Relatório do trabalho 3

Allan Nozomu Fukasawa RA:163527

22/05/2019

1 Introdução

O objetivo deste trabalho é aplicar operadores morfológicos para segmentar regiões compreeendendo *texto* e *não texto* em uma imagem, utilizando algumas métricas e razões para sua classificação. Além disso, utilziando operadores morfológicos, segmentar cada componente compreeendendo *texto* em palavras.

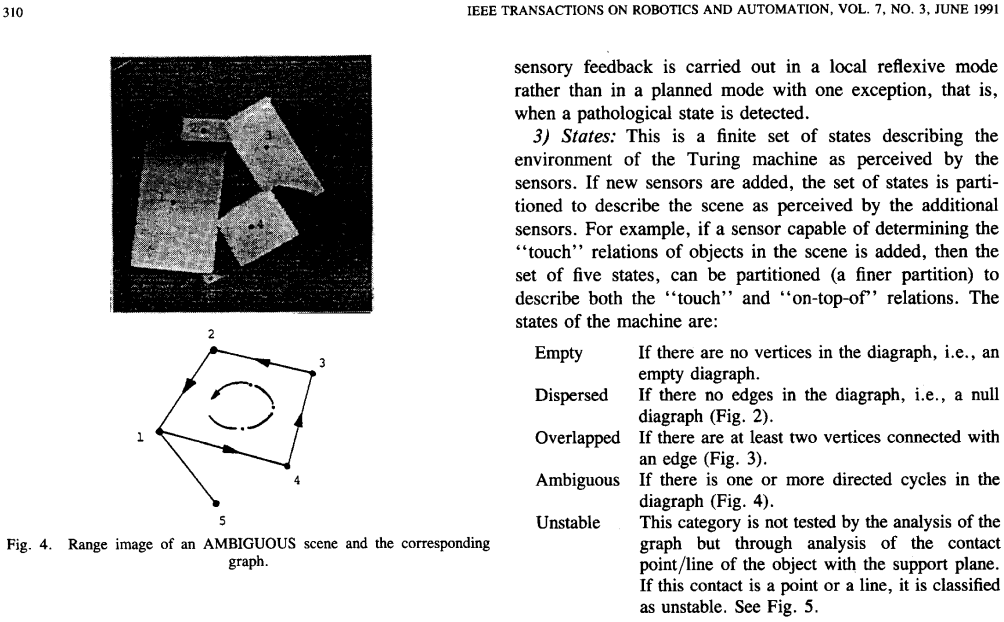
Os passos seguidos para a execução do trabalho foram extraídos de sua especificação. [1]

2 Componentes

Está sendo enviado junto a este relatório os seguintes arquivos e diretórios:

- arquivo Trabalho 3.ipynb: contém todo o código executado durante este trabalho.
- três imagens utilizadas durante o processamento: uma (*bitmap.pbm*)disponibilizada pelo professor e que foi usada durante o desenvolvimento e as outras duas (*bitmap2.pbm* e *bitmap3.pbm*) para se fazer testes e verificar se os resultados estão condizentes.
- diretório res: dentro do diretório *res* contém mais diretórios com os nomes das imagens utilizadas durante o desenvolvimento, sendo um para cada imagem. Dentro de cada diretório da imagem existem todos os componentes conexos detectados, imagens obtidas durante o processamento e a classificação em dois diretórios (*texts* e *not_texts*) dos componentes.
- imagens auxiliares em formato .png: para compor o relatório e mostrar os resultados obtidos

A imagem padrão que está sendo utilizada no programa e também nos resultados apresentados neste relatório é a imagem 1



310 IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 7, NO. 3, JUNE 1991

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

sensory feedback is carried out in a local reflexive mode rather than in a planned mode with one exception, that is, when a pathological state is detected.

3) States: This is a finite set of states describing the environment of the Turing machine as perceived by the sensors. If new sensors are added, the set of states is partitioned to describe the scene as perceived by the additional sensors. For example, if a sensor capable of determining the "touch" relations of objects in the scene is added, then the set of five states, can be partitioned (a finer partition) to describe both the "touch" and "on-top-of" relations. The states of the machine are:

Empty	If there are no vertices in the diagraph, i.e., an empty diagraph.
Dispersed	If there no edges in the diagraph, i.e., a null diagraph (Fig. 2).
Overlapped	If there are at least two vertices connected with an edge (Fig. 3).
Ambiguous	If there is one or more directed cycles in the diagraph (Fig. 4).
Unstable	This category is not tested by the analysis of the graph but through analysis of the contact point/line of the object with the support plane. If this contact is a point or a line, it is classified as unstable. See Fig. 5.

Figura 1: Imagem original

2.1 O Programa

O programa foi implementado com Jupyter Notebooks, usando Python 3.7.1. As bibliotecas utilizadas no desenvolvimento do programa foram, com suas respectivas versões:

- numpy (1.15.4): para manipulação dos vetores.
- matplotlib (3.0.2): visualização dos dados, resultados finais e intermediários.
- opencv (3.4.2): realização da leitura e escrita das imagens, transformação das cores e normalização dos dados. Em especial, deve ser utilizada a versão 3.4.2 pois as anteriores não permitem salvar em formato pbm.
- scipy (1.1.0): principalmente o pacote **ndimage** para realizar as diversas operações morfológicas.

2.2 Formato das imagens

As imagens de entrada estão no formato .pbm. As imagens de saída, tanto as finais quanto as intermediárias também se encontram neste mesmo formato.

3 Leitura, escrita e plotagem das imagens

3.1 Leitura das imagens

A imagem de entrada é lida com função **cv2.imread** que armazena a imagem em um **numpy.ndarray** de 3 dimensões (MxNx3).

Depois de lida, a imagem é convertida para níveis de cinza pela função **cv2.cvtColor**. Após convertido para escala de cinza, a imagem foi revertida, ou seja, os pixels que estavam descritos como 0 viram 1 e os que estavam como 1 viram 0. Isso se deve ao fato de que, o *openCV* tem como padrão, o branco como sendo 255 e o preto sendo 0. Porém para se aplicar as funções morfológicas, é necessário que o branco (fundo) seja descrito como 0 e o preto, como sendo 1.

3.2 Escrita das imagens

Também foi feito uma função auxiliar para facilitar na saída das imagens utilizando a função **cv2.imwrite**. Porém, assim como foi invertido a imagem quando lida, o mesmo deve ser feito quando escrita depois de aplicar todas as operações morfológicas para o *openCV* conseguir distinguir entre preto e branco novamente.

3.3 Plotagem das imagens

Foi utilizado para visualização dos resultados as funções de plotagem de imagens em **matplotlib.pyplot**, utilizando a paleta de cores em escalas de cinza. Assim como a leitura e escrita, também foi necessário fazer a inversão da imagem antes de plotá-la, uma vez que o *matplotlib* também funciona com o mesmo padrão utilizado pelo *openCV*, onde o 0 representa o preto.

4 Solução

4.1 Componentes conexas

Para a detecção das componentes conexas na imagem, foram feitas operações morfológicas de dilatação seguida de erosão usando dois diferentes elementos estruturantes: Um contendo 1 pixel de altura e 100 pixels de largura e outro contendo 200 pixels de altura e 1 pixel de largura. As operações foram feitas em duas cópias das imagens originais, de tal forma que uma operação não afetou no desempenho da outra. As seguintes funções foram usadas: **morphology.binary_dilatation** e **morphology.binary_erosion**

Cada um desses elementos estruturantes foi responsável por agrupar elementos na vertical (no caso do elemento estruturante com 100 pixels de altura) e na horizontal (para o outro elemento estruturante)

Depois de feito as operações nas duas cópias, uma nova imagem foi formada a partir da intersecção dos seus resultados. Foi aplicado em cima desse resultado, a operação de fechamento em um elemento estruturante de 1 de altura e 30 de largura para melhorar as delimitações de cada componente conexa. Foi utilizado a seguinte função (**morphology.binary_closing**)

Após a operação de fechamento, foi feito uma função auxiliar para identificar cada componente conexa e calcular um retângulo que envolve cada uma delas. Durante a execução dessa função, é verificado se dentro

do retângulo definido há de fato algum pixel preto na imagem original ou se a componente conexa foi gerada por ruídos gerados durante as operações morfológicas anteriores. Com isso, foi possível também modificar a imagem colocando um retângulo de 1 pixel de largura envolvendo cada componente conexa descoberta, como demostrado na Figura 2. Foram obtidas, ao todo, 53 copmponentes conexas.

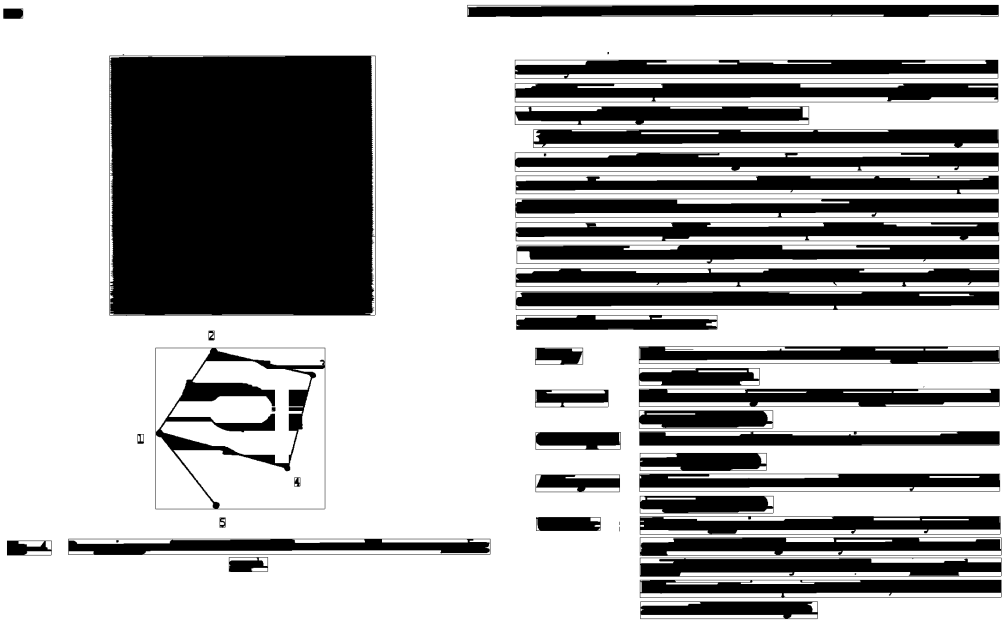


Figura 2: Resultado das operações morfológicas e retângulos definindo cada uma das 53 componentes conexas

É possível observar que as operações morfológicas acabaram desconsiderando algumas partes das letras como *pingos* nos *is*. Isto se deve porque é feito uma intersecção entre as operações realizadas nas verticais e horizontais. Apesar desses *pingos* estarem presentes utilizando as operações morfológicas com elementos estruturantes de maior altura, alguns desses pingos acabam não sendo capturados pelos elementos estruturantes de maior largura. Dessa maneira, parte desses *pingos* foram considerados como sendo diferentes componentes conexos.

4.2 Classificação *texto* e *não texto*

O próximo passo após a detecção das componentes conexas foi suas classificações entre *texto* e *não texto*. Para realizar tal feito, para cada componente conexa descoberta pelo passo anterior, foi realizado o cálculo de duas razões:

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

Para cada uma dessas razões, foi criado uma regra para se classificar os textos entre *texto* e *não texto*. Somente foram consideradas as componentes conexas que possuíam, ao menos, 50 pixels de largura pois dessa forma, evitou-se legendas, numerações de páginas e ruídos. A decisão da proporção usada foi feita apenas por observação dos resultados.

Seja b a razão dos pixels pretos e i a razão das inversões chegou-se na seguinte regra:

$$X(b,i) = \left\{ \begin{array}{ll} \text{texto}\ , & \text{se } 0.15 \leq b \leq 0.60 \text{ e } i > 0.55 \\ \text{não texto}\ , & \text{caso contrário} \end{array} \right\}$$

Seguindo esta regra, foi detectado um total de 34 linhas contendo textos. Pode-se verificar os resultados na Figura 3, onde há retângulos envolvendo cada uma das linhas detectadas.

4.3 Separação das palavras

Depois de classificado cada uma das componentes conexas em *texto* e *não texto*, deseja-se separar cada palavra de cada linha classificada. Para isso, foi utilizado uma abordagem bem similar a realizada no começo da solução, para detecção das componentes conexas, utilizando operadores morfológicos de fechamento com elementros

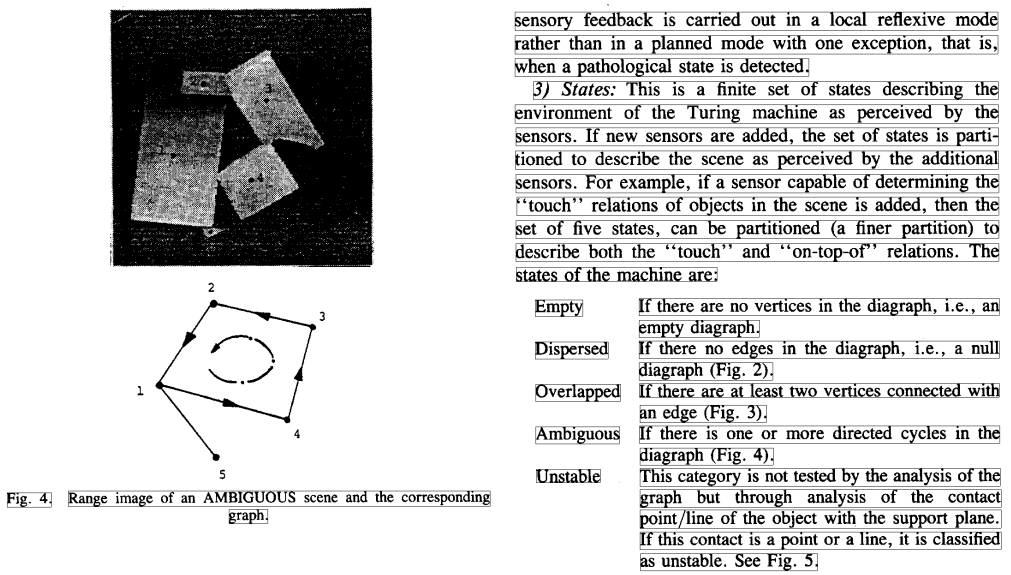


Figura 3: Resultado das classificação dos textos e retângulos definindo cada uma das 34 linhas classificadas

estruturantes de um certo tamanho para agrupar cada letra de cada palavra. Depois, utilizar a mesma função para obter cada componente conexa (palavra) junto com o retângulo que define cada uma delas.

Para o tamanho do elemento estruturante, foi feito o seguinte cálculo para cada linha descrita na imagem. Pegou-se a altura da imagem e dividiu este valor por 3.5, obtendo um certo fato. Com este fator, foi contruído um elemnto estruturante de altura $min(30, 3 \cdot fator)$ e largura $min(30, fator)$. Calculando as dimensões, o algoritmo torna-se mais escalável diante de fontes de tamanhos diferentes onde o espaçamento entre palavras também é diferente.

O tamanho máximo de 30 pixels, foi necessário pois operações de fechamento com elementos estruturantes muito grandes são custosas e causam lentidões durante o processamento. Além disso, a opção por se ter uma altura maior que a largura é justamente para obter *pingos nos is*, pontuações (vírgulas, pontos finais) que antes não eram consideradas.

Aplicando estas operações morfológicasogicas, foi detectado um total de 235 palavras na imagem inteira. Pode-se verificar os resultados na Figura 4, onde há retângulos envolvendo cada uma das palavras detectadas.

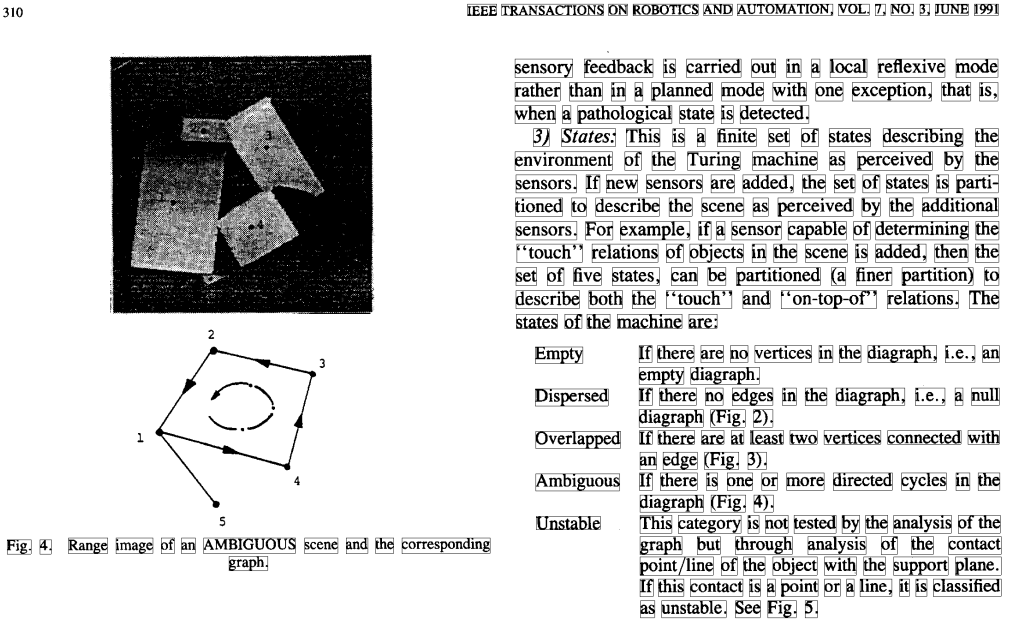


Figura 4: Resultado das classificação dos textos e retângulos definindo cada uma das 34 linhas classificadas

5 Conclusão

Após a execução de todos os procedimentos, pode-se concluir que operadores morfológicos são recursos interessantes para obter componentes conexas para utilizar, por exemplo como feito neste trabalho, classificação de textos.

Porém, há algumas limitações presentes no algoritmo utilizado para a separação das componentes conexas, principalmente quando há elementos que não são textos ao lado de um parágrafo. Tal elemento que não é texto acaba sendo detectado como sendo parte da mesma componente conexa do parágrafo, o que pode gerar resultados questionáveis.

Com a imagem utilizada por padrão neste relatório não foi possível obter este problema. Porém utilizando a imagem *bitmap3.pbm* em anexo juno a este relatório, foi possível reproduzí-lo. Observa-se que a imagem e o texto na imagem 5 resultou em uma única componente conexa bem grande pois uniu-se os textos com a imagem ao lado 6. As linhas abaixo foram detectadas separadamente, formando componentes conexas distintas.

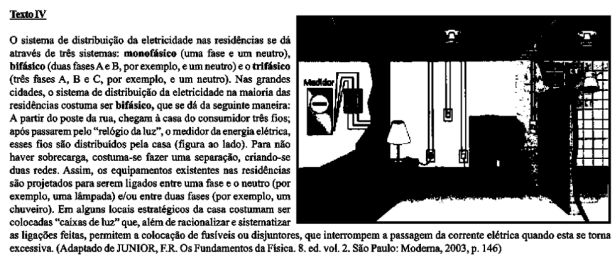


Figura 5: Texto com imagem ao lado

Figura 6: Resultado das componentes conexas após oprações morfológicas

Além disso, as operações morfológicas junto aos elementos estruturantes escolhidos durante este trabalho se tornaram bem sensíveis a ruídos. Uma possível melhoria do algoritmo seria aplicar filtros para redução de ruídos e elementos indesejáveis antes de se aplicar as operações morfológicas. Porém essas técnincas não foram aplicadas durante o desenvolvimento deste experimento, sendo apenas sugestões para melhorias futuras.

Referências

[1] H. Pedrini, *Trabalho 3.* Introdução ao Processamento de Imagens (MC920 / MO443), 2019.