



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

PreOCR - Trabalho de Processamento de Imagens T01 2023.2

Professora: Beatriz Trinchão Andrade

Grupo 13 - Everton Santos de Andrade Júnior



São Cristóvão – Sergipe

2024

Sumário

1	Introdução e Resultados	2
2	Resultados	4
3	Instruções & Observações sobre o Projeto	12
3.1	Requisitos	12
3.2	Observações	13
4	Métodos e Implementações	16
4.1	Parâmetros	16
4.2	Detalhes	17
4.2.1	Componentes Conectados	18
4.2.2	Detecção e Reconhecimento de Caracteres	19
4.2.3	Operações Morfológicas	19
4.2.3.1	Dilatação	19
4.2.3.2	Erosão	22
4.2.3.3	Filtro Mediana	23
4.2.3.4	Contagens	23
5	Conclusão	27
	Referências	28

1

Introdução e Resultados

Neste trabalho, desenvolvemos um programa capaz de processar imagens binárias no formato PBM ASCII (PGM tipo P1), contendo texto dos tipos e tamanhos de fontes, variando de 8 à 40. Conseguimos determinar o básico do trabalho que são o número de linhas e palavras no texto e os retângulos dessas palavras, gerando uma imagem de saída com esses retângulos. Além disso, nosso grupo implementou mais funcionalidades, como a detecção de colunas e blocos (ou parágrafos) do texto, utilizando o conceito de distância alinhada, incluindo a *bouding box* (bbox) desses blocos e linhas que indicam as colunas. Também foi feita a detecção dos blocos dos caracteres que formam as palavras encontradas. Após isso é feita uma análise de similaridades entre as imagens de letras padrões e as letras encontradas nessa detecção através de template matching (GONZALEZ; WOODS, 2008, 12.2). O template matching não gerou bons reconhecimentos, mesmo após testar diferentes variações como desconderar a igualdade de pixels pretos ou brancos.

Adicionalmente, para ilustrar nossos resultados de forma mais interativa, geramos uma série de imagens intermediárias que mostram o processo de detecção de blocos, colunas, palavras, e linhas destacando as regiões por um retângulo de diferentes cores. O formato dessas imagens é P3 (ascii RGB). Essas imagens foram usadas em sequência de frames para gerar um vídeo de saída auxiliando na depuração e compreensão do comportamento do algoritmo usando o programa de linha de comando “ffmpeg”.

O processamento envolve filtro mediano e, principalmente, dilatação. Também envolve os conceitos de conectividade para encontrar as palavras. A nossa abordagem é dinâmica, baseado na altura das palavras mudamos os parâmetros enquanto o programa roda. Desse modo é possível identificar estruturas de texto em diferentes alinhamentos, como justificado, esquerda, centro e direita, além de lidar com diferentes tamanhos e estilos de fonte, incluindo Comic Sans, Impact, Cascadia Code, Arial e Times New Roman. Um exemplo de vídeo gerado pode ser encontrado em <<https://youtu.be/uA45GeodGss?si=ZOgsA2av7EKZeG69>>.

Decidimos começar brevemente com os resultados ([Capítulo 2](#)). Em seguida no [Capítulo 3](#) com instruções e notas sobre o projeto o qual inclui os passos necessários para execução e as definições das imagens teste de entrada. No [Capítulo 4](#), damos detalhes sobre os algoritmos implementados. Por fim, o ?? discute sobre possíveis melhorias e palavras finais de conclusão.

2

Resultados

Os resultados desse projeto podem ser bem entendidos pelas imagens coloridas geradas, por isso, os próximos parágrafos discutem a saída gerada por este projeto para diferentes entradas. Na [Figura 2](#) temos o resultado gerado no projeto para a entrada de um texto na fonte Cascadia Code em negrito de tamanho 10, alinhado à esquerda. Essa imagem tem baixa resolução e vem com ruídos. Mesmo assim o nosso trabalho conseguiu identificar as bboxes das palavras, mesmo quando as letras possuem alturas diferentes. Os retângulos em vermelho indicam essas bboxes das palavras encontradas. A linha em amarelo é feita pelo algoritmo toda vez que encontra o começo de uma coluna. Em verde está o agrupamento dos blocos ou parágrafos que o nosso trabalho conseguiu encontrar. Especificamente para essa entrada, o algoritmo encontrou 5 blocos, 2 colunas, 42 linhas e 395 palavras. Com a mesma fonte mas de tamanho 16, e centralizado, temos outro resultado na [Figura 3](#), nesse encontramos 2 colunas, 4 blocos, 38 linhas e 226 palavras.

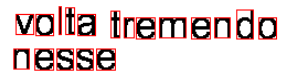
Na [Figura 4](#), já mudamos a fonte para Time News Roman, e ainda está em itálico. Nesse entrada já pulamos para quatro colunas e tamanho 18 de fonte. No resultado foi encontrado 4 colunas, 6 blocos, 38 linhas e 197 palavras. O correto deveria ser 196 palavras, o restante está correto. Pulamos para uma nova fonte chamada Impact de tamanho 40, que é um grande salto dos outros casos. Neste, o algoritmo encontra 2 colunas, 2 blocos, 18 linhas e palavras 47 palavras. O correto deveria ser 46 palavras, o restante está correto.

Ademais, foram realizados teste com fonte Arial, com exemplos tanto providos pela professor Beatriz, quanto criados pelo grupo. Um exemplo desses resultados pode ser ilustrado pela [Figura 7](#), nesse o texto é justificado com 3 colunas e tamanho 12. O algoritmo encontrou tudo corretamente, com 557 palavras e 52 linhas. A ?? inclui o resultado para um exemplo com letras de tamanho 8 e fonte Comic Sans, a menor testada no projeto. Por fim, demos upload da coleção de videos gerados por este trabalho com diferentes fontes e tamanhos testados que pode ser encontrado no youtube neste link: <https://www.youtube.com/watch?v=uA45GeodGss>.

Também houve a detecção de caracteres, os resultados te encontrar o retangulos dos

caracteres numa dada palavra funciona perfeitamente, até onde pudemos testar, pois letras são conectadas, então basta utilizar a mesma função de conectividade usada pra encontrar as palavras, mas sem fazer dilatação das imagens, pois não queremos juntas as letras e sim manter-as separadas. A ?? mostra um exemplo de bboxes encontrada para as letras num palavra encontrada.

Figura 1 – Exemplos de caracteres detectados para as palavras nesse, tremendo e volta.

A imagem mostra o texto "volta tremendo nesse" com as letras individualizadas por caixas vermelhas, demonstrando o resultado da detecção de caracteres.

Fonte: Autor.

Figura 2 – Cacadia Code em negrito, com 10 de tamanho, 2 colunas e 5 blocos de texto. A entrada possui baixa resolução e ruídos.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Quis aute iure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Av vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia

Sed ut perspiciatis unde omnis iste natus error sit voluptatem

Figura 3 – Cacadia Code centralizado, com 16 de tamanho, 2 colunas e 4 blocos de texto.

<p> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. </p> <p> Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem </p>	<p> porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? </p> <p> At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in </p>
---	--

Fonte: Autor.

Figura 4 – Times News Roman em itálico, com 18 de tamanho, com 4 colunas, 6 blocos de texto.

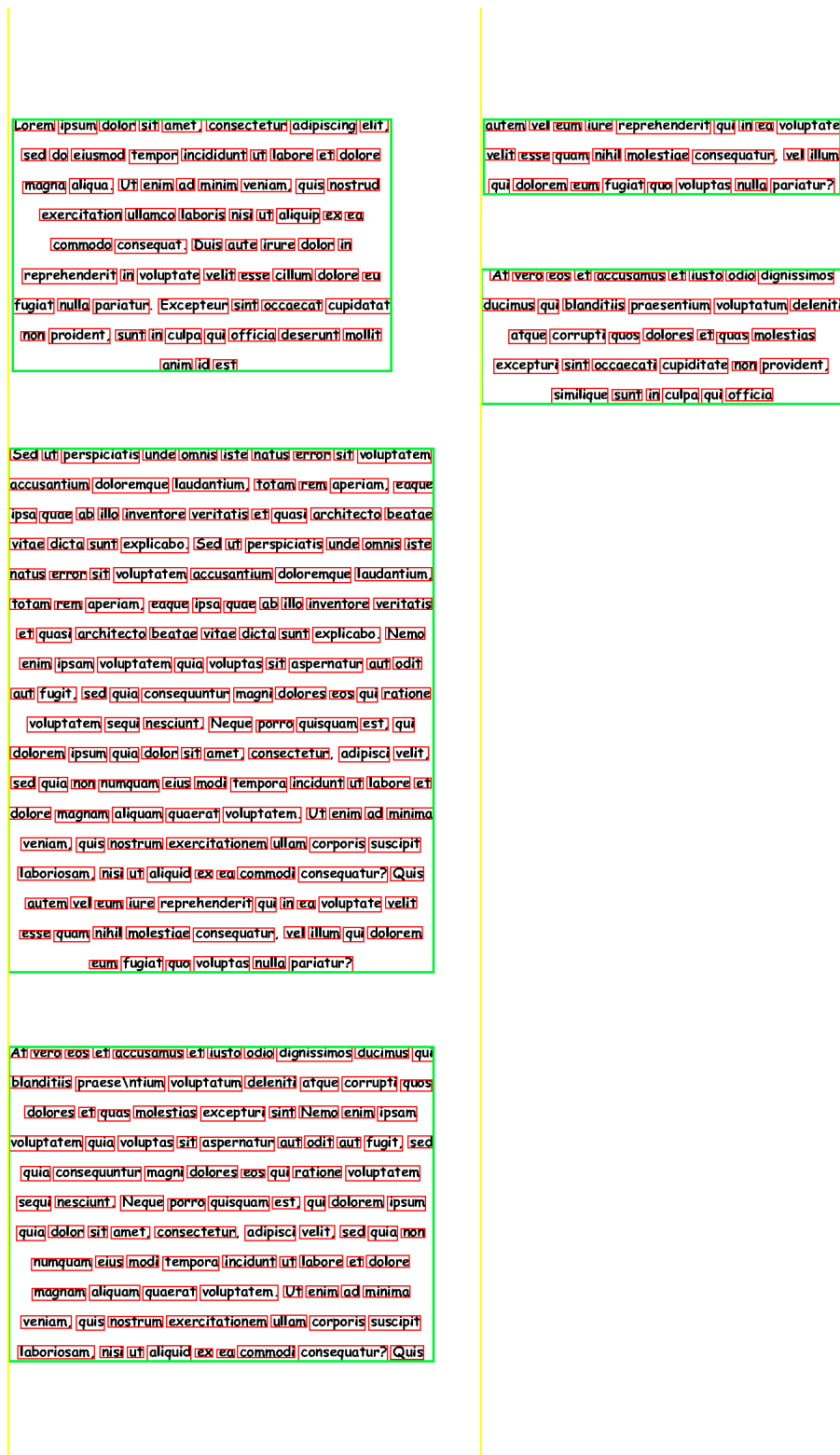
<p><i> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud</i></p> <p><i> Exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</i></p>	<p><i> Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi</i></p> <p><i> Beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro</i></p>	<p><i> quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitatione m ullam corporis</i></p>	<p><i> Suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?</i></p>
---	--	---	---

Figura 5 – Impact com tamanho 40, 2 colunas e 2 blocos de texto.



Fonte: Autor.

Figura 6 – Comic Sans centralizado, com 8 de tamanho, 2 colunas e 5 blocos de texto.



Fonte: Autor.

Figura 7 – Arial justificado, com 12 de tamanho, 3 colunas e 8 blocos de texto.

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>	<p>fugiat quo voluptas nulla pariatur?</p>	<p>Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>
<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum</p>	<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat.</p>	<p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?</p>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.</p>	<p>At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et</p>	

Fonte: Autor.

3

Instruções & Observações sobre o Projeto

O repositório do projeto pode ser encontrado através do link <https://github.com/evertonse/pim>. O método proposto está implementado em Python e pode ser executado a partir da pasta raiz do projeto. Primeiro, certifique-se de ter o repositório do projeto clonado ou baixado. Em seguida, na linha de comando, navegue até a pasta raiz do projeto e execute o seguinte comando, substituindo `caminho/para/imagem.pbm` pelo caminho da imagem que deseja processar:

```
$ python3 src/main.py caminho/para/imagem.pbm
```

Exemplo de caminho imagem pode ser:

```
python3 src/main.py assets/grupo_13_arial_esquerda_tamanho_16_colunas_2_blocos_4_
```

Se nenhuma imagem for fornecida, será usada uma imagem padrão. Imagens de exemplo adicionais geradas pelo nosso grupo podem ser encontradas na pasta "assets". Essas imagens seguem o padrão adaptado do documento PreOCR, pois precisamos identificar mais detalhes sobre a imagem. Seja F , T , C , B , L e P variáveis que representam informações sobre a fonte (tipo) e se é justificado, centralizado, etc., tamanho, número de colunas, número de linhas, número de blocos e número de palavras, respectivamente. Assim, o nome das imagens de teste segue o padrão `grupo_13_F_tamanho_T_colunas_C_blocos_B_linhas_L_palavras_P`. Um exemplo é:

- `grupo_13_arial_esquerda_tamanho_16_colunas_2_blocos_4_linhas_39_palavras_318`.

3.1 Requisitos

- É necessário ter o Python instalado para executar o script. A versão testada foi 3.10 e 3.11, mas possivelmente pega em versões superiores à 3.4

- Opcionalmente se a ferramenta `ffmpeg` for instalada e esteja disponível no caminho do sistema o projeto irá gerar vídeos. No Ubuntu, `ffmpeg` pode ser instalado usando `sudo apt install ffmpeg` ou `sudo snap install ffmpeg`.
- A biblioteca `numpy` é necessária para manipulação eficiente das matrizes (listas de pythons são absurdamente lentas). Foi instalado apenas pela estrutura de dados. As operações usadas do `numpy` são bem simples: multiplicação de matriz por escalar; tirar o máximo ou mínimo de matrizes; soma e subtração de matrizes. Pode ser instalada via `pip install numpy`. Se o `pip` não estiver instalado, ele pode ser obtido executando `sudo apt install python3-pip` no Ubuntu.

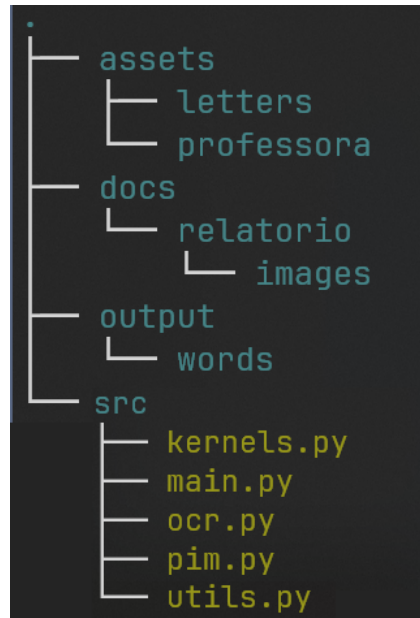
3.2 Observações

Todos os arquivos gerados durante a execução do projeto são armazenados na pasta "output/", com exceção do arquivo principal que contém os retângulos das palavras em vermelho e do arquivo de vídeo. Esses dois arquivos são prefixados com "group_13" e estão localizados no diretório atual. O site convertio.co foi usado para converter arquivos PDF para o formato PBM para arquivos de entrada adicionais. As pastas e seus conteúdos, ilustrados na [Figura 8](#), estão organizados da seguinte forma:

- **src/**: diretório que contém o código-fonte do projeto.
- **src/kernels.py**: definição dos elementos estruturante.
- **src/main.py**: arquivo principal do projeto contendo o código de execução principal.
- **src/ocr.py**: implementação da detecção e da tentativa de reconhecimento óptico de caracteres.
- **src/pim.py**: implementação das funções de processamento de imagens usadas: dilatação, erosão, filtro mediana.
- **src/utils.py**: contém funções de utilidade como ler imagem, escrever imagem, fazer resize de imagem.
- **assets/**: contém as imagens .pbm criados pelo grupo.
- **assets/letters/**: contém os modelos de letras utilizados na tentativa de reconhecimento de caracteres.
- **assets/professora/**: contém as imagens .pbm disponibilizados pela professora, mas com os nomes adaptados.

- **output/**: local onde são armazenados os arquivos gerados durante a execução do projeto, sendo apenas os resultados intermediários.

Figura 8 – Estrutura de pastas do projeto.



Fonte: Autor.

No exemplo de execução do projeto apresentado no [Código 1](#), são fornecidas diversas informações úteis durante a execução do comando. Essas informações incluem o tempo decorrido para cada etapa do processamento, como a leitura do arquivo de imagem, aplicação de filtros, aplicação da dilatação, detecção de componentes conectados, criação do vídeo resultante, entre outros. Além disso, o nome do vídeo salvo e o nome do arquivo final gerado também são exibidos durante a execução do comando. O resultado final da execução, que inclui o número total de palavras, linhas, colunas e blocos detectados na imagem processada, é apresentado na última linha, prefixado com o caractere ">".

Código 1 – Rodando o projeto na linha de comando.

```
1  USAGE: python3 src/main.py [path/to/image.pbm]
2
3  INFO: using default image path
4      './assets/grupo_13_cascadia_code_bold_esquerda_noisy
5  INFO: read_ppm_file took 0.0697s to execute.
6  INFO: Image is 795x1124 (width x height).
7  INFO: fast_median_blur took 1.0870s to execute.
8  INFO: fast_dilate2 took 0.0114s to execute.
9  INFO: fast_dilate2 took 0.0142s to execute.
10 INFO: fast_dilate2 took 0.0073s to execute.
11 INFO: closing took 0.0171s to execute.
12 INFO: find_connected_components_bboxes took 0.7458s to execute.
13 INFO: choose_best_height took 0.0001s to execute.
14 INFO: median height found is 11 pixels.
15 INFO: group_bboxes took 1.6317s to execute.
16 INFO: wrote video output to 'group_13_video.mp4'.
17 INFO: create_video_from_images took 2.9656s to execute.
18 INFO: wrote final output to
19     './group_13_detected_colunas_2_blocos_5_linhas_42_palavras_395.ppm'.
20 > 395 words, 42 lines, 2 columns, 5 blocks.
```


4

Métodos e Implementações

Neste trabalho, aplicamos técnicas de processamento de imagem para detectar e analisar palavras, linhas, colunas e blocos em documentos de texto digitalizados. Para alcançar esse objetivo, utilizamos operações de dilatação e fechamento, juntamente com a detecção de componentes conectados, o parâmetros são especificados na [seção 4.1](#). Criamos elementos estruturantes personalizados para realizar operações específicas, como a detecção de palavras e a segmentação de blocos de texto. Para remoção de ruídos usamos o filtro mediana. Há uma dilatação condicional da imagem, se necessário, com base na altura média dos componentes conectados (palavras) já filtrados.

4.1 Parâmetros

Nesta seção, apresentamos os parâmetros utilizados nos algoritmos, incluindo os elementos estruturantes de dilatação e fechamento, com diferentes tamanhos e formatos. Além disso, destacamos variáveis de limiar que são ajustadas para alterar o comportamento do algoritmo, adaptando-o às características da imagem sendo processada.

1. Filtro Mediana: 3×3
2. Primeira Dilatação: elemento estruturante horizontal truncada 5×5

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3. Fechamento: elemento estruturante para dilatação horizontal 3x3

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

4. Dilatação Condicional: elemento estruturante para dilatação horizontal 5x5

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5. Limiar para ocorrer dilatação condicional: altura mediana deve ser superior à 31 pixels.
6. Calculo da quantidade de iterações para dilatação condicional: $\text{round}(1 + \text{altura_mediana} / 31)$
7. Componentes Conectados: consideramos 8-connectividade, isto é, as diagonais são consideradas conectadas.

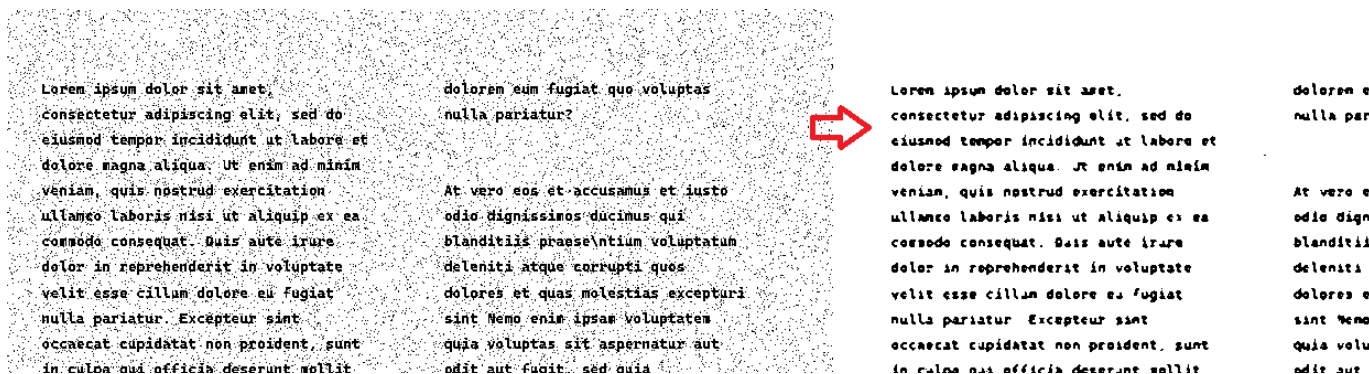
4.2 Detalhes

O algoritmo inicia lendo a imagem de entrada e realizando pré-processamento, o qual inclui a aplicação do filtro da mediana (chamado de `median_blur` no código) e a inversão das cores. Essas etapas têm como objetivo remover possíveis ruídos, como o ruído do tipo sal e pimenta, como ilustrado na [Figura 9](#). Como os ruídos são confirmados ter tamanho máximo de 1 pixel, um filtro 3×3 é suficiente. A inversão da imagem é realizada para facilitar a aplicação de algoritmos morfológicos, como a dilatação.

A dilatação é aplicada para melhorar a conectividade dos componentes de texto (palavras) na imagem. Dependendo dos parâmetros especificados, o algoritmo pode realizar operações adicionais, como fechamento (`closing`) ou abertura (`opening`), para refinar ainda mais as regiões de texto.

A abordagem do algoritmo envolve o uso de um elemento estruturante, conhecido como SE (*Structuring Element*), para conectar as letras. Isso pode ser alcançado com um SE de formato horizontal. Embora tenham sido testados diferentes tipos de SE, como formatos circular, de cruz e vertical, o SE horizontal mostrou-se mais eficaz. Essa escolha é justificada pelo fato de que as letras normalmente estão alinhadas horizontalmente, exigindo uma expansão na direção horizontal para conectá-las, como ilustrado na [Figura 10](#).

Figura 9 – Remoção de ruído.



Fonte: Autor.

Figura 10 – Resultado da aplicação de 2 iterações de dilatação com SE horizontal 5x5.



Fonte: Autor.

4.2.1 Componentes Conectados

As operações de dilatação horizontal notavelmente aprimoraram a conectividade entre caracteres adjacentes na imagem. Esses conjuntos conectados de pixels representam as palavras individuais dentro da imagem. O conceito de componentes conectados é fundamentado em grafos e pode ser aplicado a imagens binárias, onde os pixels brancos correspondem aos nós. Existem diferentes tipos de conectividade. Se considerarmos apenas os vizinhos de um pixel como à esquerda, à direita, acima e abaixo, temos o que chamamos de 4-conectividade (GONZALEZ; WOODS, 2008, 2.5.2 Adjacency, Connectivity, Regions, and Boundaries). Ao adicionar as diagonais, obtemos a 8-conectividade.

O algoritmo descrito no Código 2 identifica esses componentes conectados e se baseia em uma busca em profundidade, marcando os pixels visitados. A busca continua até que não haja mais caminhos a serem explorados na imagem, considerando também as diagonais, se *connectivity* for igual 8). Quando não há mais caminhos a serem percorridos, o conjunto de pixels visitados forma um componente conectado. Esse processo é repetido até que todos os componentes tenham sido identificados, atualizando os pontos mais à esquerda inferior e mais à direita superior do componente, formando a *bounding box* da palavra.

Em seguida, um retângulo é desenhado ao redor dessas palavras na imagem de saída. Opcionalmente, as palavras isoladas podem ser escritas em uma nova imagem, para uso

posterior em tentativas de reconhecimento óptico de caracteres (OCR). Após a identificação dos componentes de texto, é realizada uma filtragem para eliminar as *bounding boxes* correspondentes à pontuação (vírgulas, pontos) e manter apenas aquelas associadas às palavras.

O [Código 3](#) ilustra como as palavras são então agrupadas em blocos com base em sua proximidade. A distância considerada é a menor diferença possível entre as duas *bounding boxes*; se estiverem sobrepostas, então a distância é zero. Finalmente, a função `group_bboxes` retorna a lista resultante, que contém todos os retângulos agrupados com base na distância máxima especificada; se a distância for maior, é considerado parte de outro grupo. Neste trabalho, essa distância máxima é encontrada de maneira dinâmica, executando o algoritmo uma vez para encontrar uma certa altura das palavras e, em seguida, executando novamente com essa informação adquirida. Para visualizar de forma interativa as etapas do algoritmo na descoberta desses blocos, é gerado um vídeo.

4.2.2 Detecção e Reconhecimento de Caracteres

O processo de reconhecimento de caracteres é composto por três etapas principais: a detecção dos caracteres na imagem, a correspondência desses caracteres com os modelos predefinidos e o reconhecimento. Durante essa sequência, as *bounding boxes* são geradas para cada caractere detectado. O [Código 4](#) mostra esse processo. Nele é carregado modelos de letras predefinidos no diretório `./assets/letters/`. Esses modelos são imagens binárias de letras individuais, como "v", "o", "l" e "t", com cada letra invertida para garantir que os pixels de interesse (brancos) tenham o valor 255 e os pixels de fundo (pretos) tenham o valor 0. A imagem a ser comparada é a subimagem que contém o caractere detectado; em seguida, as *bounding boxes* são ordenadas da esquerda para a direita, seguindo a ordem de leitura.

Este recorte da letra é redimensionado para ter o mesmo tamanho das letras de referência. Para cada letra de referência, calcula-se a similaridade entre a letra recortada e o modelo. Essa similaridade é medida contando o número de pixels brancos iguais, na mesmas coordenadas nas duas imagens. A letra com a maior similaridade é considerada a melhor correspondência para a região.

4.2.3 Operações Morfológicas

4.2.3.1 Dilatação

A dilatação morfológica é útil para preencher pequenos espaços entre objetos. No [Código 5](#), a função `understandable_dilate` implementa a dilatação de forma "compreensível", mais compatível com a definição dada em [Gonzalez e Woods \(2008, 9.2.2 Dilation\)](#).

Nessa implementação a variável `kernel` representa o elemento estruturante, que define a forma e o tamanho da dilatação. Durante o processo, a imagem é percorrida pixel a pixel, e para cada pixel, verifica-se se pelo menos um dos pixels na vizinhança definida pelo kernel é branco

Código 2 – .

```

1 def find_connected_components_bboxes(image, min_area=0,
  connectivity=8):
2     nbrs = [(1, 0), (-1, 0), (0, 1), (0, -1)]
3     if connectivity == 8:
4         nbrs.extend([(1, 1), (1, -1), (-1, 1), (-1, -1)])
5
6     def dfs(y, x):
7         nonlocal nbrs, image
8         min_y, min_x, max_x, max_y = y, x, y, x
9         stack = [(y, x)]
10        while stack != list():
11            cy, cx = stack.pop()
12            if (
13                0 <= cy < image.shape[0]
14                and 0 <= cx < image.shape[1]
15                and not visited[cy, cx]
16                and image[cy, cx] == 255
17            ):
18                visited[cy, cx] = True
19                min_y = min(min_y, cy)
20                min_x = min(min_x, cx)
21                max_x = max(max_x, cy)
22                max_y = max(max_y, cx)
23
24                for dy, dx in nbrs:
25                    stack.append((cy + dy, cx + dx))
26        return min_y, min_x, max_x, max_y
27
28    visited = np.zeros(image.shape, dtype=bool)
29    bounding_boxes = list()
30
31    for y in range(image.shape[0]):
32        for x in range(image.shape[1]):
33            if not visited[y, x] and image[y, x] == 255:
34                min_y, min_x, max_x, max_y = dfs(y, x)
35                bounding_boxes.append((min_y, min_x, max_x, max_y))
36
37    return bounding_boxes

```

(valor 255). Se pelo menos um dos pixels na vizinhança é branco, o pixel central é definido como branco (255) na imagem resultante. Caso contrário, o pixel central é definido como preto (0).

Uma interpretação útil é que o elemento estruturante define qual é a vizinhança, onde tiver 1 no elemento indica que faz parte da vizinhança, se tiver zero então não faz parte. Para cada x,y percorridos a imagem resultado é o máximo o valor de sua vizinhança definido pelo elemento estruturante.

No [Código 6](#), a função `fast_dilate2` implementa uma dilatação morfológica mais

Código 3 – .

```

1 def group_bboxes(bboxes, max_distance, image) -> list[list]:
2     """
3     Group bboxes based on a max distance.
4     'image' is video for purposes
5     """
6     result = list()
7     rest_idx = [i for i in range(len(bboxes))]
8     while len(rest_idx) > 0:
9         close_idx = [rest_idx.pop()]
10        bbox = bboxes[close_idx[0]]
11        counter = 0
12        while counter < len(rest_idx):
13            r_idx = rest_idx[counter]
14            counter += 1
15            dist = distance(bbox, bboxes[r_idx])
16            if (dist < max_distance) or bbox_overlap(bbox,
17                bboxes[r_idx]):
18                close_idx.append(r_idx)
19                rest_idx.remove(r_idx)
20                bbox = enclosing_bbox([bbox, bboxes[r_idx]])
21                counter = 0
22
23            vid_img = image.copy()
24            y, x, y2, x2 = bbox
25            rectangle(vid_img, (y, x), (y2, x2), (0, 244, 55), 2)
26            video_frames.append(vid_img)
27
28            y, x, y2, x2 = bbox
29            rectangle(image, (y, x), (y2, x2), (0, 244, 55), 2)
30            result.append([bboxes[i] for i in close_idx])
31    return result

```

rápida, aproveitando as operações de matriz do NumPy. A imagem é expandida com um preenchimento adequado para evitar problemas de borda durante a aplicação (o elemento estruturante é chamado de *kernel* no código). Em seguida, usando a função `np.maximum`, calculamos o valor máximo elemento a elemento da matriz entre a imagem resultante e o subconjunto da imagem com padding.

Inicialmente, a imagem resultante é preenchida com entradas zeradas e, à medida que iteramos sobre a altura j e a largura i do elemento estruturante, aplicamos essa função `maximum` em um **subconjunto diferente** da imagem com *padding*, dependendo de j e i atual. Isso é feito apenas se o elemento estruturante for maior ou igual a 1 na altura j e largura i . Isso efetivamente realiza uma dilatação, onde o pixel central de uma vizinhança é definido como o valor máximo dessa vizinhança. No entanto, observe que a vizinhança depende do elemento estruturante, apenas onde os valores são 1 são considerados vizinhança.

Código 4 – Função para reconhecimento de caracteres.

```

1  # Pacote OCR
2  def string(img, letters):
3      print(img.shape)
4      orig = pim.convert_to_rgb(img)
5      img = pim.invert(img)
6      bbxs = pim.find_connected_components_bboxes(img, connectivity=8)
7      # Start from leftmost
8      bbxs = sorted(bbxes, key=lambda b: b[1])
9
10     letter_sequence = []
11     for b in bbxs:
12         y, x, y2, x2 = b
13         best_sim = 0
14         best_letter = "_"
15         for l, limg in letters.items():
16             resized = pim.resize(img[y:y2, x:x2], *limg.shape)
17             sim = 0
18             for j in range(limg.shape[0]):
19                 for i in range(limg.shape[1]):
20                     if resized[j, i] > 0:
21                         if limg[j, i] > 0:
22                             sim += 1
23             if sim > best_sim:
24                 best_sim = sim
25                 best_letter = l
26         letter_sequence.append(best_letter)
27
28     return "".join(letter_sequence), orig

```

Ambas as implementações alcançam o mesmo resultado de dilatação morfológica, mas `fast_dilate2` é mais eficiente devido ao uso de operações vetorizadas do NumPy. Isso resulta em uma execução mais rápida, especialmente em imagens grandes.

4.2.3.2 Erosão

A erosão segue o mesmo princípio da dilatação, mas com a operação sendo o mínimo no lugar do máximo. A iteração começa com a imagem resultante tendo todas as entradas definidas como 1 em vez de 0. Na versão `understandable` da erosão, apenas quando **todos os pixels** da imagem se alinham com o elemento estruturante é que o pixel central é definido como 1 no lugar de **pelo menos um**. Vale ressaltar que a erosão não é necessária para o funcionamento do projeto, pois mesmo sem `closing` ou `opening`, que usam implicitamente a erosão, o algoritmo é robusto para os casos de teste mencionados na introdução. A função `opening` é a operação de abertura, que consiste em primeiro realizar uma erosão seguida de uma dilatação. É útil para remover pequenos ruídos e separar objetos próximos, mas neste trabalho, removemos esses ruídos através

Código 5 – .

```
1 def understandable_dilate(image, kernel):
2     result = np.zeros(image.shape)
3     height = image.shape[0]
4     width = image.shape[1]
5     kernel_height = kernel.shape[1]
6     kernel_width = kernel.shape[0]
7     kernel_width_delta = kernel_width // 2
8     kernel_height_delta = kernel_height // 2
9     for y in range(height):
10        for x in range(width):
11            all_good = False
12            for j in range(kernel_height):
13                for i in range(kernel_width):
14                    i_offset = i - kernel_width_delta
15                    j_offset = j - kernel_height_delta
16                    color = 0
17                    if (
18                        (x + i_offset) >= 0
19                        and (x + i_offset) < width
20                        and y + j_offset >= 0
21                        and y + j_offset < height
22                    ):
23                        color = image[y + j_offset, x + i_offset]
24                        kcolor = kernel[j, i]
25                        if int(kcolor) * int(color):
26                            all_good = True
27                            break
28                    if all_good:
29                        break
30            if all_good:
31                result[y, x] = 255
32            else:
33                result[y, x] = 0
34    return result
```

do filtro da mediana.

4.2.3.3 Filtro Mediana

O [Código 7](#) implementa um filtro de mediana. Ele percorre cada pixel da imagem e substitui o valor do pixel pela mediana dos valores dos pixels em sua vizinhança, definida pelo tamanho do filtro. Isso ajuda a remover ruídos e detalhes pequenos.

4.2.3.4 Contagens

A contagem de linhas, proposto no [Código 8](#), verifica se a lista de caixas delimitadoras está vazia, depois as *bounding boxes* ordenadas verticalmente com base na coordenada y. Para

Código 6 – .

```
1 def fast_dilate2(image, kernel):
2     global counter
3     height, width = image.shape
4     kernel_height, kernel_width = kernel.shape
5
6     kernel_width_delta = kernel_width // 2
7     kernel_height_delta = kernel_height // 2
8
9     # We pad by the kernel delta, top, bottom, left and right
10    padded_image = pad(
11        image,
12        kernel_height_delta,
13        kernel_height_delta,
14        kernel_width_delta,
15        kernel_width_delta,
16    )
17
18    dilated = np.zeros(image.shape, dtype=np.uint8)
19    for j in range(kernel_height):
20        for i in range(kernel_width):
21            if kernel[j, i] == 1:
22                shifted_sub_image = padded_image[j : j + height, i :
23                    i + width]
24                dilated = np.maximum(
25                    dilated, shifted_sub_image
26                )
27    return dilated
```

cada *bounding box*, calcula-se a sobreposição vertical com a anterior. Se a sobreposição for menor que uma fração mínima, contamos uma nova linha.

O algoritmo para contar colunas (Código 9) é semelhante ao de linhas, mas agora classificamos os retângulo horizontalmente em vez de verticalmente. Para cada retângulo ordenado, verificamos se sua coordenada x do canto inferior esquerdo é maior do que a coordenada x do canto inferior direito da caixa mais à direita encontrada até o momento (*max_right*). Se for, consideramos isso como o início de uma nova coluna. Além disso, desenhamos uma linha vertical indicando o início da nova coluna na imagem de vídeo.

Código 7 – Implementação do filtro de mediana.

```
1 def understandable_median_blur(image, filter_size=3):
2     assert len(image.shape) == 2
3
4     result = image.copy()
5     height = image.shape[0]
6     width = image.shape[1]
7     for y in range(height):
8         for x in range(width):
9             colors = list()
10            for j in range(filter_size):
11                for i in range(filter_size):
12                    i_offset = i - filter_size // 2
13                    j_offset = j - filter_size // 2
14
15                    color = 0
16                    if (
17                        (x + i_offset) >= 0
18                        and (x + i_offset) < width
19                        and y + j_offset >= 0
20                        and y + j_offset < height
21                    ):
22                        color = image[y + j_offset, x + i_offset]
23                    colors.append(color)
24                colors.sort()
25                result[y, x] = colors[(len(colors) // 2)]
26    return result
```

Código 8 – Contagem de linhas.

```
1 def count_lines(bboxes, orig):
2     if len(bboxes) == 0:
3         return 0
4     bboxes = sorted(bboxes, key=lambda bbox: (bbox[0], bbox[2]))
5
6     lines = 1
7     prev1, _, prev2, _ = bboxes[0]
8     xdiffs_bboxes = list()
9     for bbox in bboxes[1:]:
10        y1, _, y2, _ = bbox
11        overlap = max(0, min(prev2, y2) - max(prev1, y1))
12        if overlap > abs(prev1 - prev2) / 100:
13            continue
14        lines += 1
15        prev1 = y1
16        prev2 = y2
17
18    return lines
```

Código 9 – Contagem de colunas.

```
1 def count_columns(bboxes, orig):
2     # Sort by left x-coordinate
3     sorted_bboxes = sorted(bboxes, key=lambda bbox: bbox[1])
4
5     num_columns = 0
6     max_right = float("-inf")
7
8     for bbox in sorted_bboxes:
9         _, left, _, right = bbox
10
11         # If the left x-coordinate of the bounding box is greater
12         # than the current rightmost x-coordinate,
13         # it indicates the start of a new column
14         if left > max_right:
15             num_columns += 1
16
17         # Update current rightmost x-coordinate
18         if max_right < right:
19             max_right = right
20     return num_columns
```

5

Conclusão

Com base nos resultados obtidos, é possível afirmar que o desempenho do nosso algoritmo foi bastante satisfatório. Ele se mostrou eficaz para diversas fontes e tamanhos de texto, independentemente de estarem justificados, centralizados, alinhados à direita ou à esquerda. Além disso, funcionou igualmente bem com imagens de resoluções variadas, tanto baixas quanto altas, demonstrando uma boa robustez. A utilização do filtro da mediana mostrou-se suficiente para lidar com uma quantidade considerável de ruído na imagem. A estratégia de realizar a dilatação horizontal foi uma sacada importante que contribuiu significativamente para a melhoria da conectividade entre os caracteres.

A abordagem de utilizar alturas encontradas como heurística para decidir se novas operações morfológicas deveriam ser aplicadas também se mostrou eficaz, ampliando a capacidade do algoritmo de lidar com diferentes casos. No entanto, há espaço para explorar ainda mais heurísticas, como usar a distância média entre caracteres como limiar para decidir sobre a aplicação de operações de fechamento, ou estudar o espaço em branco entre as palavras. Uma outra possibilidade seria considerar uma porcentagem do tamanho das letras em relação à imagem inteira para tomar decisões.

Embora a detecção de caracteres tenha funcionado bem, o método de *template matching* não teve o mesmo desempenho. Esse aspecto merece uma investigação mais aprofundada para entender por que não foi eficaz para reconhecer os caracteres (deu muitas similaridades erradas). No entanto, mesmo com esse desafio, o projeto como um todo foi capaz de realizar uma série de tarefas interessantes e diversas, incluindo a criação de um vídeo para visualização das etapas do algoritmo e a implementação de algoritmos vetorizados, proporcionando um entendimento mais profundo e intuitivo desses processos morfológicos.

Referências

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. Upper Saddle River, NJ: Pearson Prentice Hall, 2008. Citado 3 vezes nas páginas 2, 18 e 19.