MC920 – Introdução ao Processamento Digital de Imagem  
Trabalho 3

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

Uma atividade que é cada vez mais comum, com a popularização de dispositivos celulares com câmeras, é tirar fotos de documentos. Isso pode acontecer desde digitalização de documentos até anotação rápida de informações. Em muitos desses casos, é interessante a detecção de conteúdo textual presente na imagem. Um primeiro passo para usar uma tecnologia de detecção de caracteres (Optical Character Recognition – OCR) é conseguir segmentar uma imagem em linhas e palavras.

Neste trabalho, foi criado um algoritmo para detecção de palavras e linhas de texto em uma imagem binária no formato PBM (Portable BitMap). As imagens foram testadas com o algoritmo e os resultados e escolhas do algoritmo são descritos a seguir.

# 2. Execução

A entrega deste trabalho inclui o presente PDF e um diretório de nome “projeto 3”. 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, basta chamar o arquivo de script com o parâmetro desejado:

*python main.py \*opções arquivo\_de\_entrada arquivo\_de\_saida flags*

O significado de cada parâmetro da entrada é:

1. *Opções*: parâmetro opcional:
   1. *example*: gera todas os casos de exemplo (diretório “pictures”).
2. *Arquivo\_de\_entrada*: caminho para a imagem base. Este parâmetro não é usado para a opção *example*.
3. *Arquivo\_de\_saida*: caminho para o arquivo de saída da imagem. A opção *example* não usa este parâmetro – as imagens são geradas no diretório “results”.
4. *Flags*: caso exista, é aplicado o efeito definido:
   1. *-a*: usa o algoritmo do limiar ao invés do operador morfológico para o passo 10 (mais explicações a frente).

Alguns exemplos de uso são mostrados a seguir:

1. python main.py example
2. python main.py./pictures/text.pbm ./results/text.pbm
3. python main.py ./pictures/text.pbm ./teste.pbm -a

## 2.1. Entrada

A entrada, em todos os casos, é dada por imagens PBM. Para este relatório, foram usadas 6 imagens, das quais uma foi fornecida no enunciado do trabalho (*http://www.ic.unicamp.br/~helio/imagens\_morfologia/*), imagens do diretório PBMA e capturas de tela de um artigo (autoria própria) e um livro (domínio público).

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

## 2.2. Saída

A saída do programa são imagens no formato PBM binária (P4) e a quantidade de palavras e linhas encontradas na saída do programa. 1 na imagem representa preto, enquanto 0 representa branco. A imagem de saída possui cada palavra encontrada contornada por um retângulo preto.

# 3. Solução

O método adotado segue as indicações do enunciado do trabalho até o passo 6:

1. Dilatação da imagem original com um elemento estruturante de 1 pixel de altura e 100 pixels de largura;
2. Erosão da imagem resultante com o mesmo elemento estruturante do passo (1);
3. Dilatação da imagem original com um elemento estruturante de 200 pixels de altura e 1 pixel de largura;
4. Erosão da imagem resultante com o mesmo elemento estruturante do passo (3);
5. Aplicação da intersecção (AND) dos resultados dos passos (2) e (4);
6. Fechamento do resultado obtido no passo (5) com um elemento estruturante de 1 pixel de altura e 30pixels de largura.

O passo 7 pede a identificação de componentes conexos. Foi usado o programa fornecido (*comp\_conexos.c*) com uma modificação: é escrito na saída do programa as coordenadas dos 2 pontos que definem cada retângulo das componentes conexas. A saída do programa é então redirecionada para um arquivo, que é lido e processado para realizar o passo 8.

1. Uso do programa modificado para detectar componentes conexas;
2. Cálculo da quantidade de pixels pretos, proporção quanto a área total e quantidade de transições;
3. Classificação das componentes como texto e não-texto, feito usando as proporções de pixels pretos e transições horizontais e verticais;
4. Segmentação das linhas em blocos de palavras.

Os intervalos de valores aceitos no passo 9 foram usados com análises das saídas do programa para diferentes imagens. Para o passo 10, a segmentação foi feita usando o operador morfológico de fechamento, com elemento estruturante de altura 2 vezes maior que a altura da linha e largura proporcional a parte da altura da linha. Isso foi feito pensando que a altura da linha determina aproximadamente o tamanho da fonte usada, e o espaço entre as palavras será aproximadamente proporcional à fonte. Os valores exatos foram testados empiricamente com as imagens. Mais desses testes serão descritos mais adiante.

# 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).

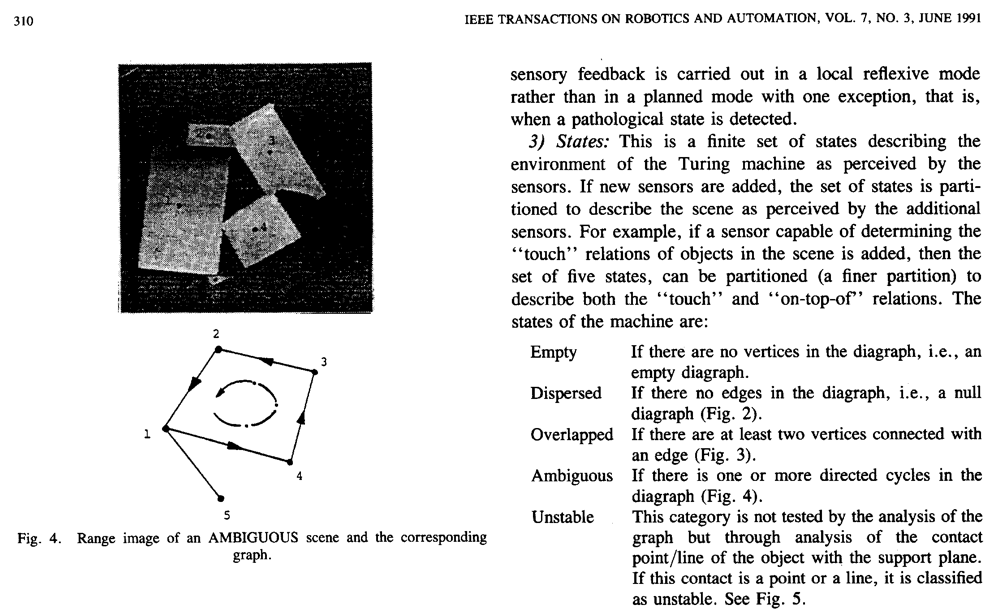


Figura 1. Versão original de “text.pbm”.

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*.

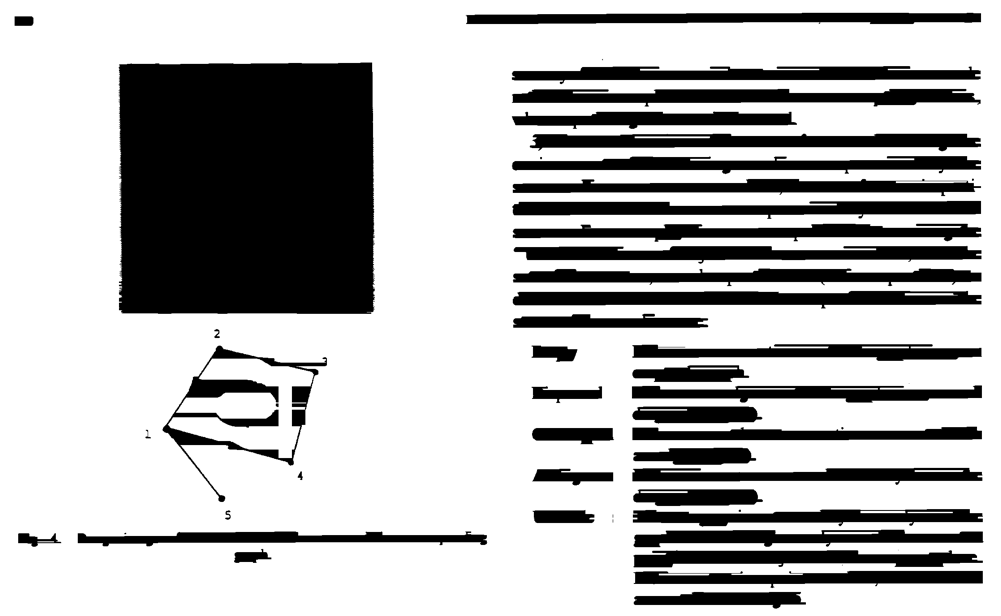


Figura 2. Resultado dos 6 primeiros passos em “text.pbm”.

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.

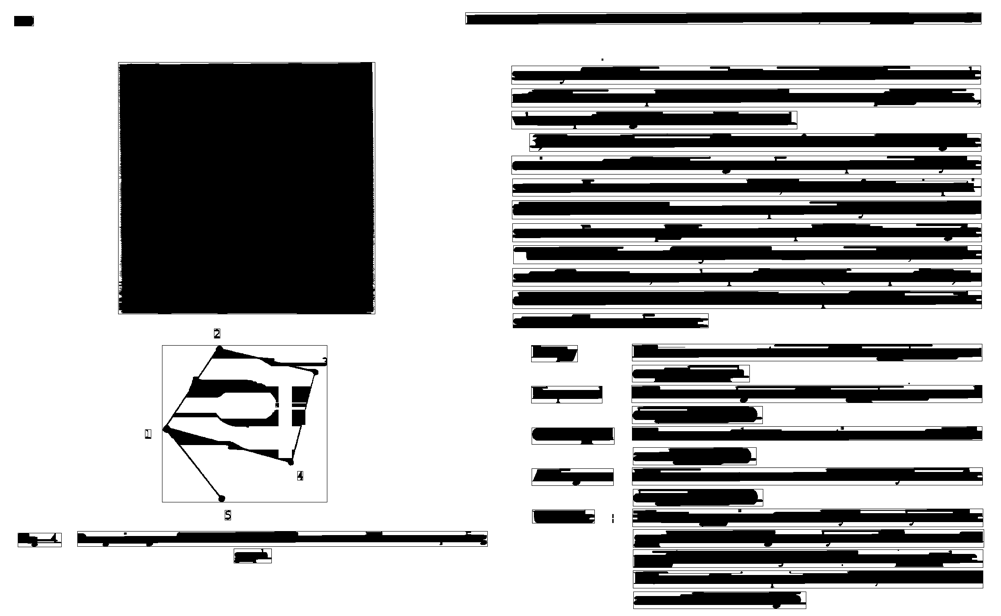


Figura 3. Retângulos envolvem cada componente conexa de “text.pbm”.

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.

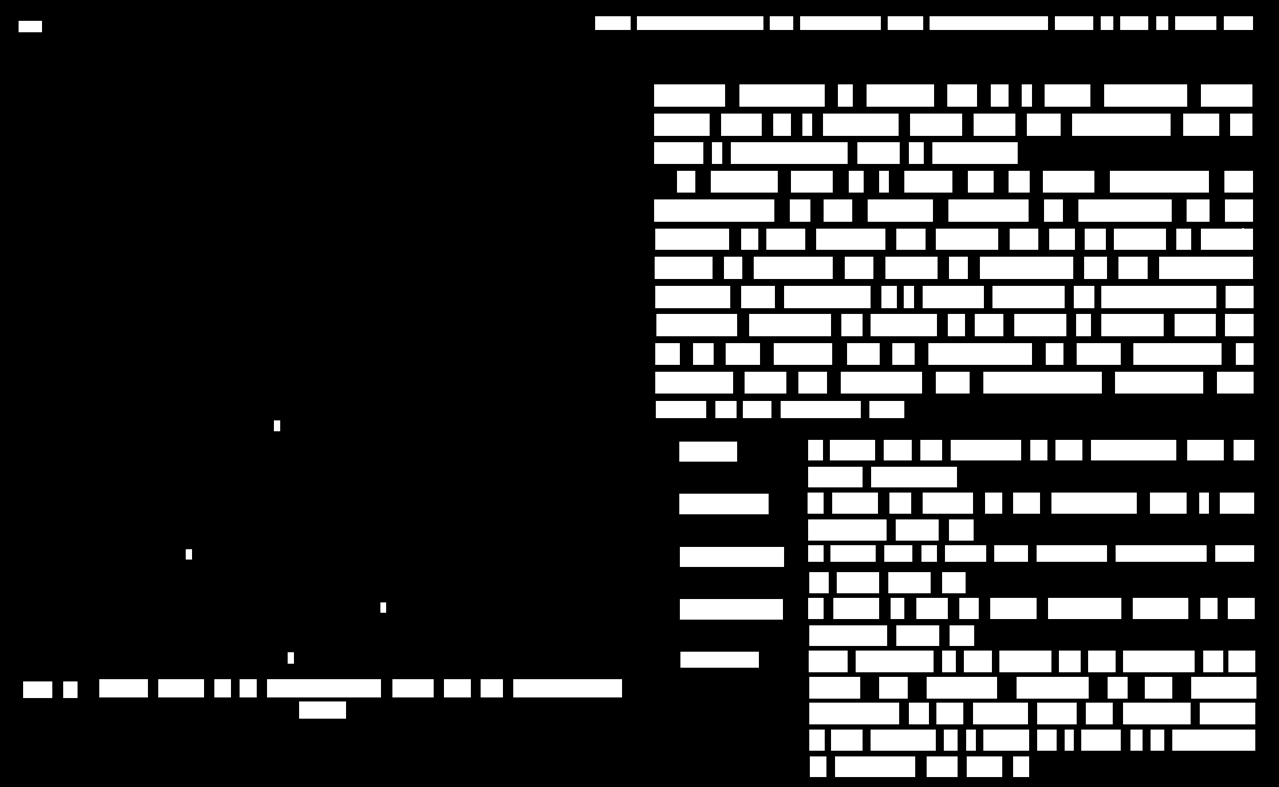


Figura 4. “text.pbm” após ser aplicado o fechamento em cada componente.

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.

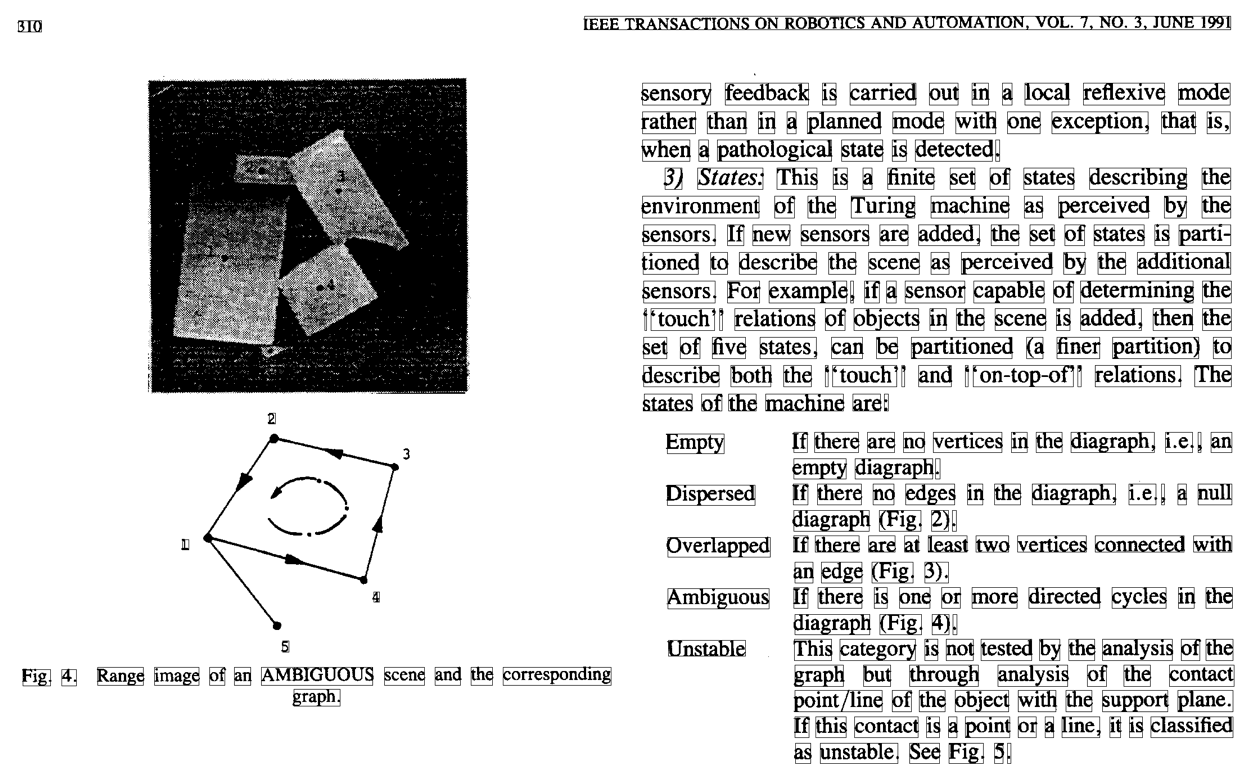


Figura 5. Resultado final de “text.pbm” usando o limiar no passo 10.

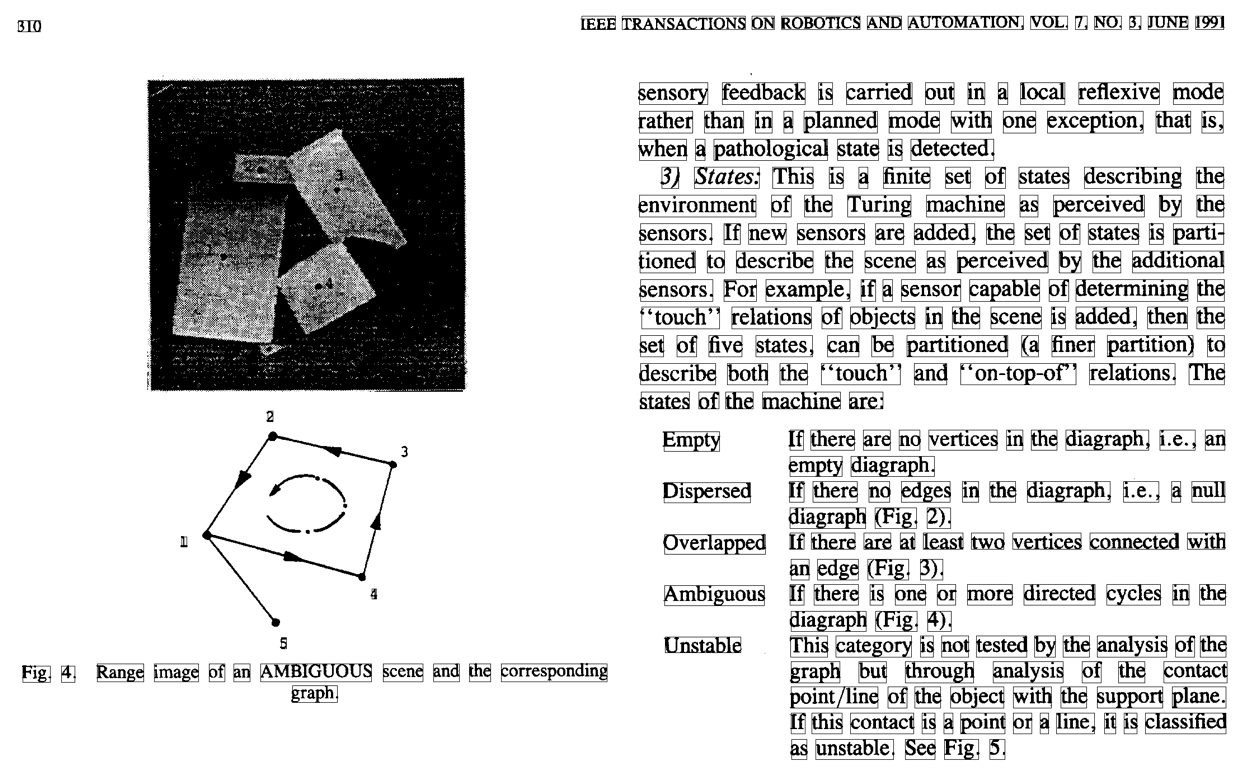


Figura 6. Resultado final de “text.pbm” usando o operador morfológico no passo 10.

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”.

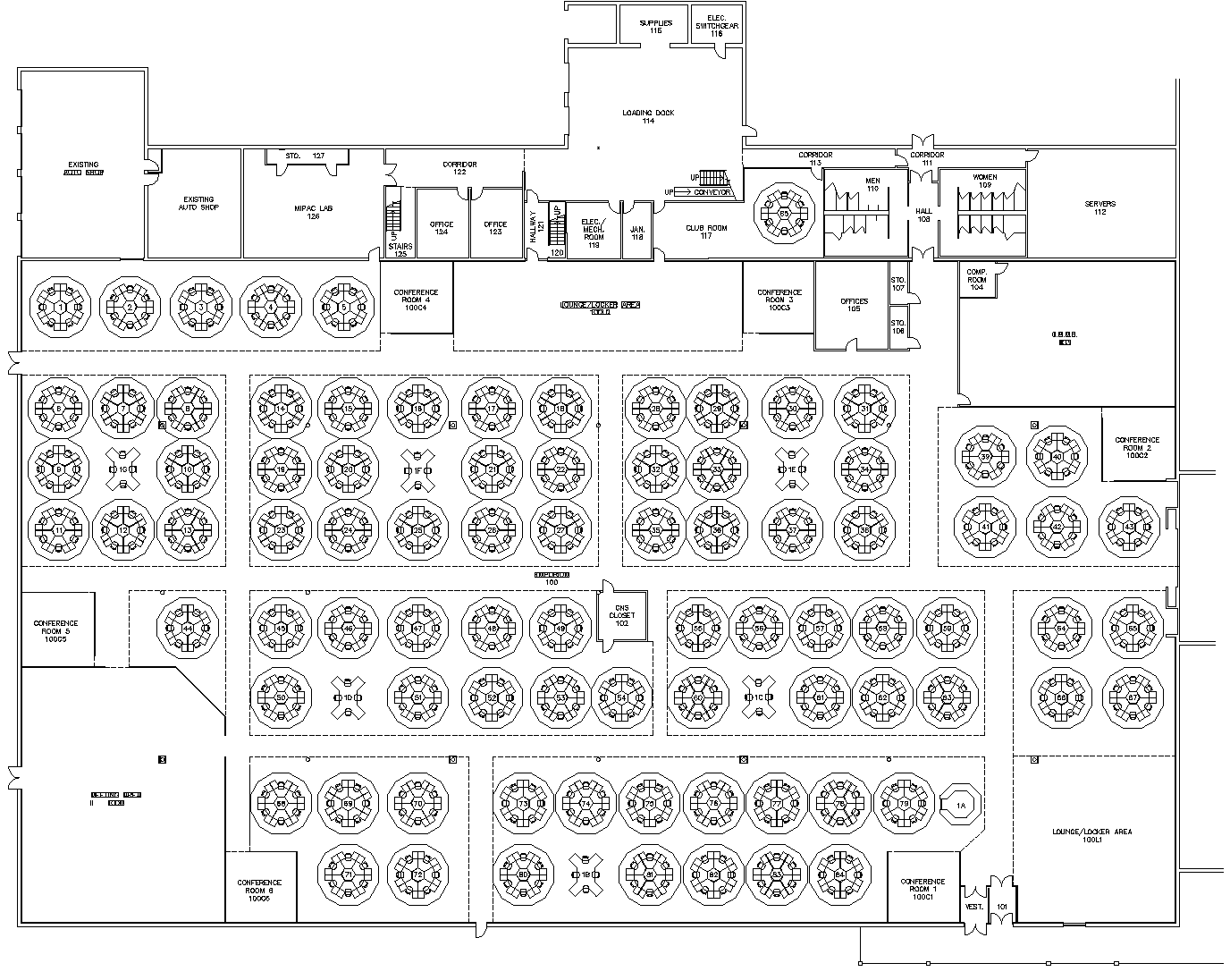


Figura 10. Imagem de saída para “map.pbm”.

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.

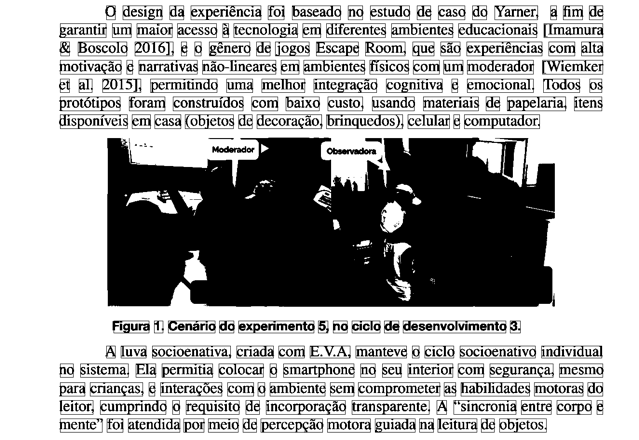
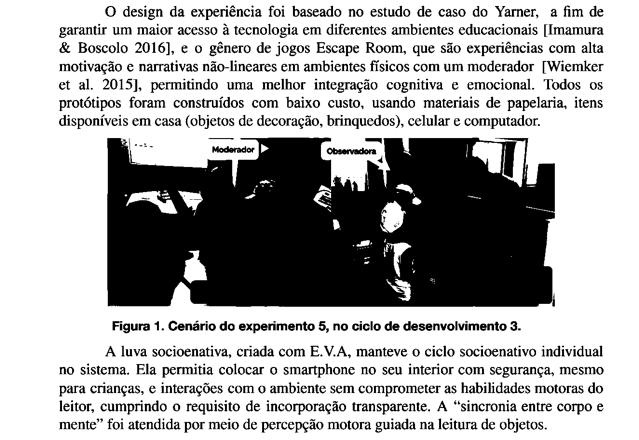


Figura 11. Entrada e saída para “paper.pbm”.

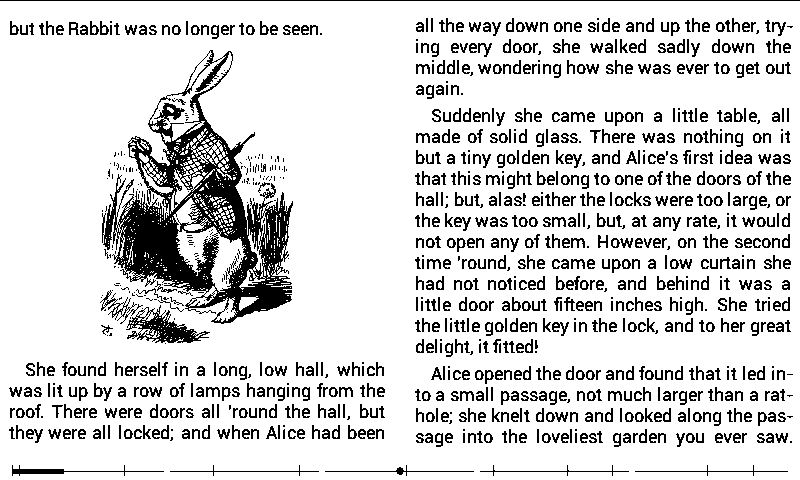
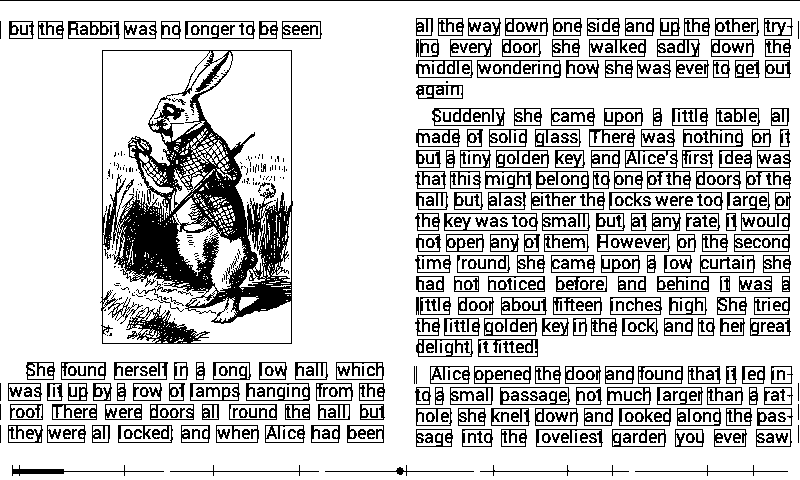
 

Figura 12. Entrada e saída da imagem “rabbit.pbm”.

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.

Aplicando a versão do algoritmo que usa o limiar nas imagens “paper.pbm” e “rabbit.pbm” a diferença é bem grande. Os resultados podem ser vistos na Tabela 1.

Tabela 1. Comparação do algoritmo de classificação com limiar e operador morfológico.

|  |  |  |  |
| --- | --- | --- | --- |
| Imagem | Linhas | Palavras (limiar) | Palavras (operador) |
| text.pbm | 40 | 246 | 242 |
| letter\_a.pbm | 1 | 1 | 1 |
| scs.pbm | 1 | 1 | 1 |
| map.pbm | 9 | 10 | 18 |
| paper.pbm | 14 | 98 | 159 |
| rabbit.pbm | 26 | 33 | 220 |

# 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. (2015) “OpenCV Documentation”. Disponível em: <https://docs.opencv.org/>. Acesso em: mai. 2019.

PGBA. (2011) “PBMA data directory”. Disponível em: < <https://people.sc.fsu.edu/~jburkardt/data/pbma/pbma.html>>. Consulta em mai. 2019.

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