MC920 – Introdução ao Processamento Digital de Imagem  
Trabalho 4

Rafael Eiki Matheus Imamura - RA 1761271

1Instituto de Computação – Universidade Estadual de Campinas (Unicamp)  
Caixa Postal 6176 – CEP 13083-970 – Campinas – SP – Brasil

ra176127@students.ic.unicamp.br

# 1. Introdução

Em diversas aplicações, é de grande interesse poder combinar imagens a fim de se obter uma visão mais completa da figura fotografada. Desde entretenimento e desejos pessoais até aplicações como visualizações por satélite, as fotos panorâmicas são bem versáteis em sua utilidade. No entanto, fazer esse processo manualmente pode ser bastante custoso, além de impreciso.

Neste trabalho foi criado um algoritmo que a partir de duas imagens   
JPEG (*Joint Photographic Experts Group*), gera uma foto panorâmica, combinando as duas. Além de gerar a imagem combinada, também são mostrados os pontos de semelhança da imagem. O algoritmo possibilita o uso de algoritmos de descritores de imagens diferentes e suas implicações são discutidas neste relatório.

# 2. Execução

A entrega deste trabalho inclui o presente PDF e um diretório de nome “projeto 4”. Nela se encontram o código, as dependências e as imagens de entrada usadas para os testes.

Para executar o código, deve-se ter instalado o OpenCV e o Numpy. Um *freeze* das dependências está presente no arquivo “requirements.txt”. Uma alternativa rápida para fazer a instalação é o comando:

*pip install -r requirements.txt*

Com este comando instalado, é possível chamar o arquivo de script de duas formas.

1. *python main.py example threshold*

Nesta opção, são gerados os resultados para todas as imagens de exemplo disponíveis no diretório “pictures”, usando todos os algoritmos, com o limiar dado pelo valor de “threshold”.

1. *python main.py arq1 arq2 algoritmo limiar diretório\_de\_saída arq\_saída*

Nesta opção, é gerada a imagem usando um algoritmo de descritor específico. O significado de cada parâmetro é detalhado a seguir:

1. *arq1* e *arq2*: arquivos de imagem de entrada, no formato JPEG;
2. *algoritmo*: algoritmo do descritor a ser usado. Pode ser *orb* ou *brief*.
3. *limiar*: valor do limiar para considerar uma correspondência boa;
4. *diretório\_de\_saída:* diretório de saída das imagens (deve incluir uma “/” no final);
5. *arq\_saída:* nome base do arquivo de saída (sem extensão).

Alguns exemplos de uso são mostrados a seguir:

1. python main.py example 0.7
2. python ./main.py ./pictures/foto1A.jpg ./pictures/foto1B.jpg orb 0.75 ./results/ foto1
3. python ./main.py ./pictures/foto2A.jpg ./pictures/foto2B.jpg orb 0.8 ./results/ foto2

## 2.1. Entrada

A entrada, em todos os casos, é dada por imagens JPEG. Para este relatório, foram usadas 6 imagens, das quais uma foi fornecida no enunciado do trabalho (*http://www.ic.unicamp.br/~helio/imagens\_registro/*).

Todas as entradas usadas estão disponíveis dentro do zip, no diretório “pictures”.

## 2.2. Saída

A saída do programa são imagens no formato JPEG. Para cada par de imagens é gerado um par de imagens com os nomes “BASE\_lines.jpeg” e “BASE\_panoramic.jpeg”. Elas representam, respectivamente, as semelhanças entre as duas imagens, ligando os pontos por retas, e a imagem panorâmica resultante. Em casos de que não sejam encontradas correspondências o suficiente (mais detalhes a seguir), o programa não gera as imagens e um erro é dado.

# 3. Solução

A solução geral criada possui o seguinte formato:

1. **Ambas as imagens de entrada são lidas e convertidas em escala de cinza**

O primeiro passo é converter as imagens em níveis de cinza. Ambas as imagens (original, colorida) e em tons de cinza são guardadas.

1. **São encontrados os pontos de interesse e descritores para cada imagem, usando um dos algoritmos (ORB ou BRIEF)**

Os pontos de interesse e descritores de cada uma das imagens (versão escala de cinza) são computados. Como funciona o processo depende do algoritmo escolhido, o que é detalhado mais adiante.

1. **As similaridades entre cada descritor são computadas e são selecionados as melhores correspondências**

Neste passo, são procuradas as similaridades entre as imagens, com um matcher do OpenCV que depende do algoritmo que é usado. Neste caso em específico, os matchers do ORB e BRIEF são iguais, pois usam o *cv2.NORM\_HAMMING* sem *crossCheck*. Mais detalhes destes métodos podem ser vistos na documentação do OpenCV.

As similaridades são então filtradas por pares, sendo mantidas apenas aquelas que satisfazem a fórmula a seguir. Dadas duas correspondências A e B, a correspondência A é armazenada se:

*Distancia(A) < Distancia(B) \* limiar*

Onde *Distancia(X)* é a informação da distância da correspondência e o *limiar* é uma das entradas do programa. Quanto menor o limiar, mais restritivo é a seleção de similaridades.

1. **É executada a técnica RANSAC (*RANdom SAmple Consensus*), estimando a matriz de homografia**

Usando o método do OpenCV *cv2.findHomography* com o parâmetro *cv2.RANSAC*, é estimada a matriz de homografia, usada para realizar a projeção posteriormente.

1. **Usando as informações anteriores das imagens em escala de cinza, as versões coloridas das imagens são alinhadas e sobrepostas, formando a imagem panorâmica**

Usando o método *cv2.warpPerspective* do OpenCV, uma nova imagem é criada, com a soma da altura das duas imagens originais, e largura de ambas as imagens somadas. A imagem é posicionada levando em conta a posição da figura a esquerda. A imagem resultante é preta nas regiões que não há conteúdo da foto panorâmica. Essas regiões pretas são removidas no final do processo.

1. **As imagens coloridas também geram uma nova imagem, onde há segmentos de retas ligando os pontos de semelhança**

É gerada uma imagem que ambas as imagens de entrada estão lado a lado. Suas semelhanças são destacadas através de retas que as ligam.

# BRIEF (Binary Robust Independent Elementary Features) é uma técnica de descrição de *features*. Usando uma seleção de pares da imagem, ele se apoia na velocidade de cálculo da distância de Hamming para strings binárias que representam os descritores. Um ponto fraco do BRIEF é possuir problemas com rotações em imagens. Este não é um método que encontra os pontos de interesse da imagem, e por isso é aplicado no projeto usando o CenSurE (*STAR detector* do OpenCV).

O ORB (*Oriented FAST and Rotated BRIEF*) é uma alternativa de boa performance e com custo computacional páreo ao de alguns grandes concorrentes, como o SIFT e o SURF. Ele usa o FAST para encontrar os pontos de interesse e descritores BRIEF. O algoritmo ainda possui modificações que encobrem problemas conhecidos de sua base, já que o FAST é sensível a rotações e o BRIEF tem dificuldades com elas.

# 4. Resultados

As imagens resultantes passaram por diversos processos. A seguir são mostrados alguns passos a fim de clarificar o algoritmo criado. A imagem “text.pbm” será usasda como o principal exemplo (Figura 1).

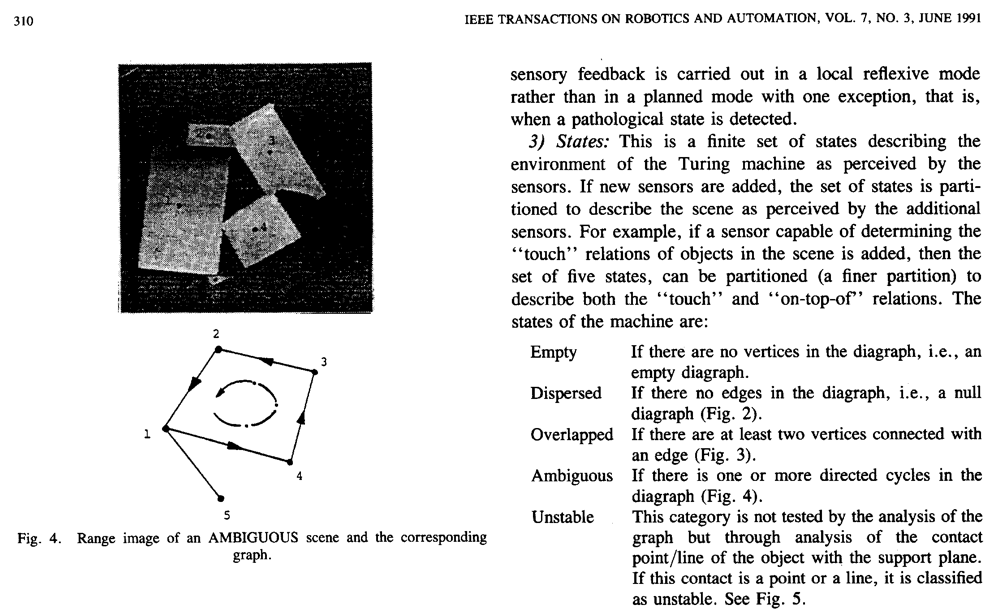


Figura 1. Versão original de “text.pbm”.

Os 6 primeiros passos do programa seguem o padrão sugerido no enunciado. Quando terminado, esses 6 passos são consolidados em uma imagem temporária intermediária (Figura 2), usada pelo programa *comp\_conexos.c*.

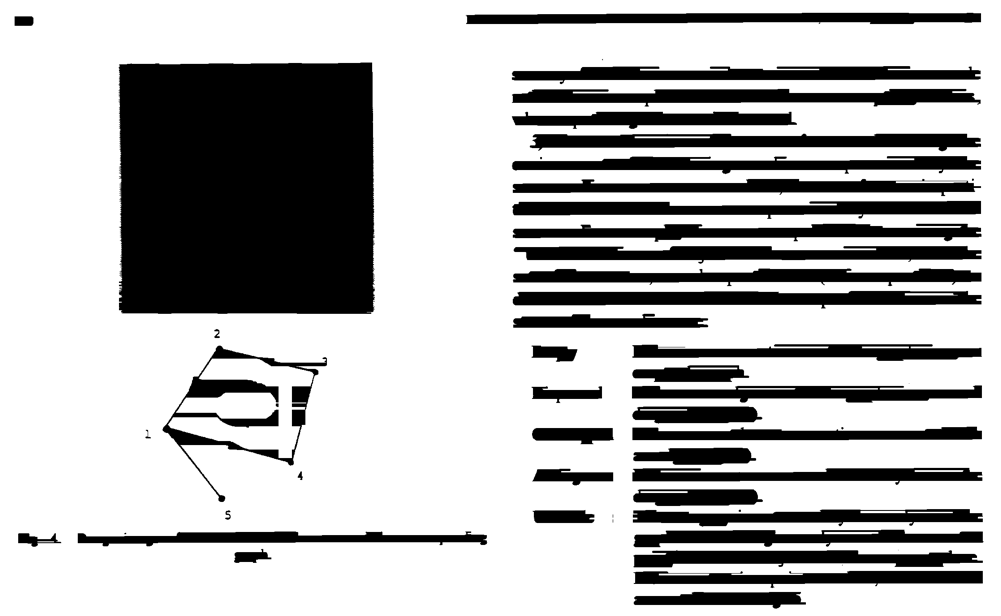


Figura 2. Resultado dos 6 primeiros passos em “text.pbm”.

O estado dessa imagem (Figura 2) foi usado para verificar quais seriam os componentes que poderiam ser descobertos ou não. Nesse exemplo, é possível ver o que número “3”, próximo aos números 1, 2, 4 e 5, foi fundido com a figura. Assim, ele não poderia ser descoberto como um caractere posteriormente.

O resultado do passo 7 é dado em texto e em imagem. Neste exemplo, foram encontradas 53 componentes conexas e a imagem resultante pode ser vista na Figura 3.

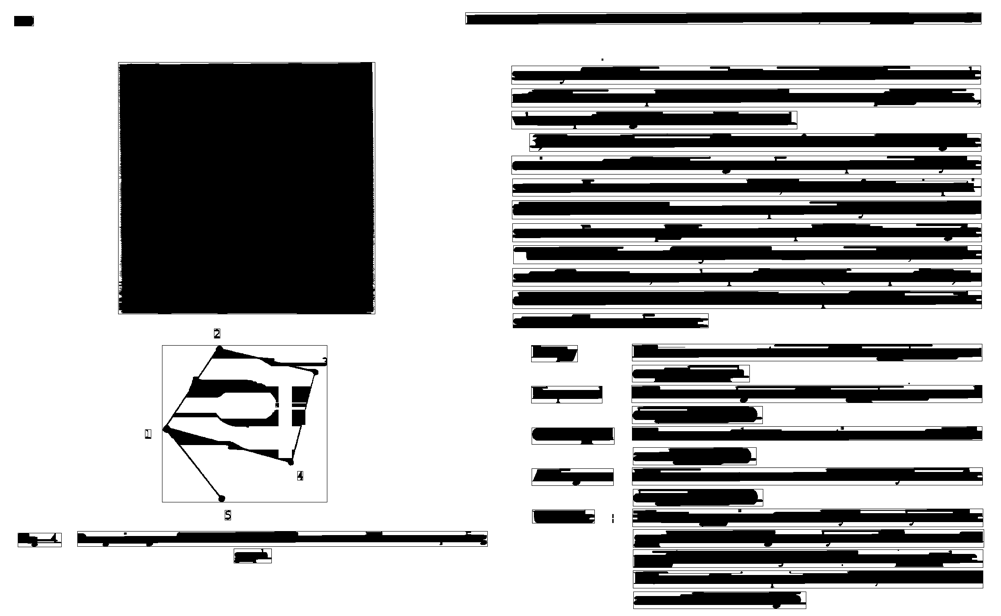


Figura 3. Retângulos envolvem cada componente conexa de “text.pbm”.

Para classificação em texto, a seguinte regra foi aplicada em cada componente:

Onde:

* BP é a proporção de pixels pretos pela área da componente;
* VT é a razão de transições verticais de branco para preto (percorrendo de cima para baixo) e o total de pixels pretos;
* HT é a razão de transições horizontais de branco para preto (percorrendo da esquerda para a direita) e o total de pixels pretos.

A quantidade de textos nas componentes representava aproximadamente essa quantidade de texto, enquanto a quantidade de transições verticais e horizontais não poderia ser excessiva para a imagem “text.pbm”. No entanto, comparando com as demais imagens, em alguns casos, uma quantidade de pixels pretos poderia ser menor quando as transições verticais e hortizontais eram mais frequentes. Com essa classificação de texto, todos os textos presentes em componentes identificados no passo 7 foram encontrados.

Para a detecção de palavras, foram desenvolvidos 2 algoritmos: um que considera o desvio padrão do tamanho dos espaços e cria um limiar para ver quais são espaços entre caracteres e quais são entre palavras; e outro, que usa o operador morfológico de fechamento.

O algoritmo que usa o limiar parte da ideia de que existem 2 tipos de espaços nas linhas de texto: espaços entre caracteres (que são consideravelmente menores) e espaços entre palavras. Para escolher o limiar, é calculado o tamanho de todos os espaços (os espaços são considerados quando a quantidade de pixels pretos na vertical é baixa (menos de 5% da altura da componente). O limiar é a média dos tamanhos. Foi tentado usar a mediana, mas os resultados foram significamente piores. Espaços acima do limiar são espaços entre palavras. Para saber se o componente é uma única palavra, é checado se o desvio é pelo menos um pouco acima do limiar, tendo assim uma diferença grande entre o tamanho dos espaços.

O algoritmo que usa o operador morfológico aplica um fechamento em cada componente conexa de texto a fim de transformar cada palavra em blocos. O elemento estruturante é de:

* Altura: 2 vezes a altura da componente;
* Largura:

A altura é usada para dilatar o texto para a altura toda da componente. A largura é justificada como o tamanho aproximado do espaço para a fonte. A constante 0.62 é usada para calcular o tamanho mínimo de espaço, independente do tamanho da fonte.

Depois de ter esses blocos, é aplicado o algoritmo do limiar, com o valor forçado a ser 0, aceitando qualquer espaço existente. Um exemplo do resultado do processo de criação dos blocos pode ser visto na Figura 4.

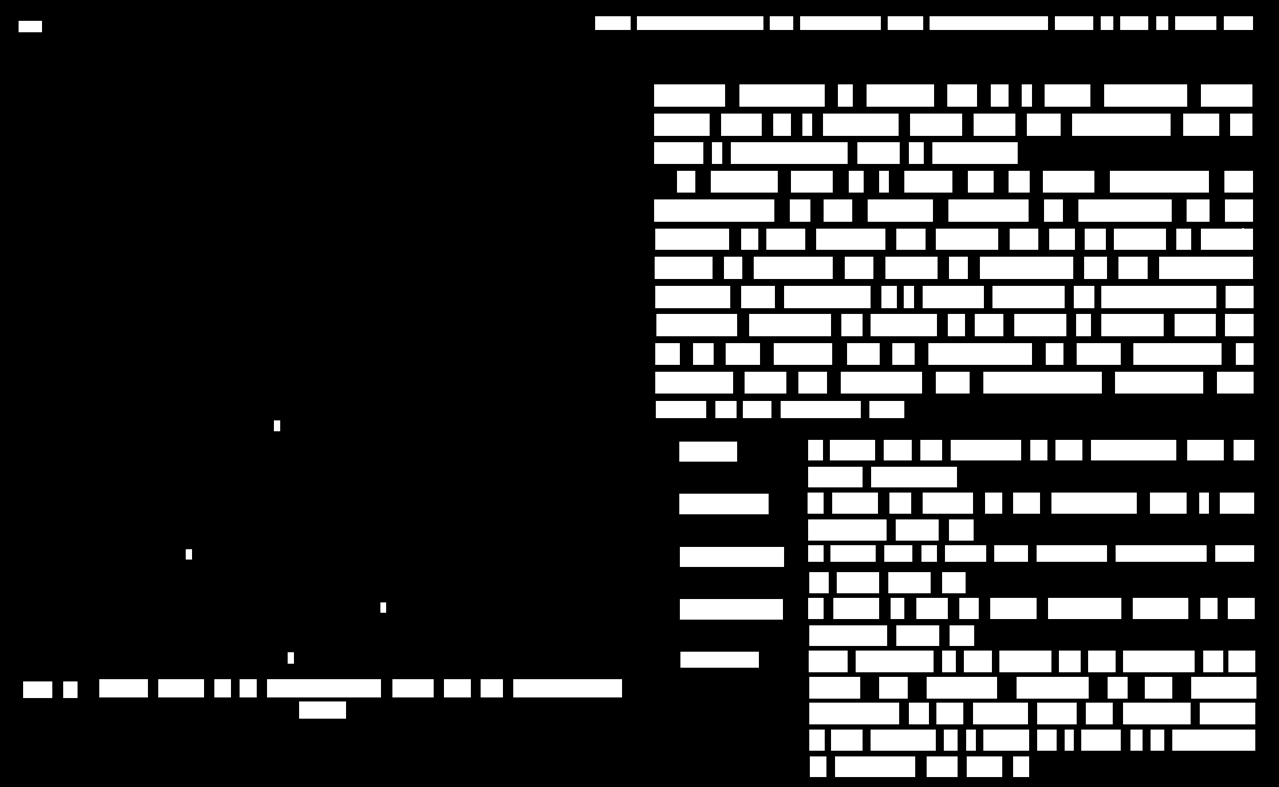


Figura 4. “text.pbm” após ser aplicado o fechamento em cada componente.

Os resultados de cada algoritmo para essa entrada são:

1. Com limiar: 40 linhas e 246 palavras (Figura 5);
2. Com operador morfológico: 40 linhas e 242 palavras (Figura 6).

A Figura 5 possui maiores erros. O título é juntado todo em uma palavra só, e a pontuação é separada como uma palavra a parte em diversos casos, como nas aspas e vírgulas após pontos. No entanto, ele é um algoritmo que é mais rápido, não sendo necessário a aplicação de um operador morfológico e tendo um erro não muito grande neste caso em específico (de 10 palavras detectadas como 1 e 13 pontuações detectadas como palavras). A Figura 6 detectou precisamente todas as palavras e suas pontuações.

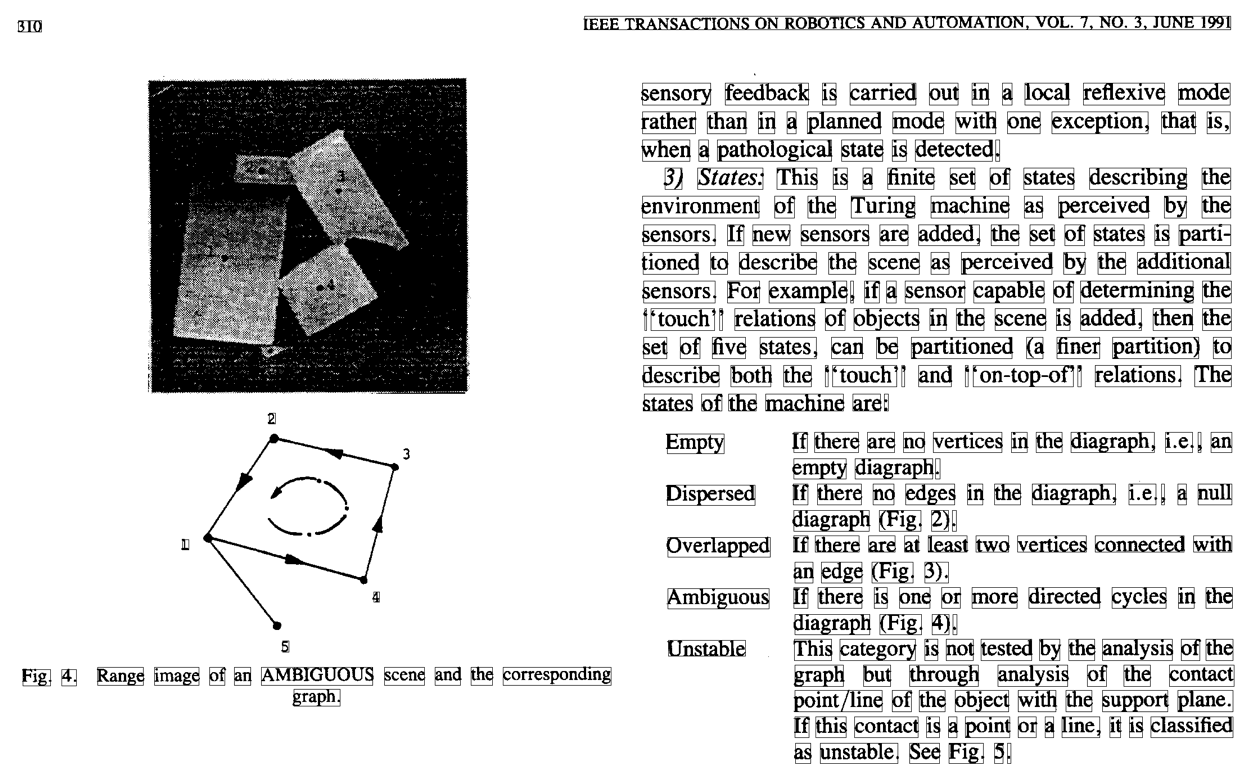


Figura 5. Resultado final de “text.pbm” usando o limiar no passo 10.

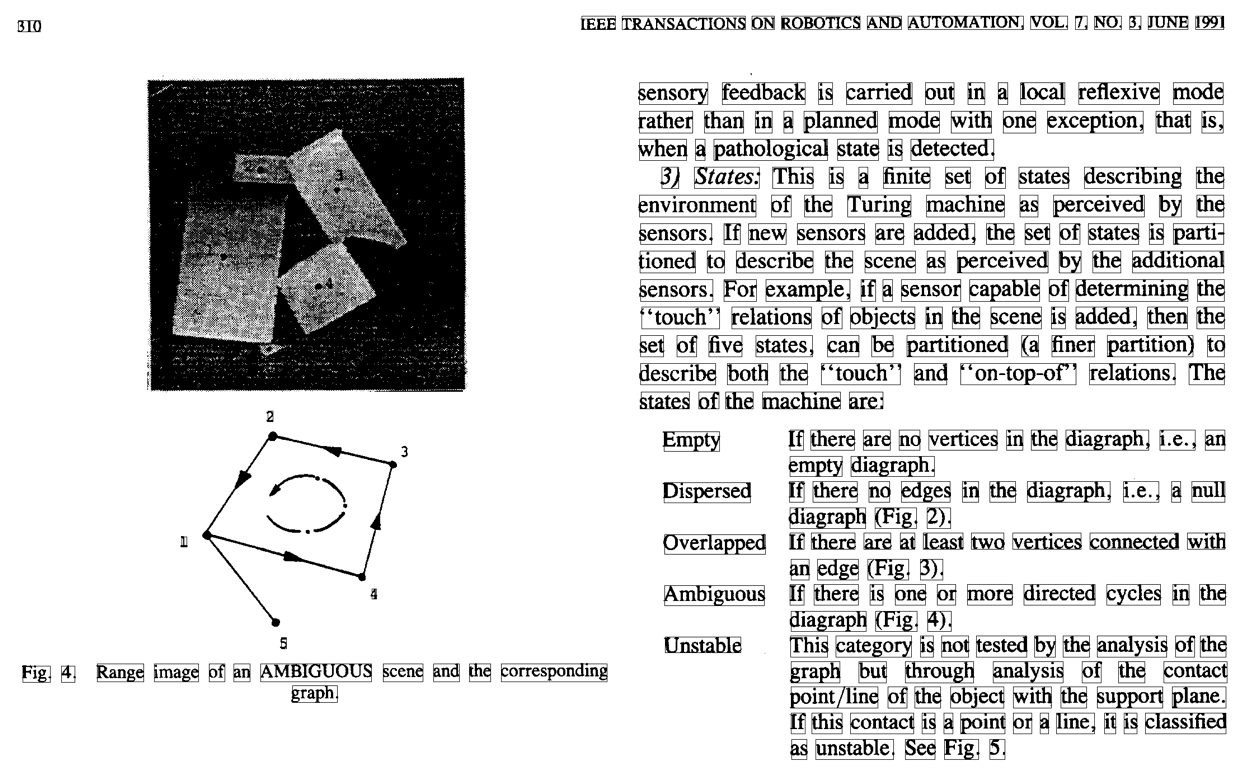


Figura 6. Resultado final de “text.pbm” usando o operador morfológico no passo 10.

Alguns outros exemplos de imagens usadas para os testes são mostrados a seguir. Em todos os casos, foi aplicado o operador morfológico para encontrar as palavras.

Figura 7. Entrada e saída para a imagem “letter\_a.pbm”.

Figura 8. Entrada e saída para a imagem “scs.pbm”.

As Figuras 7 e 8 foram perfeitamente detectadas. São casos de letras grandes e uma única palavra presente na imagem.

A close up of a keyboard

Description automatically generated

Figura 9. Imagem original de entrada “map.pbm”.

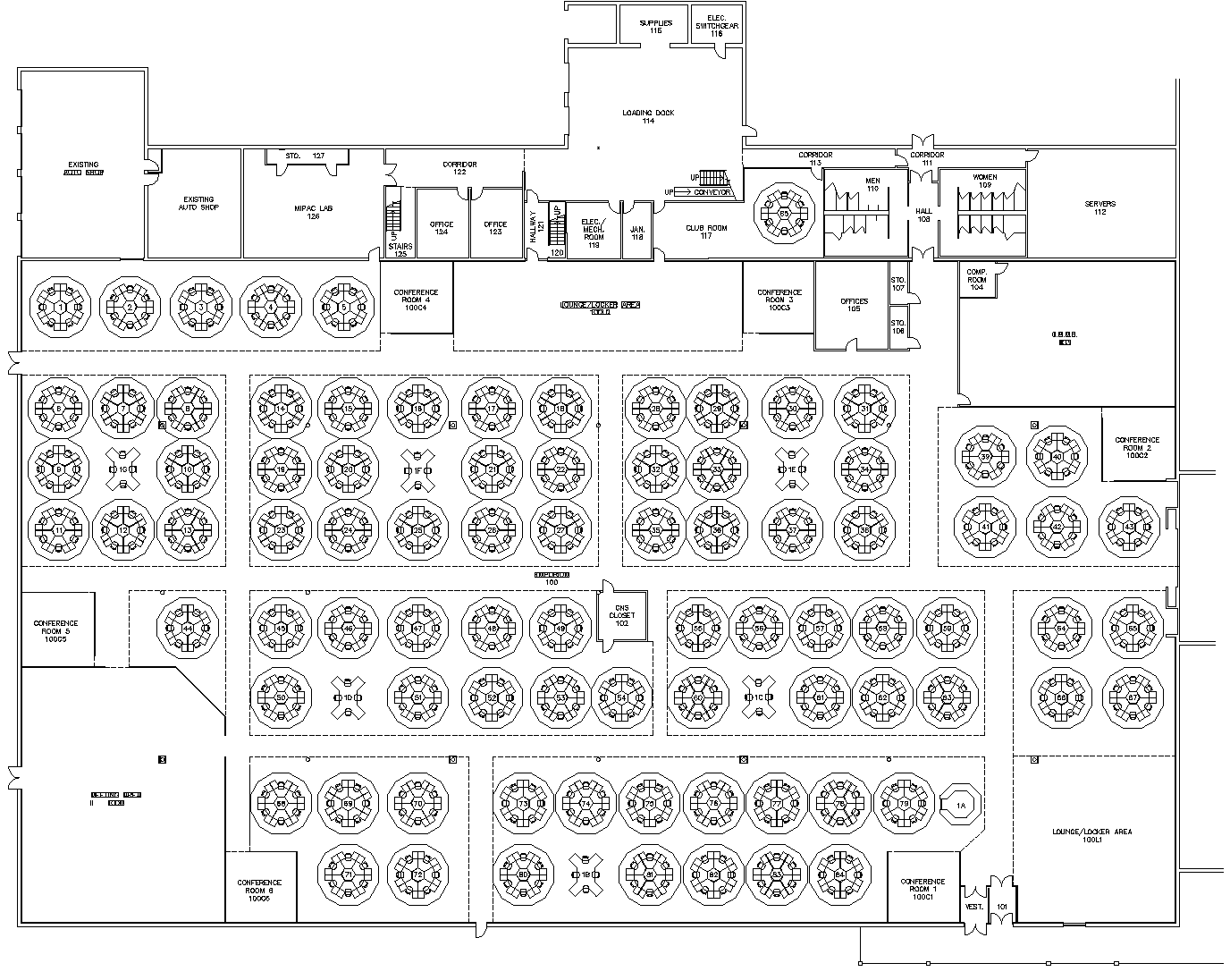


Figura 10. Imagem de saída para “map.pbm”.

As Figuras 9 e 10 apresentam um caso complexo, com diferentes padrões misturados com letras pequenas em disposições diferentes. O algoritmo foi capaz de detectar 9 linhas com 18 palavras, espalhadas em posições como a parte inferior esquerda, central superior, e no centro. É um caso particularmente difícil, mas que foi possível ainda assim extrair alguns dados.

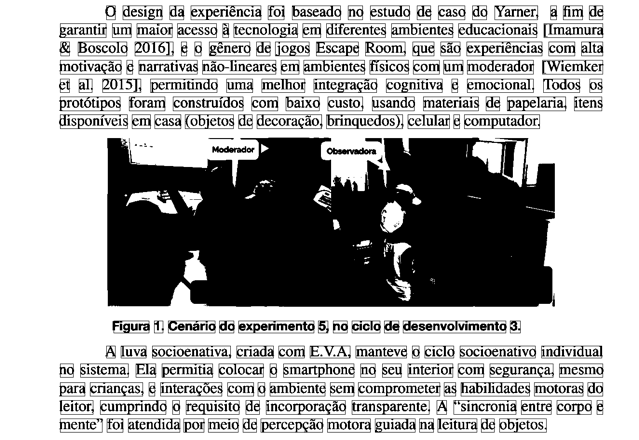
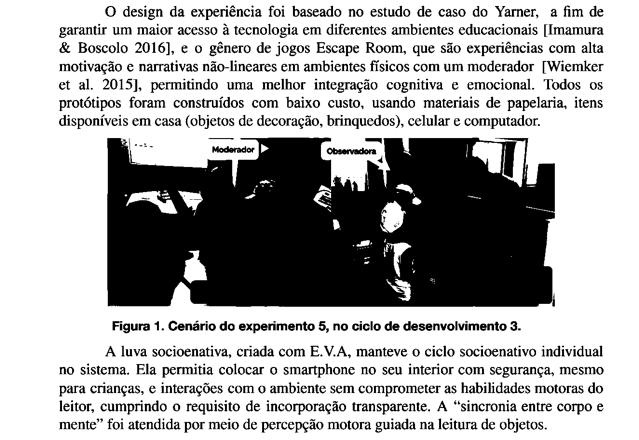


Figura 11. Entrada e saída para “paper.pbm”.

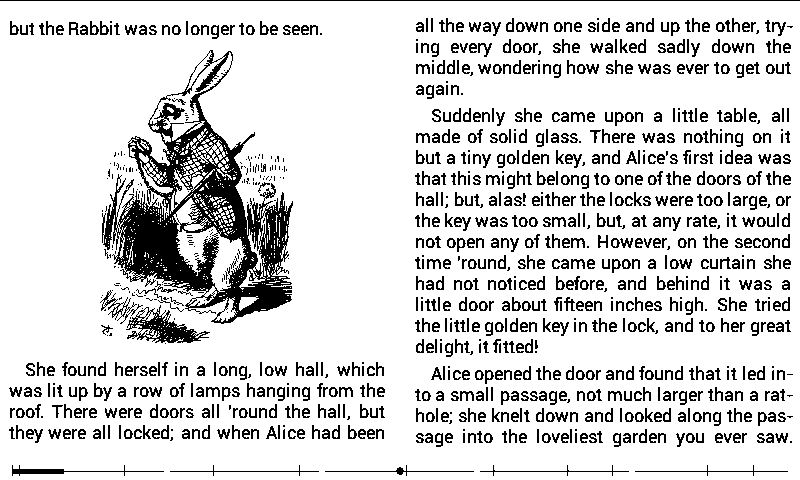
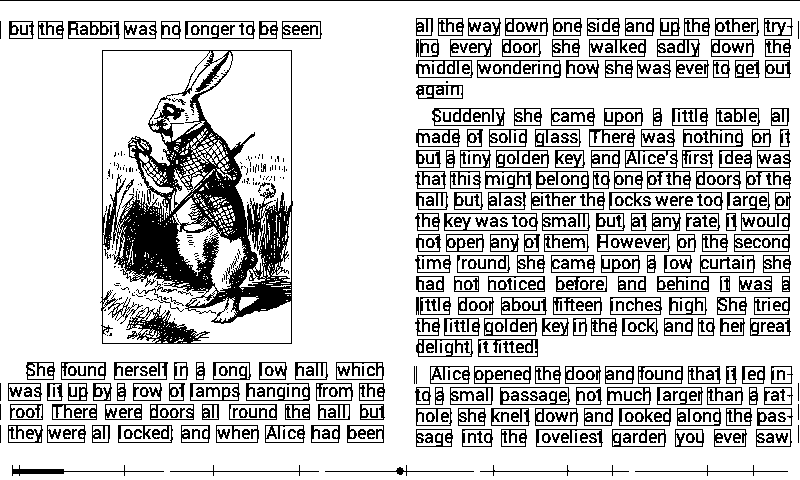
 

Figura 12. Entrada e saída da imagem “rabbit.pbm”.

Nas Figura 11 e 12 ocorrem os casos mais “comuns”, em que há texto e figuras nas imagens. Na Figura 11, é mostrado um artigo (autoria própria), em que o texto foi perfeitamente detectado, tanto as linhas como as palavras (14 linhas, 159 palavras). Já na Figura 12, o desenho do coelho foi detectado como uma palavra, totalizando 26 linhas e 220 palavras, o único caso de falso positivo dos casos de teste.

# 5. Conclusões

Este trabalho apresentou um algoritmo para detecção de palavras e linhas em imagens binárias no formato PBM. Foram criadas 2 variações do algoritmo, usando o operador morfológico e uma versão mais simples. A versão com o operador morfológico obteve resultados melhores, enquanto a versão usando limiar possui um desempenho maior. Em alguns casos, a diferença nos resultados foi bastante grande.

As imagens testadas apresentaram alguns casos de falso negativo, como na imagem “map.pbm”. Esses casos parecem estar atrelados a textos com fontes pequenas, próximas a outras figuras. Apenas um caso de falso positivo foi encontrado, na imagem “rabbit.pbm”. Uma limitação dos testes é que os textos estavam alinhados horizontalmente para serem detectados.

De forma geral, os resultados são bastante satisfatórios. Para detecção de palavras em tamanhos e espaçamento regulares, o algoritmo funcionou bem. Adaptações podem ser necessárias para determinadas imagens, mas no algoritmo proposto ele faz um trabalho genérico bastante razoável.

Para ser usado em situações mais genéricas, o algoritmo precisaria analisar outros possíveis problemas das imagens, como desalinhamento do texto, fontes pequenas e textos próximos de imagens.

# Referências

OpenCV. (2013) “BRIEF (Binary Robust Independent Elementary Features)”. Disponível em: < <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_brief/py_brief.html>>. Acesso em: jun. 2019.

OpenCV. (2013) “ORB (Oriented FAST and Rotated BRIEF)”. Disponível em: < <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html>>. Acesso em: jun. 2019.

SciPy.org. (2019) “NumPy Reference”. Disponível em: < https://docs.scipy.org/doc/numpy/reference/>. Acesso em: jun. 2019.