Trabalho 3 - MC920 - Operações Morfológicas

Luiz Eduardo Cartolano - 183012

Maio de 2019

Resumo

As transformações morfológicas são algumas operações simples baseadas no formato da imagem que normalmente são executadas em imagens binárias, neste trabalho o objetivo da aplicação das operações foi segmentar textos e palavras em uma dada imagem binária. Os resultados obtidos, disponibilizados nas Figuras 7 e 8, foram bastante positivos, validando a utilização das técnicas para o objetivo proposto.

1 Introdução

O objetivo deste trabalho, como solicitado em 1, é aplicação de operadores morfológicos para segmentar regiões compreendendo texto e não texto em uma dada imagem de entrada. Para tal, foram usadas operações bem conhecidas, como a dilatação, erosão, entre outras, como será abordado com mais detalhes nas próximas seções.

Transformações morfológicas são algumas operações simples baseadas no formato da imagem. Normalmente são executadas em imagens binárias e seu uso depende de duas imagens de entrada, uma é a imagem original, a segunda é chamada de elemento estruturante, que decide a natureza da operação. Uma das aplicações práticas dessas operações é a explorada nesse relatório, no qual iremos segmentar o conteúdo de uma imagem em texto e não texto, e depois segmentar as palavras presentes em cada linha identificada como texto.

Este relatório encontra-se organizado da seguinte forma: a Seção 2 apresenta a maneira como o trabalho foi implementado a fim de solucionar o problema do enunciado. A Seção 3 apresenta os resultados obtidos pela implementação feita para determinadas imagens de entrada. As discussões a respeito dos resultados encontrados estão dispostas na Seção 4, e as conclusões na Seção 5. As imagens, tabelas e equações apresentadas encontram-se nos Anexos ao final do relatório.

2 Implementação

2.1 Uso do Código

O arquivo com os códigos implementados está nomeado como *proj3.ipynb*, e é um *notebook* do *Jupyter* [1]. Desse modo, para executá-lo, é preciso possuir o *Jupyter*, além da versão 3.7 do *Python* e as bibliotecas que foram usadas no desenvolvimento, sendo elas: *Opencv*, *Matplotlib* e *Numpy*. Além disso, é preciso possuir uma pasta *inputs*, com os arquivos de entrada, e uma *outputs*, na qual serão salvas as imagens geradas pelo código.

Para executar o código basta executar, célula a célula, o *notebook* fornecido. Vale lembrar que é essencial que as células sejam executas na ordem em que aparecem.

2.2 Detalhes da Implementação

Nesta seção serão comentadas cada uma das operações morfológicas que foram aplicadas, explicando cada uma delas com mais detalhes. As funções do *OpenCV* que foram usadas podem ser vistas com maior detalhes em 2. Um detalhe interessante é que o *OpenCV* considera os pixels brancos como sendo

[1] https://jupyter-notebook.readthedocs.io/en/stable/

"1"e os pretos como "0". Uma matriz inversa a essa foi criada para realizar os passos solicitados na Seção 2.2.6. Além disso, usou-se o negativo da imagem lida para os demais passos.

2.2.1 Dilatação

Na dilatação, um elemento de pixel é "1"se pelo menos um pixel "abaixo"do kernel for "1". Por isso, ela aumenta a região branca na imagem ou, em outras palavras, o tamanho do objeto em primeiro plano aumenta.

Para realizar a implementação da dilatação contou-se com a função *cv2.dilate(img,kernel,iterations=1)*, na qual o parâmetro *img* representa a imagem a ser dilatada e o *kernel* o elemento estruturante que será usado na técnica.

A criação do elemento estruturante foi feita com auxílio da biblioteca *Numpy*, criando-o a partir da função *np.ones((altura,largura))*.

2.2.2 Erosão

A idéia básica da erosão é eliminar os limites do objeto em primeiro plano. Então, o *kernel* desliza pela imagem (como na convolução 2D). Um pixel na imagem original (1 ou 0) será considerado 1 somente se todos os pixels sob o kernel forem 1, caso contrário, ele será erodido (transformado em zero).

Assim como na dilatação, a técnica da erosão também está implementada na biblioteca do *Opencv*, de modo que sua utilização é feita por chamadas a função *cv2.erode(img,kernel,iterations=1)*. Assim como na função responsável por exercer a dilatação, o parâmetro *img* representa a imagem a ser erodida e o parâmetro *kernel* o elemento estruturante.

2.2.3 Fechamento (ou *Closing*)

O fechamento é uma dilatação seguida de uma erosão. É útil para fechar pequenos furos dentro dos objetos de primeiro plano, ou pequenos pontos pretos no objeto.

A operação de fechamento é mais uma das transformações morfológicas abrangidas pela biblioteca do *Opencv*. Sua utilização é feita através da chamada de função *cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)*. É interessante notar que, diferente das funções vistas anteriormente, a função usada para realizar o fechamento possui um terceiro parâmetro, o *cv2.MORPH_CLOSE*, ele é o responsável por indicar para a função que a operação que será realiza será um fechamento, isso é necessário pois a função responsável pelo método de abertura possui a mesma assinatura. Os demais parâmetros são iguais aos vistos para a dilatação e erosão.

2.2.4 Intersecção *AND*

A intersecção *AND* é feita através da aplicação da operação lógica de *AND* entre duas imagens, desse modo, o pixel será "1"caso ele possua valor "1"em ambas as matrizes para uma dada posição da imagem, caso contrário será "0". A realização de tal operação em *python* pode ser feita com a seguinte expressão: *img_1 & img_2*.

É interessante ressaltar que para que o comando apresentado funcione como o esperado é preciso que ambas as matrizes sejam compostas por números inteiros "0"ou "1". Para tal, foi preciso, na implementação feita primeiro usar a função *cv2.normalize(img, None, alpha=0, beta=1,...)* e em seguida fazer uso do método *.astype()* da biblioteca *Numpy*, para então aplicar o operador lógico.

2.2.5 Componentes Conexos

Os Componentes Conexos são grupos de pixels "1"isolados uns dos outros, eles tem este nome pois tomado um ponto dentro da componente conexa, é possível andar para qualquer outro ponto da componente passando somente por pontos que também estão nela.

Para descobrir quais eram os componentes conexos da imagem no trabalho, fez-se uso da função *cv2.connectedComponentsWithStats(img)*, seu parâmetro é uma imagem, cuja matriz deve ser composta por números inteiros no intervalo de [0,255]. A função por sua vez possui quatro retornos, sendo que dois deles foram essenciais para o restante do trabalho. Foram eles, o primeiro retorno, que informa o número

de componentes conexas encontradas na imagem, e o terceiro retorno, que é uma lista na qual qual cada elemento é um vetor com as estatísticas da *bounding box* que demarca o componente conexo.

Para demarcar os componentes na imagem original percorreu-se cada um dos elementos da lista de estatísticas e usou-se a função *cv2.rectangle(img,(x_ i,y_ i),(x_f,y_f),(0, 0, 0), 2)*. Ela é responsável por desenhar o contorno de um retângulo a partir das posições de dois pontos extremos (o mais alto a esquerda e o mais baixo a direita).

2.2.6 Razão de elementos pretos e de transições

Para poder realizar a separação de quais elementos conexos são considerados texto e quais não são, é preciso realizar algumas operações matemáticas intermediárias, a fim de elencar estatísticas que serão usados como parâmetros da identificação posterior. Ambas as estatística estudadas foram obtidas para cada um dos componentes conexos obtidos como explicado anteriormente, portanto, nessa etapa, percou-se cada um dos componentes e aplicou-se as técnicas que serão descritas.

A primeira estatística implementada foi a razão entre o número de pixels pretos e o número total de pixels, para obter tal fator, somou-se os elementos da matriz da componente conexa e dividiu-se o valor obtido pelo valor obtido pela multiplicação $altura \cdot largura$ da componente.

A segunda estatística obtida foi descobrir a razão entre o número de transições branco para preto e o número de pixels pretos. Para o obter o número de transições percorreu-se os elementos da matriz delimitada pela componente conexa e contou-se todas as vezes nas quais o pixel [i,j] era "0"e o pixel [i+1,j] ou [i,j+1] era "1", logo, na abordagem escolhida, ignorou-se o caso de transições preto para branco. Por fim, somava-se o elementos da matriz para encontrar o número de pixels pretos, e então realizava-se a divisão de ambos os valores.

Nessa etapa da implementação obteve-se duas listas distintas de razões e de tamanho igual a lista de estatísticas dos componente conexos.

2.2.7 Identificação de textos

Uma vez obtido as listas de razões e a lista com os componentes conexos, percorreu-se cada componente conexo e avaliou-se suas razões. Caso o componente possuísse razão de pixels pretos sobre o total de pixels com um valor entre 0.2 *e* 0.8, e razão de transições branco para preto sobre o total de pretos com valor entre 0.25 *e* 0.8, este componente era considerado um *texto*.

Nesse caso, o componente é adicionado a uma nova lista, que armazena as linhas de texto encontradas na imagem. Além disso desenha-se, com auxílio da função *cv2.rectangle()* a demarcação da linha de texto.

Outro ponto importante é que ao fim da iteração o número de linhas encontradas na imagem será equivalente ao tamanho da lista de linhas criada nessa etapa. Também é importante salientar que os valores escolhidos como *thresholds* foram escolhidos por tentativa e erro.

2.2.8 Identificação de palavras

A última etapa envolvida no projeto foi a identificação de palavras em cada uma das linhas de texto. Para segmentar as palavras percorreu-se elemento a elemento da lista de linhas da etapa anterior, então aplicou-se, para cada linha, uma operação de dilatação, seguida de uma erosão (ambas com um elemento estruturante de (7,1) pixels) e, para finalizar, um fechamento (com um elemento estruturante de (1,11)). Os elementos estruturantes usados nessa etapa, assim como os *thresholds* da etapa anterior, foram escolhidos por tentativa e erro, até ser encontrado o resultado que mais agradou.

Após o processamento das linhas, usou-se a função *cv2.connectedComponentsWithStats()* para encontrar os componentes conexos da linha, esses foram demarcados com o auxílio da função *cv2.rectangle()*. Além disso, para cada componente demarcado acrescentava-se o contador de palavras.

2.3 Restrições da Implementação

Os resultados obtidos para as implementações adotadas foram bem satisfatórios, como será visto nas próximas seções. Contudo, a solução possui algumas restrições.

A primeira delas pode ser observada na etapa responsável por encontrar os componentes conexos, um dos números presentes no grafo que é mostrado na Figura 4 da imagem não foi identificado, provavelmente,

tal fato se deu pela sua proximidade as linhas que conectam os pontos do grafo, os demais números, que estão mais distantes, foram encontrados.

Na separação dos componentes que foram considerados como texto é possível notar mais uma restrição. Palavras com um espaçamento muito grande entre elas foram consideradas como estando em elementos separados, desse modo, a contagem das linhas foi prejudicada, pois algumas linhas, foram contabilizadas duas vezes.

Uma última restrição pode ser observada na etapa que responsável pela separação das palavras de uma linha, houveram alguns casos nos quais acentos e pontuações foram considerados como palavras independentes, o que, assim como no passo anterior, aumentou o número de palavras encontradas pela solução.

3 Resultados

Os resultados obtidos no trabalho podem ser divididos em etapas de pré-processamento e os processamentos finais. O primeiro diz respeito as operações morfológicas que foram aplicadas no processamento inicial da imagem de entrada. O segundo diz respeito as imagens com os textos e palavras demarcados. A Figura 1, por sua vez, nos apresenta a imagem original, sem qualquer alteração.

Na Figura 2 temos a imagem resultante após a aplicação das operações de dilatação e erosão com um *kernel* de 1 pixel de altura e 100 pixels de largura, como pode ser observado foram "borradas"varias das linhas horizontais que tinham algum pixel preto nelas. Já na Figura 3 temos a imagem resultante após a dilatação e a erosão quando usado um *kernel* de 200 pixels de altura e 1 pixel de largura, ao contrário da figura anterior, nesta a maioria dos "borrões"tem sentido vertical, sendo que algumas linhas, inclusive, ficaram intactas. A Figura 4, por sua vez, nos mostra a figura resultante após a intersecção *AND* das figuras anteriores, é possível notar que a Figura 2 predomina na imagem. Por fim, para tirar os ruídos que podemos observar na imagem pós intersecção, aplicou-se um fechamento com *kernel* de 1 pixel de altura e 30 pixels de largura, obtendo como resultado a Figura 5.

Todo o pré-processamento foi feito para que fosse possível identificar os componentes conexos da imagem, que são os observados na Figura 6, vale ressaltar que nessa etapa apenas um número não foi identificado, e o número de componentes conexas encontradas, para conectividade 8, foi 54. E então, identificar os textos presentes na imagem, que podem ser observados na Figura 7 e as palavras presentes na imagem, como visto na Figura 8, em ambas é possível perceber que a maioria das linhas e palavras, respectivamente, foram corretamente detectadas, sendo que alguns problemas podem ser observados como a consideração de acentos e pontuações como palavras na Figura 8 e algumas linhas foram separadas quando a distância entre suas palavras era um pouco maior na Figura 7.

Por fim, o número de linhas encontradas pelo algoritmo foi de 35, é possível notar que dependendo da interpretação do que é considerado uma linha, esse número pode estar um pouco acima do real. Já o número de palavras identificadas foi de 241, o número também não é perfeito, já que considera alguns acentos e pontuações como palavras.

4 Discussão

Analisando as imagens resultantes após a aplicação das operações de dilatação e erosão para dois *kernels* diferentes, é notável a diferença que as dimensões deste trazem a imagem final. Na Figura 2, que se usou um *kernel* horizontal ($1x100$ pixels) há mais ruídos do que na Figura 3, na qual se aplicou um *kernel* vertical ($200x1$ pixels). Esta, por outro lado, acaba "borrando"vários espaços brancos da imagem, além de não trazer diferenças significativas para espaços pequenos de texto, como o título "Fig 4. ..."que está no canto inferior esquerdo da Figura 1.

Ao analisar a imagem resultante após a aplicação da intersecção *AND* as Figuras 2 e 3, é possível perceber que a maioria dos espaços em brancos que o *kernel* vertical "borrou"são ignorados e, de certa forma, prevalecem os resultados obtidos pós aplicação do *kernel* horizontal, ambas as figuras são bem semelhantes inclusive. No entanto, é possível notar que a Figura 4 possuí muito ruídos, há vários "riscos brancos"no meio de blocos de pretos, para corrigir esses ruídos, fez-se uso da técnica do fechamento, visto que, de acordo com a teoria 2, ela é útil para fechar pequenos orifícios dentro dos objetos de primeiro

plano. O resultado obtido, que é mostrado na Figura 5, condiz com o esperado, trazendo uma imagem "mais limpa"e bem demarcada.

O objetivo principal do projeto era segmentar a imagem em *texto* e *não texto* e o texto, por sua vez, em *palavras*. Para atingir tal objetivo, foi preciso, primeiramente, encontrar os componentes conexos presentes na imagem pós os passos de pré-processamento (dilatação, erosão e fechamento). O resultado obtido para essa etapa (Figura 6) foi bastante satisfatório, ao olhar a imagem com detalhes, é possível perceber que o único componente não identificado foi um número "3"no grafo da "Fig. 4"da imagem original (Figura 1). Acredita-se que o motivo que levou a essa situação foi a proximidade do número com a imagem do grafo, visto que, dos números que a figura possui ele é o que está mais próximo e também, foi o único não identificado separadamente.

Após identificar os componentes conexos, cada um deles foi classificado como sendo *texto* ou *não texto*, e os que foram considerados *textos* foram demarcados (vide Figura 7). O resultado novamente foi bastante satisfatório, o que mostra que os valores escolhidos como *threshold* foram condizentes. Nessa etapa é importante ressaltar que uma técnica melhor para escolher os limiares poderia trazer resultados ainda mais expressivos. É importante notar que, devido a um critério de implementação adotado, o números presentes na "fig.4"da imagem original não foram considerados texto. Nessa etapa, também foi possível contar o número de linhas presentes na imagem, o resultado obtido contudo, não foi perfeito, já que, ele considera palavras muito distantes entre si, como estando em linhas diferentes, logo, aumentando o valor correto.

Por fim, para cada linha encontrada, aplicou-se novas técnicas de pré-processamento e buscou-se os componentes conexos nelas, com o objetivo de ser possível identificar o que seria considerado uma *palavra* e o que não seria. Novamente, o resultado encontrado (vide Figura 8) foi muito próximo ideal, os únicos pontos negativos foram que alguns acentos foram considerados como independentes da palavra a eles associada. No geral, o número de palavras encontrado também foi satisfatório.

5 Conclusão

Após a finalização do trabalho, é possível concluir que a aplicação de operações morfológicas para a segmentação de textos em imagens binárias apresenta resultados bastante satisfatórios e possuem uma implementação bastante simples. Portanto, são um bom caminho para atingir o objetivo proposto.

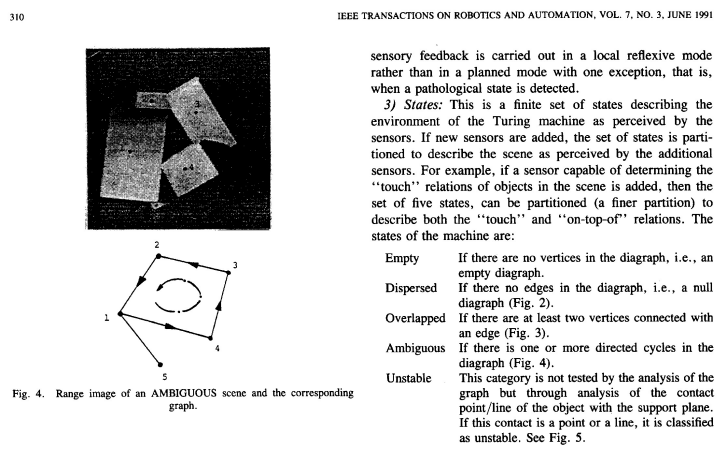
Dada as restrições comentadas ao longo deste relatório é possível perceber que, para iterações futuras, alguns pontos tem a possibilidade de implementações mais complexas que trariam resultados ainda mais expressivos do que os já obtidos. Uma das melhorias que podem ser implementadas, por exemplo, é o uso de técnicas mais aprimoradas, como um aprendizado de máquina ou algo semelhante, na obtenção dos *thresholds* usados para classificação dos textos. Além disso, a contagem de transições branco para preto poderia ser feita considerando a mudança em ambos os sentidos, o que poderia trazer melhores resultados também.

Referências

[1] Enunciado do projeto 3. Disponível em: http://www.ic.unicamp.br/~helio/disciplinas/MC920/trabalho3.pdf, Acesso em: 20-05-2019.

[2] Operações morfológicas opencv. Disponível em: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html, Acesso em: 20-05-2019.

ANEXOS



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) Figura original sobre a qual se aplicou as operações morfológicas.

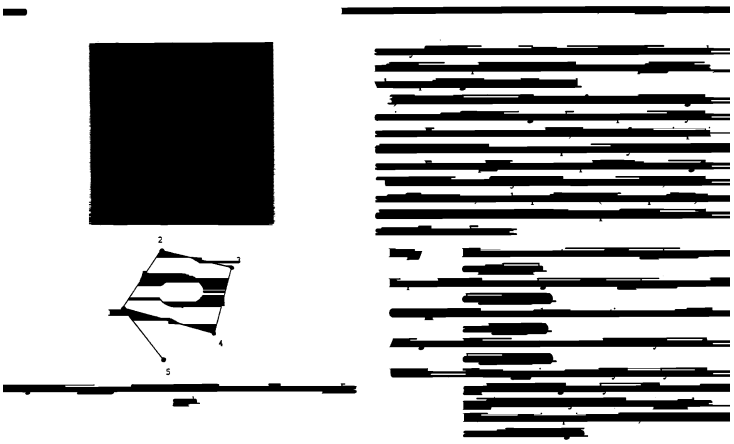


Figura (2) Figura após aplicação da dilatação e erosão para o elemento estruturante de 1x100 pixesl.

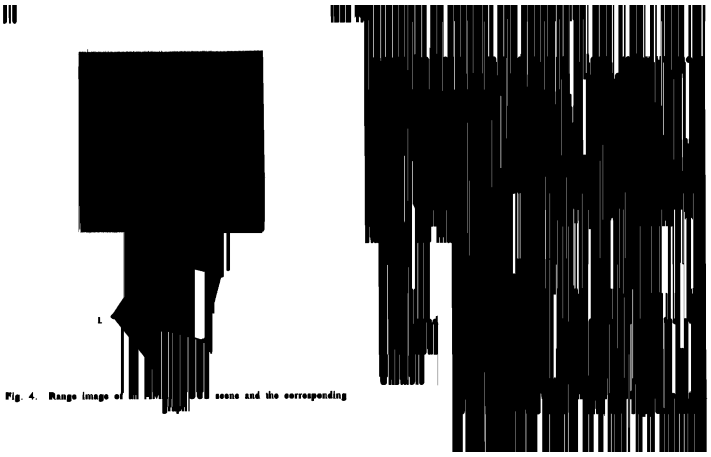


Figura (3) Figura após aplicação da dilatação e erosão para o elemento estruturante de 200x1 pixesl.

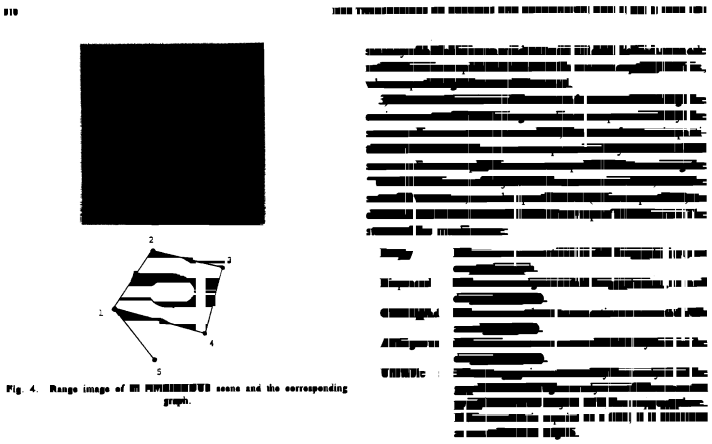


Figura (4) Figura após aplicação da intersecção AND entre as Figuras 2 e 3.

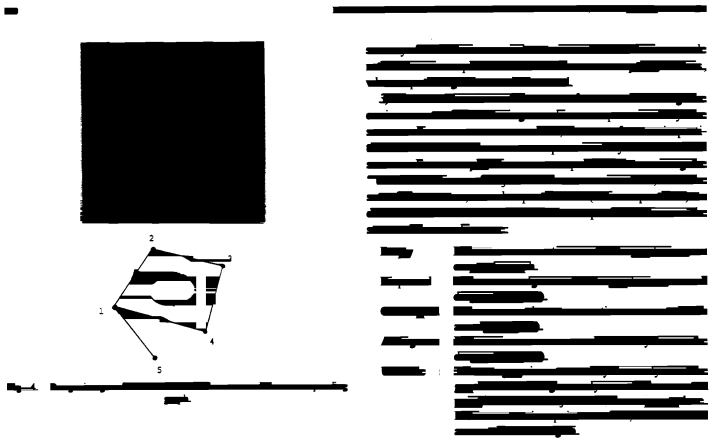
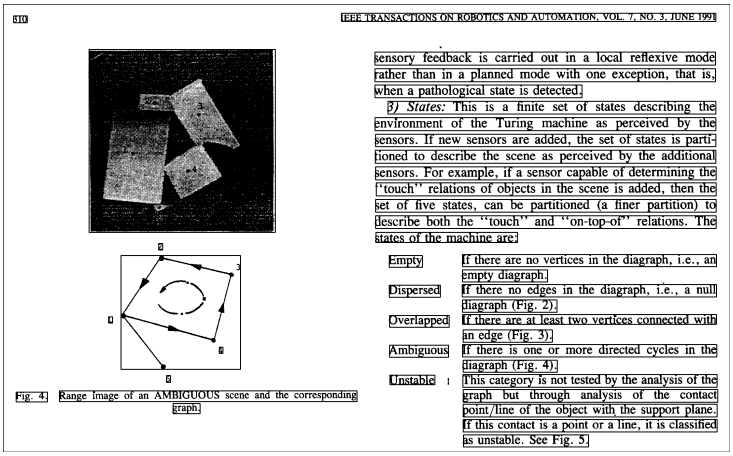


Figura (5) Figura após aplicação da operação de fechamento a Figura 4.



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.

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

Figura (6) Figura com os componentes conexos da imagens marcados com um retângulo.

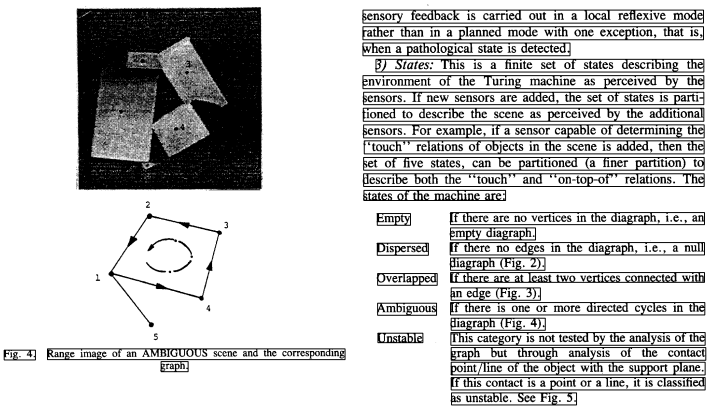


Figura (7) Figura com os textos da imagens marcados com um retângulo.

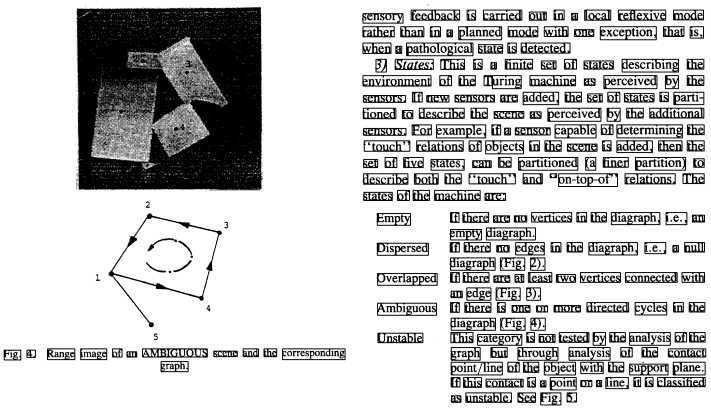


Figura (8) Figura com as palavras da imagens marcados com um retângulo.