Relatório Trabalho 3

MC920 – Introdução ao Processamento de Imagem Digital

Julianny Favinha Donda RA 156059

1. Solução

1.1. Projeção horizontal

Para o algoritmo de projeção horizontal, a imagem foi transformada para a escala de cinza e foi feita uma binarização da imagem dado um limiar (técnica de limiarização global). A imagem foi negativada para que os pixels pretos sejam os pixels do texto. Então, para cada ângulo, de 0° a 180°, a imagem foi rotacionada e foi calculada a projeção horizontal e a função objetivo para essa projeção, da seguinte maneira:

```
projecao_horizontal = quantidade de pixels pretos para cada linha da
imagem rotacionada

funcao_objetivo = soma dos quadrados das diferenças dos valores em
células adjacentes da projecao_horizontal
```

Com a função objetivo, podemos calcular linhas que tem valor muito diferente na quantidade de pixels pretos. Isso indica que, tendo uma linha branca, esta faz parte das linhas entre as linhas de texto. Logo, o texto estará alinhado horizontalmente.

Após esse processo, temos que o maior valor retornado pela função objetivo é o ângulo onde a imagem ficou alinhada horizontalmente. O ângulo final de rotação foi escolhido de acordo com os diagramas a seguir.

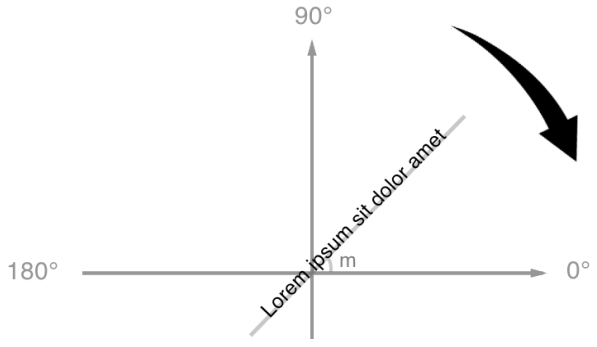


Figura 1 - Diagrama livre da rotação para texto com uma inclinação m entre 0° e 90°.

Se o algoritmo encontrou um ângulo m entre 0° e 90° onde a função objetivo da projeção horizontal é máxima, então deve “recuar” para ficar alinhado horizontalmente, ou seja, deve fazer uma rotação m graus negativos, dada a orientação de rotação, indicada pela seta.

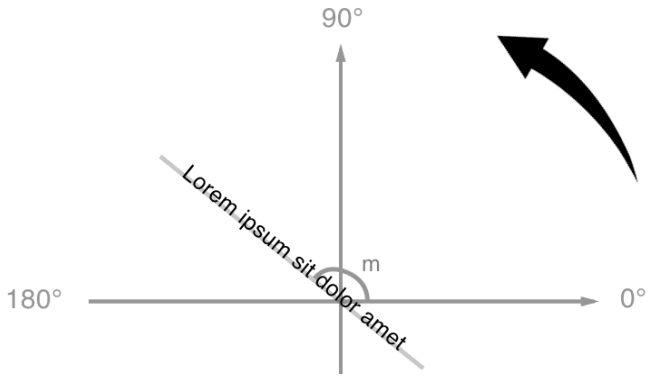
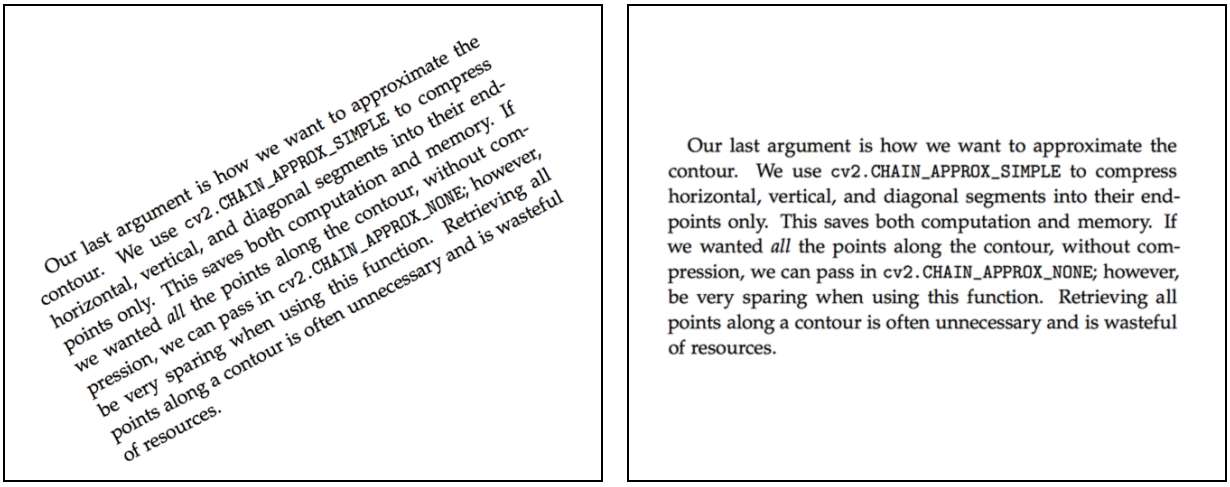


Figura 2 - Diagrama livre da rotação para texto com uma inclinação m entre 90° e 180°.

Por outro lado, se o algoritmo encontrou um ângulo m entre 90° e 180° que maximiza a função objetivo, então deve “avançar” para ficar alinhado horizontalmente, ou seja, deve fazer uma rotação de (m - 180) graus, dada a orientação de rotação, indicada pela seta. Isso evita que textos fiquem de ponta-cabeça.

Ao executar o programa `align.py` para a imagem `neg_28.png`, foi obtido o resultado a seguir.

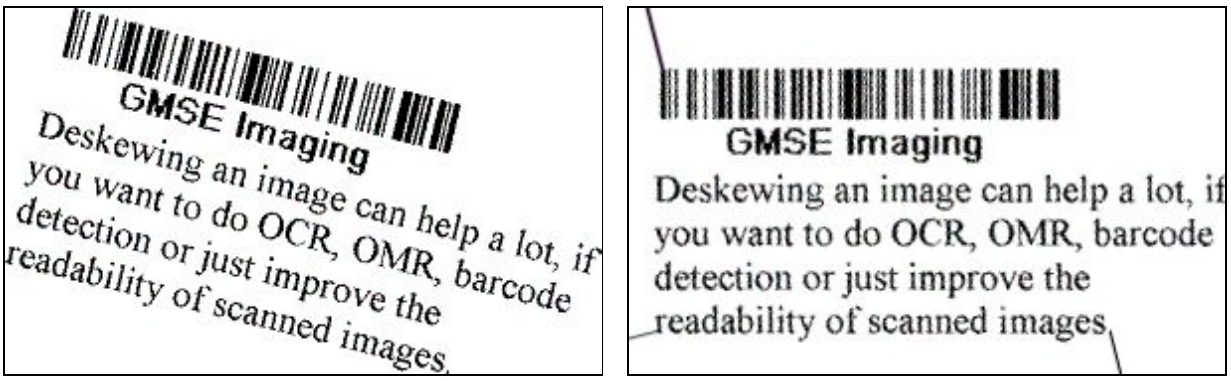


(a) Imagem original `neg_28.png`.

(b) Imagem de saída.

Figura 3 - Imagem de saída para o programa `align.py`, usando a técnica de projeção horizontal.

Também foi feito um teste usando imagens que possuem elementos que não são caracteres.



(a) Imagem original `sample1.png`. (b) Imagem de saída.

Figura 4 - Imagem de saída para o programa `align.py`, usando a técnica de projeção horizontal.

Para preencher as bordas que ficariam transparentes após o processo de rotação, com a função `rotate()` da biblioteca `skimage`, foi usado o parâmetro `mode='edge'`. Esse parâmetro pode ser encontrado em `numpy.pad` e, especificamente, `'edge'` replica os valores das bordas da imagem (array). Note que na Figura 3(b) temos linhas diagonais saindo das bordas da imagem, indicando que um pixel preto da borda foi replicado. Um outro parâmetro utilizado nessa função foi o `resize=True`, para aumentar o tamanho da imagem de forma a não cortar informação ao rotacionar.

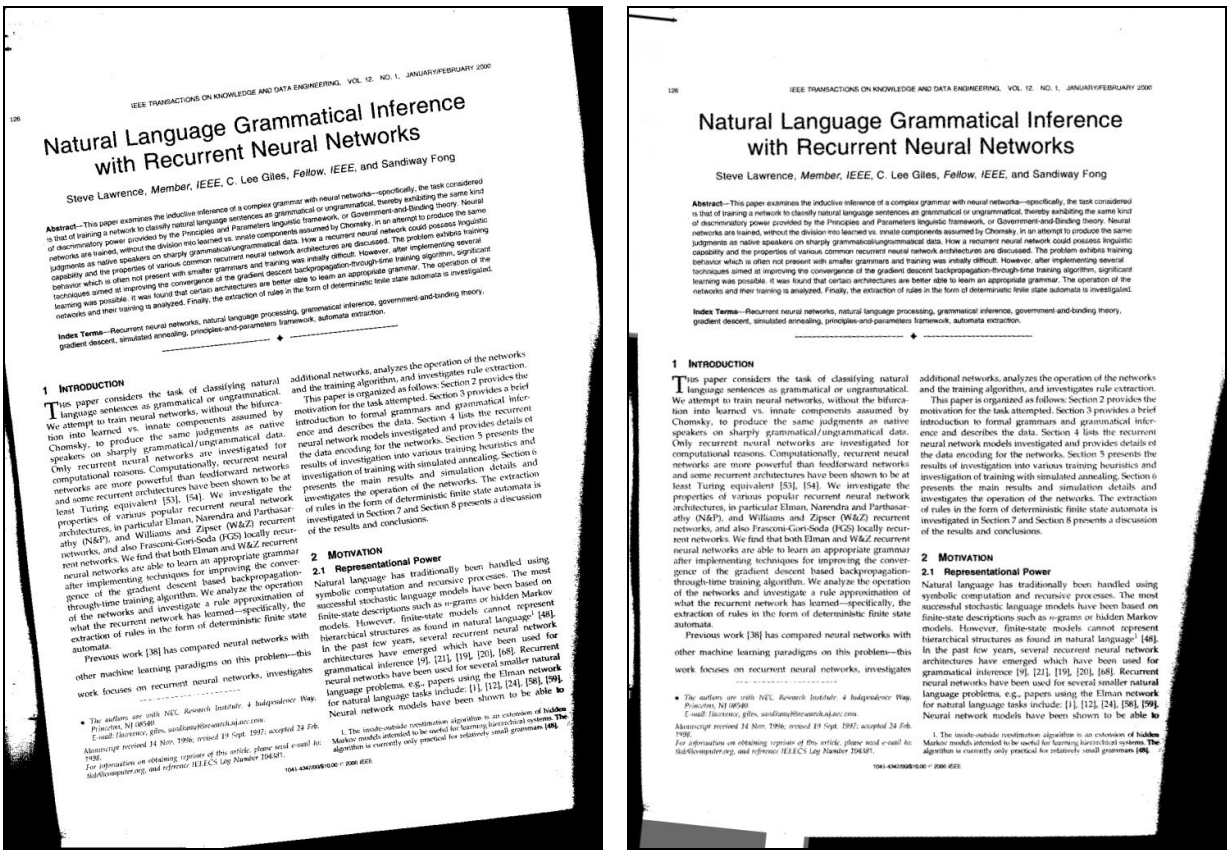
1.2. Transformada de Hough

Para o algoritmo de de transformada de Hough, a imagem foi transformada para a escala de cinza. Com essa imagem, foi aplicado o detector de bordas Canny (dado na função `canny()` da biblioteca `skimage`) e foi calculado o espaço de Hough para detectar linhas (dado nas funções `hough_line()` e `hough_line_peaks()`, ambas também disponíveis na biblioteca `skimage`). Um pseudocódigo é explicitado a seguir.

```
# deteccao de bordas
edges = canny(img_gray)

# deteccao de linhas
angles = hough_line(edges)
max_angles = hough_line_peaks(angles)
```

A função `hough_line_peaks()` retorna os ângulos onde houve maior concentração de encontro de retas no espaço de Hough. Logo, o ângulo de maior frequência em `max_angles` indica o ângulo da reta que foi mais encontrada na imagem. Esse ângulo é usado então para alinhar a imagem horizontalmente.



(a) Imagem original `sample2.png`. (b) Imagem de saída.

Figura 5 - Imagem entrada e de saída para o programa `align.py`, usando a técnica de transformada de Hough.

2. Tesseract Optical Character Recognition

O Tesseract Optical Character Recognition (Tesseract OCR)[1] é uma ferramenta que suporta Unicode e tem a habilidade de reconhecer mais de 100 linguagens numa imagem. Essa ferramenta foi utilizada para verificar o algoritmo aplicado à imagem, ou seja, se é possível reconhecer o texto contido na imagem.

2.1. Projeção horizontal

Foi aplicado o Tesseract para imagens alinhadas pelo algoritmo de projeção horizontal.

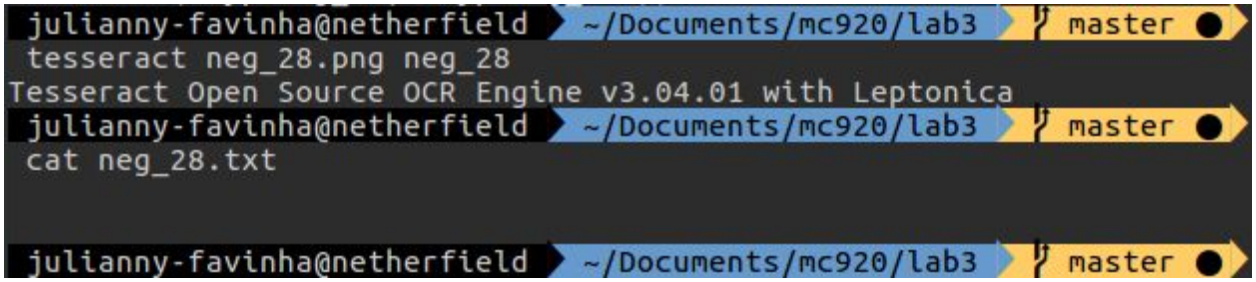


Figura 6 - Captura de tela da execução do Tesseract e arquivo de saída para o arquivo `neg_28.txt`. Note que o arquivo de saída está vazio, ou seja, não foi possível determinar nenhum caractere.

Vemos que para a imagem `neg_28.png`, a ferramenta não reconheceu texto algum. Após o processo de alinhamento, dado na imagem `neg_28_aligned.png`, a ferramenta reconheceu quase integralmente o texto, com apenas alguns caracteres diferentes, mas relativamente similares em sua forma.

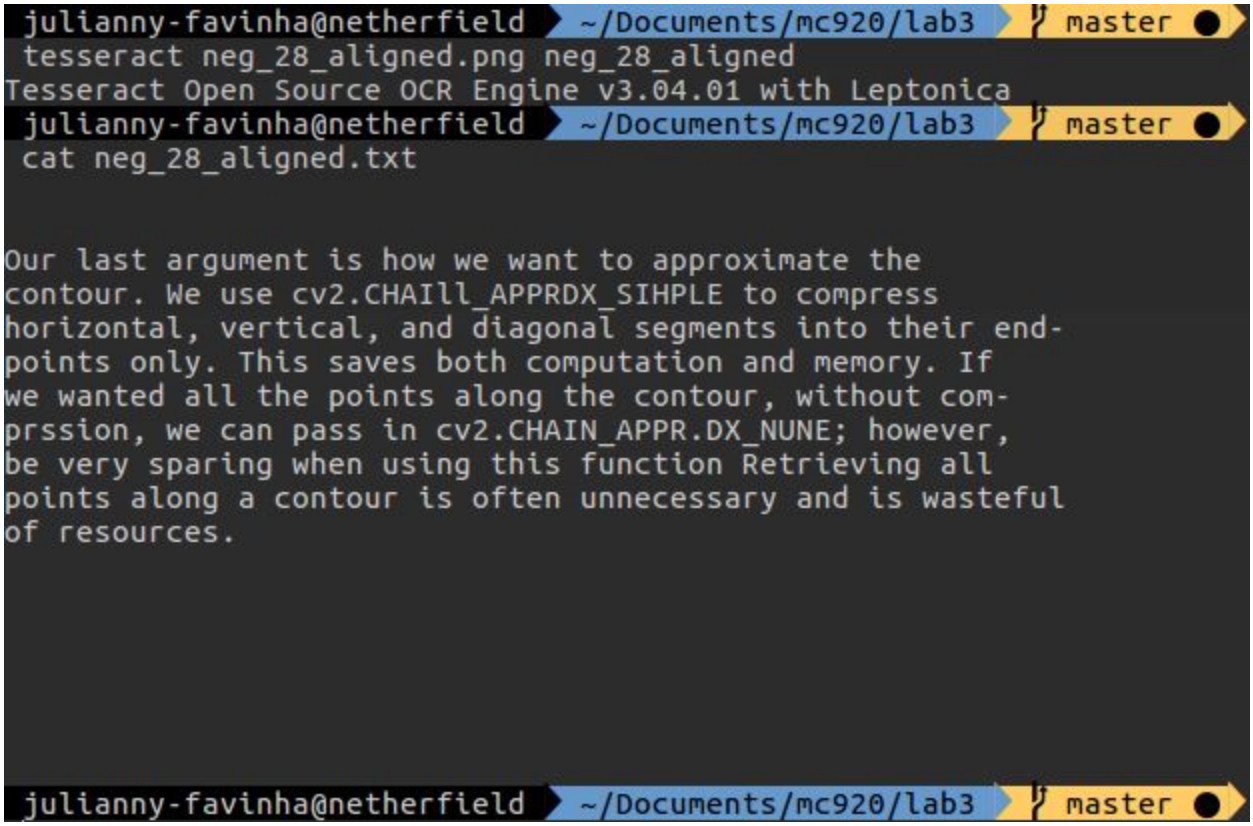


Figura 7 - Captura de tela da execução do Tesseract e arquivo de saída para o arquivo `neg_28_aligned.txt`. Note que a ferramenta conseguiu definir praticamente todos os caracteres da imagem.

2.2. Transformada de Hough

Foi aplicado o Tesseract para imagens alinhadas pelo algoritmo de transformada de Hough.

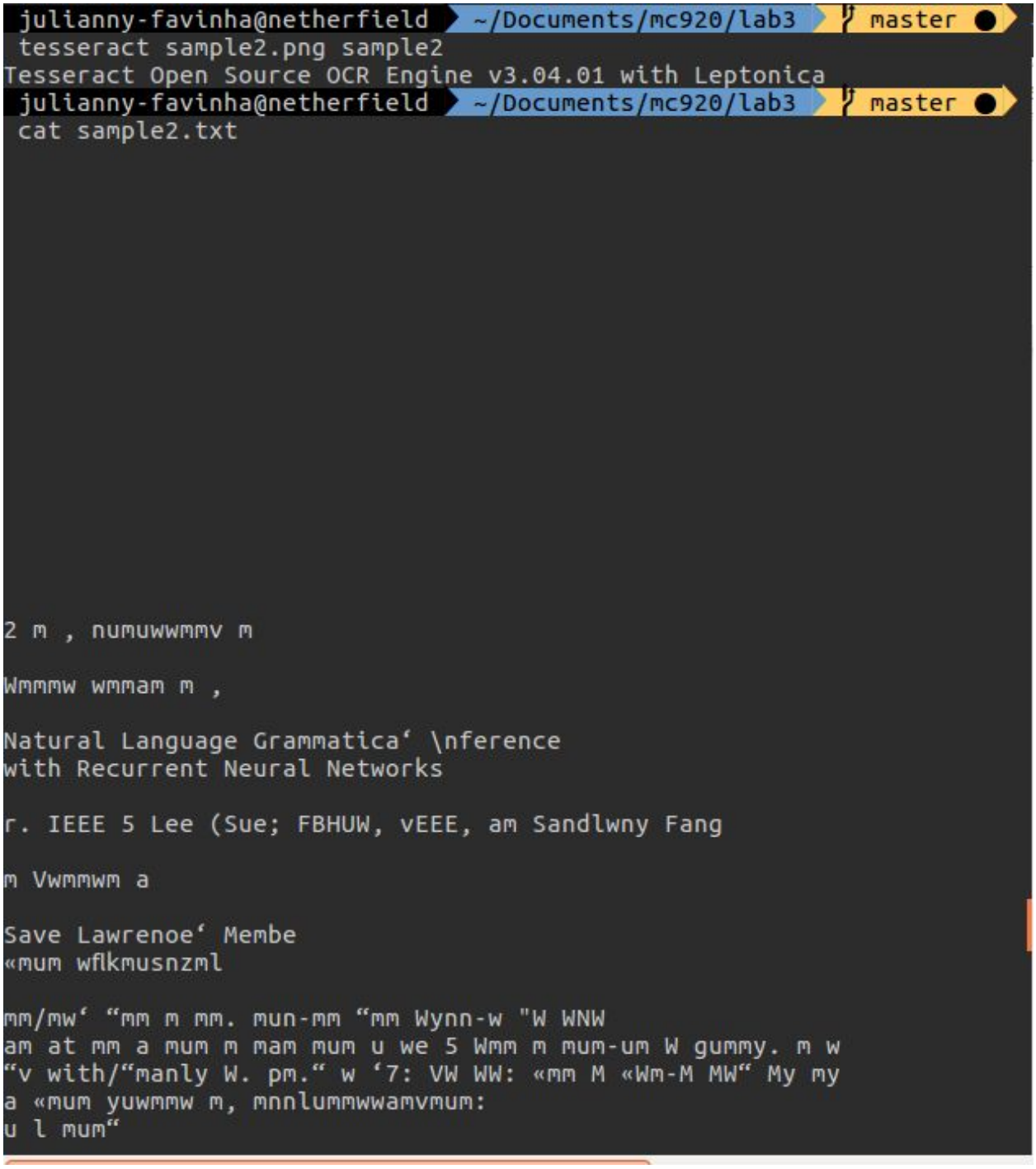


Figura 8 - Captura de tela da execução do Tesseract e parte do arquivo de saída para o arquivo `sample2.txt`.

Note que, para a imagem `sample2.png`, existem caracteres que não fazem parte do texto. A ferramenta detectou somente algumas palavras que fazem parte do título contido na imagem.

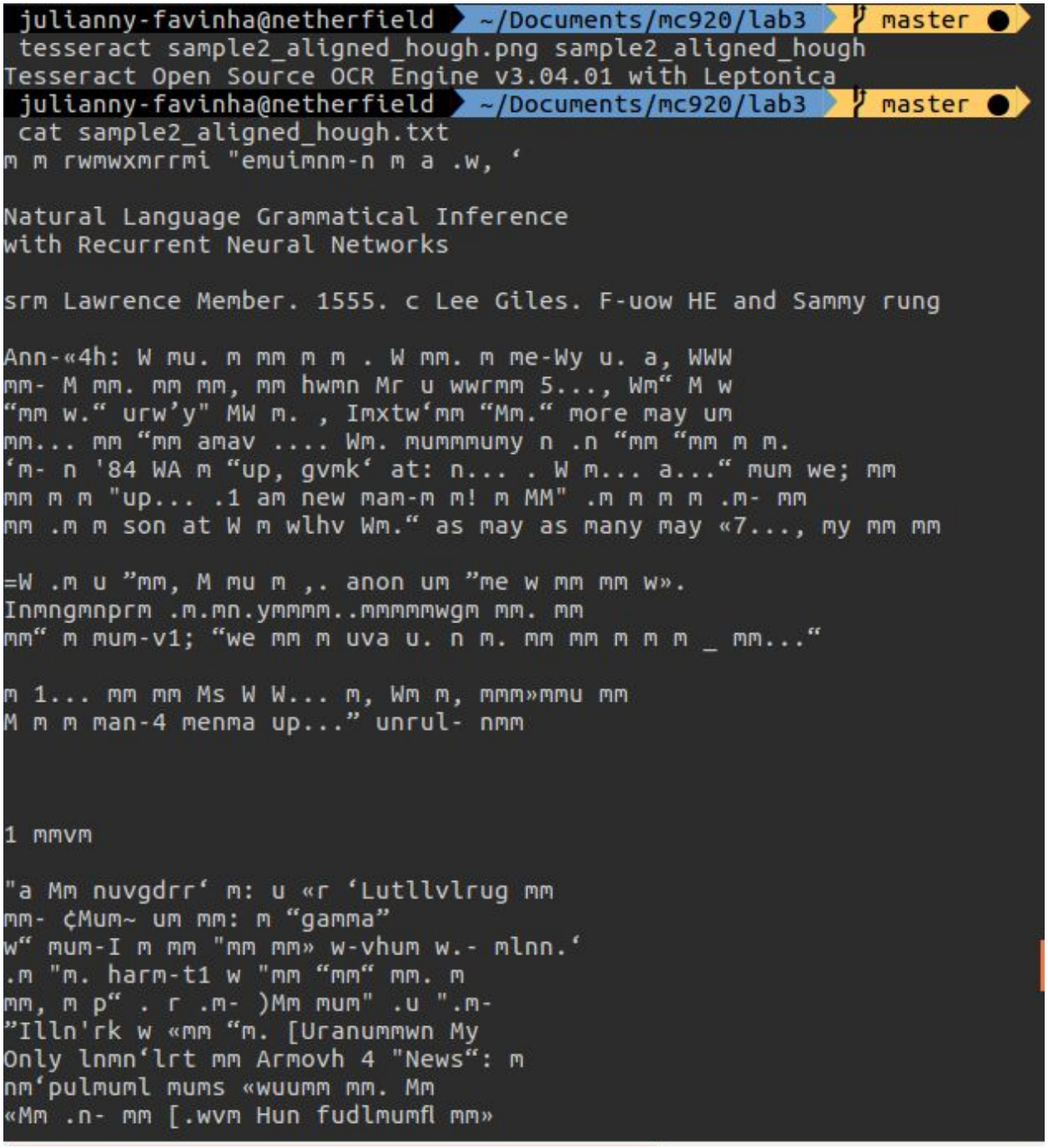


Figura 9 - Captura de tela da execução do Tesseract e parte do arquivo de saída para o arquivo `sample2_aligned_hough.txt`.

Note que, para a imagem alinhada `sample2_aligned_hough.png`, também existem caracteres que não fazem parte do texto. A ferramenta detectou somente textos que fazem parte do título contido na imagem, entretanto de uma forma relativamente melhor que a anterior, ou seja, a imagem original.

2.3. Conclusão

Ao analisar os resultados obtidos após executar o Tesseract OCR para as imagens, foi possível observar que o algoritmo é sensível à rotação do texto contido na imagem. Quanto mais horizontal for o texto, ou seja, quanto mais alinhado, mais caracteres serão reconhecidos. Além disso, a ferramenta também é sensível à qualidade da imagem.

3. Execução

É necessário que as bibliotecas **numpy** e **scikit-image** estejam instaladas. No terminal, execute

```
$ python3 align.py [nome da imagem de entrada].png modo [nome da imagem de saída].png
```

onde `modo` é a string “horizontal” ou “hough”, indicando a técnica que deseja-se utilizar.

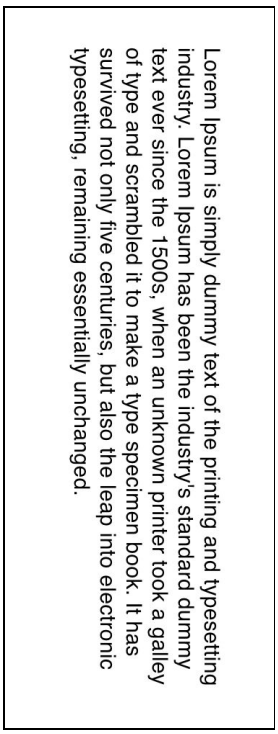
O programa `align.py` gera o seguinte arquivo:

- Imagem `[nome da imagem de saída].png`, que é a imagem onde foi aplicada a técnica desejada.

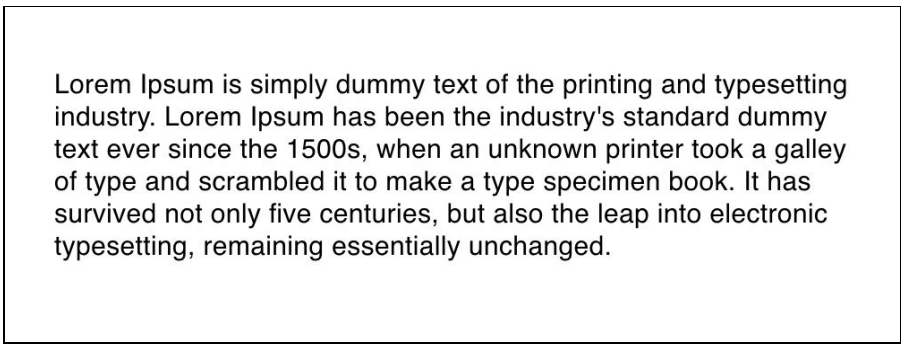
Além disso, o programa exibe na saída padrão o ângulo detectado pelo programa para alinhar a imagem de entrada com respeito ao eixo horizontal.

4. Limitações

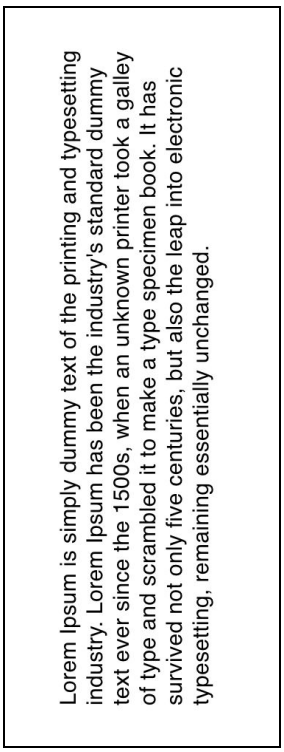
Tanto para projeção horizontal, quanto para transformada de Hough, não é possível garantir que imagens com rotação de -90° ou 90° não fiquem de ponta-cabeça.



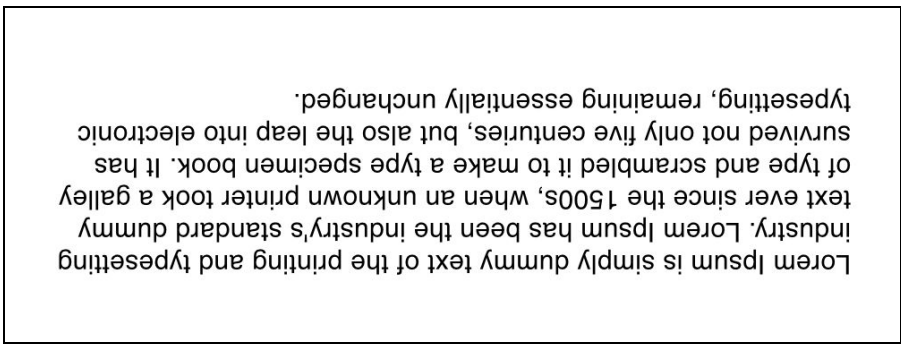
(a) Imagem original `pos_90.png`.



(b) Imagem após o alinhamento de `pos_90.png`.



(c) Imagem original `neg_90.png`.



(d) Imagem após o alinhamento de `neg_90.png`.

Figura 10 - Imagens entrada e saída do programa `align.py`.

De fato, como não temos nenhuma informação sobre a orientação do texto na imagem, não é possível cobrir todos os casos para que a imagem não fique de ponta-cabeça.

5. Referências

[1] Tesseract OCR. Disponível em <https://github.com/tesseract-ocr/tesseract>
Acesso em 8 mai 2018.