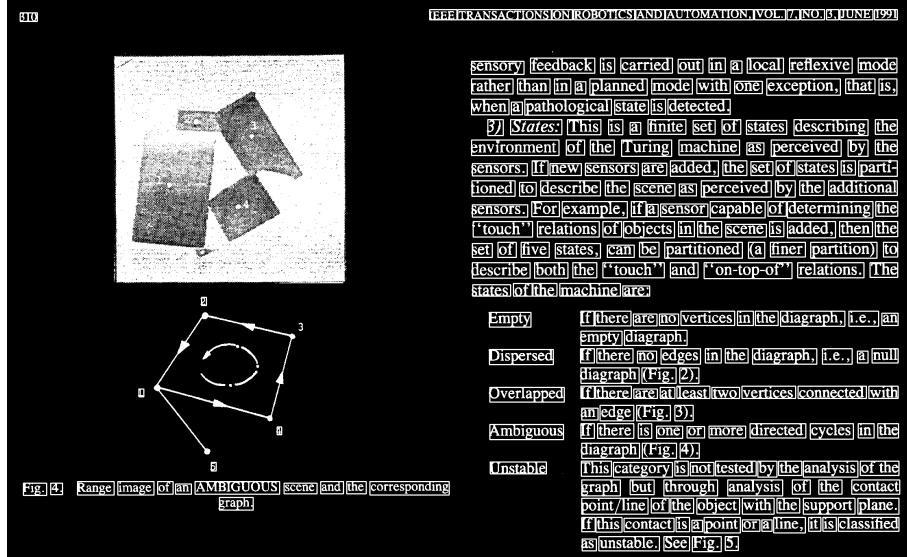


Figura 13: Resultado Final de contagem de palavras

## 2.2 Conclusão

O experimento acima ilustra como operadores morfológicos podem ser implementados de maneira simples de forma a fornecer instrumentos poderosos na detecção de padrões e elementos de uma imagem. Um simples entendimento de como dividir e conquistar os elementos de uma imagem binária aliados às operações forneceram um resultado simples e relativamente performático.