MC920 - Turma A.
Professor: Hélio Pedrini.

Trabalho 03

Aluno: Mateus Feitosa Olivi
RA: 222059

1. Introdução

A digitalização de documentos tem sido amplamente utilizada na preservação e disseminação de informação em formato eletrônico. Um problema frequente que ocorre nesse processo é o desalinhamento do documento, isto é, o posicionamento do papel com uma inclinação diferente do eixo do digitalizador.

A correção da inclinação é fundamental para o adequado funcionamento de sistemas de reconhecimento ótico de caracteres. E é isso o que será feito neste trabalho, irei implementar algoritmos para alinhamento automático de imagens de documentos.

2. Códigos desenvolvidos

Nesse tópico irei deixar claro as dependências utilizadas, como identificar e executar os programas construídos e especificar onde e como serão salvos os arquivos gerados.

2.1 Dependências

Esse trabalho foi desenvolvido em python 3.8, as bibliotecas utilizadas foram: OpenCV 4.5.3, NumPy 1.20.1, Matplotlib 3.3.4, Pillow 8.4.0, scikit-image 0.18.3 e o Tesseract OCR, essas são bibliotecas tipicamente utilizadas para o processamento de imagem e que fornecem recursos extremamente úteis, como será visto mais adiante.

2.2 Diretórios necessários

Esse projeto exige algumas pastas para as saídas dos meus programas. No mesmo diretório dos scripts em python é necessário uma pasta com nome ‘output’, dentro dessa pasta é necessário 2 outras pastas, uma para cada algoritmo desenvolvido:

- Para o algoritmo baseado em projeção horizontal, é necessário a pasta:
    a) output/projecao
- Para o algoritmo baseado na transformada de hough. é necessário a pasta:
    a) output/hough

O arquivo que eu enviarei já está com o conjunto de diretórios pronto porém, por conveniência, criei o programa setDir.py, que prepara o esquema de diretórios, a execução dele é simples, veja abaixo:

```
$ python setDir.py comando
```

Para o comando temos duas opções, set ou clear. O comando set pode ser usado para criar todo o esquema de diretórios, descrito nos parágrafos anteriores. Já o comando clear é utilizado para remover todos os arquivos .png e .txt dentro dos diretórios de output assim, caso queira utilizá-los, basta executar:

```
$ python setDir.py set
```

```
$ python setDir.py clear
```

Lembrando que o uso desse programa é opcional, já que os diretórios podem ser construídos manualmente e a deleção dos arquivos é opcional.

2.3 Entradas e Saídas

Os programas não necessitam de um diretório com as imagens de entrada, já que neles o caminho da imagem utilizada deve ser passado como argumento, isso será mostrado mais adiante. Algumas imagens padrão, fornecidas pelo professor, foram salvas na pasta input, essa pasta só foi introduzida para melhor organização do projeto, ela não é necessária para o funcionamento dos programas.

As imagens de saída dos programas são todas em PNG, e serão salvas em sua respectiva pasta de output, como já explicado. Além disso teremos algumas saídas de texto, no formato TXT, relacionados ao reconhecimento de caracteres das imagens utilizando o Tesseract OCR, elas também serão salvas em sua respectiva pasta de output

2.4 Execução

Para o projeto foi criado um programa em python do tipo .py, ele se refere ao exercício proposto no enunciado. Veja como devem ser executados os arquivos:

```
$ python alinhar.py imagem_de_entrada modo imagem_de_saida
```

2.5 Argumentos

Agora vou esclarecer alguns aspectos que podem não ter ficado claro sobre sobre alguns dos argumentos:

O argumento imagem_de_entrada se refere ao caminho da imagem de entrada, que deve estar no formato PNG.

O argumento modo se refere ao algoritmo que será utilizado para o alinhamento dos textos nas imagens. Os modos disponíveis são:

a) projecao: para chamar o algoritmo baseado em projeção horizontal.
b) hough: para chamar o algoritmo baseado na transformada de hough.

O argumento imagem_de_saida se refere ao nome da imagem de saída que será salva no diretório de output correspondente, como já explicado.

Veja um exemplo de uso para executar o programa quando a imagem_de_entrada é input/pos_24.png, o modo é baseado na transformada de Hough e a imagem_de_saída é rotacionada_pos_24.png

```
$ python alinhar.py input/pos_24.png hough rotacionada_pos_24.png
```

Como já explicado, a saída estará no diretório output/hough

3. Soluções

Nesse tópico irei apresentar as soluções do enunciado, irei explicar a lógica por trás do que foi feito e por fim apresentar e analisar os resultados obtidos.

Como já dito anteriormente, o programa feito faz a rotação utilizando o algoritmo baseado na transformada de Hough, ou o algoritmo baseado em projeção horizontal para se descobrir o ângulo de inclinação, nos subtópicos a seguir explicarei como esses algoritmos foram desenvolvidos.

Quando o programa é iniciado, a imagem é lida utilizando o OpenCV, usando a função cv2.imread(), considerando que a imagem será lida na escala de cinza

Para que o programa pudesse ser usado na análise de imagens de documentos, foi feito o uso do Tesseract OCR antes e depois do alinhamento da imagem. Para isso, depois de carregada a imagem, uma imagem segmentada foi obtida por meio de um algoritmo de limiarização local, isso porque ao aplicar uma técnica de limiarização local é possível obter uma imagem mais limpa para o OCR, o que garante resultados melhores.

Para a limiarização, foi utilizado o limiar adaptativo, dado pela função cv2.adaptiveThreshold(), que calcula o limiar para pequenas regiões da imagem, isto é, existirão valores de limiar diferentes para regiões diferentes.

O método utilizado, para a limiarização, foi o Gaussiano, definido pelo parâmetro cv2.ADAPTIVE_THRESH_GAUSSIAN_C, que determina o valor do limiar como a soma ponderada de valores de vizinhança, em que os pesos são uma janela gaussiana.

Ainda, como o objetivo era obter uma imagem mais limpa para obter melhores resultados usando o OCR, o tipo de limar usado foi o binário, definido pelo parâmetro cv2.THRESHOLD_BINARY.

3.1 Algoritmo baseado em projeção horizontal

A projeção horizontal de uma imagem é o perfil de projeção dela ao longo do eixo horizontal. Ela pode ser calculada, para cada linha, como a soma de todos os valores de pixel da coluna dentro da respectiva linha.

A detecção de inclinação baseada em projeção horizontal foi realizada variando o ângulo testado e projetando a quantidade de pixels pretos em cada linha de texto. O ângulo escolhido é aquele que otimiza uma função objetivo calculada sobre o perfil da projeção horizontal. A função objetivo escolhida foi a soma dos quadrados das diferenças dos valores em células adjacentes do perfil de projeção.

A primeira coisa que foi feita foi obter uma imagem após a aplicação da limiarização, através da função cv2.adaptiveThreshold(), assim como foi mostrado anteriormente. Depois foi necessário obter uma imagem binária, onde os pixels do texto fossem representados por 1 e pixels que não faziam parte do texto fossem representados por 0. Para isso, foi criada uma nova imagem, iniciada com zeros, os pixels dessa nova imagem receberam 1 para cada pixel 0 da imagem após a limiarização, isso garantiu que se fosse obtida uma nova imagem, em que os pixels do texto valiam 1 e os pixels que não faziam parte do texto valiam 0.

Obtida a imagem binária, pude utilizar a função np.sum(), do Numpy, que realiza uma soma dos elementos da matriz sobre o eixo dado, no meu caso o eixo horizontal. A projeção foi então exibida como um histograma, utilizando o Matplotlib.

Para descobrir o ângulo de rotação, a imagem deve ser girada em um intervalo de -90 a 90 graus, que cobre um total de 180 graus de rotação, uma vez que se presume que a imagem do documento fornecida não será de cabeça para baixo. As rotações serão feitas utilizando a função skimage.transform.rotate() do scikit-image.

Para cada imagem rotacionada, a projeção horizontal foi calculada. O perfil com maior variação entre picos e vales é o que proporciona o melhor alinhamento da imagem do documento, já que o número de pixels de texto colineares é o maior.

Para achar o perfil com maior variação entre picos e vales, pode-se usar a soma dos quadrados das diferenças dos valores entre células adjacentes do perfil de projeção, quanto maior o resultado dessa operação, maior a variação entre picos e vales e portanto melhor o perfil. Considere a projeção horizontal x_0, x_1, …, x_{n-1} em que x_i representa a soma dos pixels da linha i, pode-se calcular a soma dos quadrados das diferenças dos valores de linhas adjacentes seguindo a expressão:

$$S = \sum_{i=1}^{N-1} (x_i - x_{i-1})^2$$

O ângulo de rotação será aquele em que a imagem tiver maior valor de S, já que um maior valor de S representa uma maior variação entre o tamanho das barras laterais, o que implica no número de pixels colineares ser o maior, sendo característico em um texto.

Apliquei toda essa metodologia no meu código, e executei o programa alinhar.py para o arquivo neg_28.png e pos_24.png da seguinte forma:

```
$ python alinhar.py input/neg_28.png projecao alinhada_neg_28.png
```

Vamos primeiro fazer a análise do histograma das imagens de entrada:

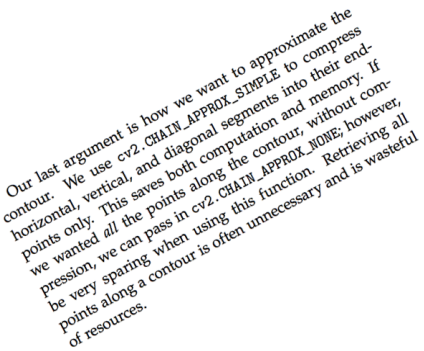


Imagem de entrada neg_28.png

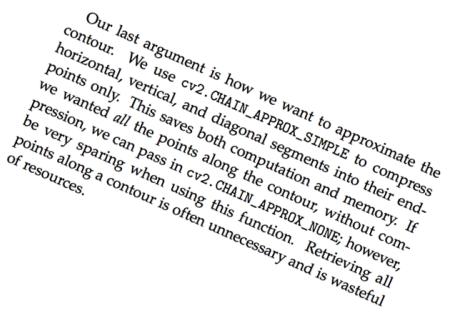
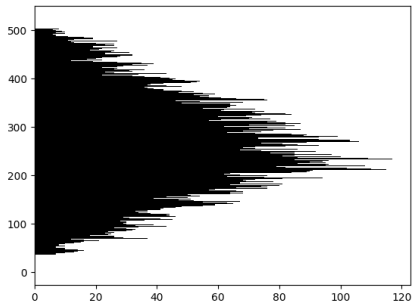
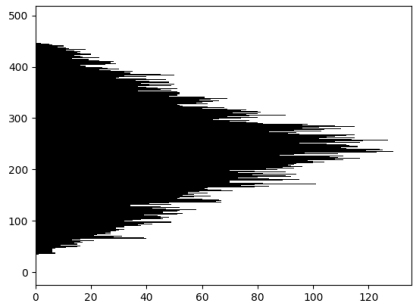


Imagem de entrada pos_24.png



Projeção horizontal de neg_28.png



Projeção horizontal de pos_24.png

Nas imagens acima estão representados os perfis das projeções horizontais das imagens inclinadas, é possível notar que a parte central dos histogramas é onde as barras estão maiores, isso acontece porque as inclinações são tais que a quantidade de pixels de texto ficam mais concentradas no meio da projeção, o que era esperado analisando as imagens de entrada.

Agora vamos ver as imagens após terem sido alinhadas e seus histogramas:

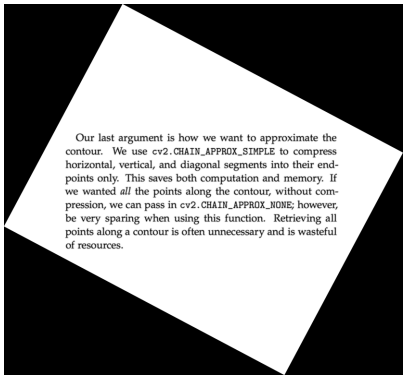


Imagem alinhada de neg_28.png

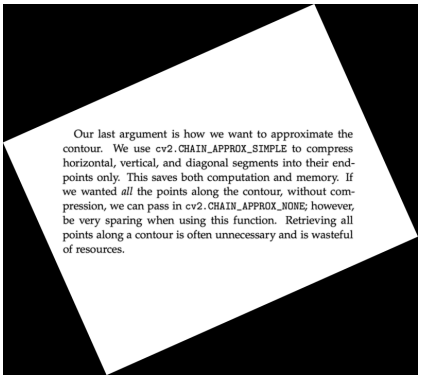
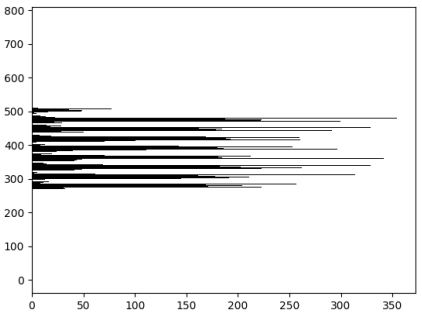
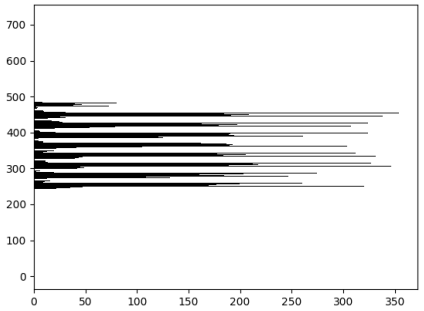


Imagem alinhada de pos_24.png



Projeção horizontal de neg_28.png alinhada



Projeção horizontal de pos_24.png alinhada

Inicialmente, é possível notar visualmente que os textos das imagens ficaram bem alinhados. Observando os histogramas é possível ver claramente as linhas que se formaram, que representam as linhas do texto, ainda é visível que o histograma possui uma grande variação entre o nível das barras laterais, resultado da soma dos quadrados das diferenças, o que nos mostra que esta função objetivo foi uma boa escolha, já que assim foi possível identificar o perfil em que se possuía faixas de pixels colineares, que existem devido às linhas do texto.

O OCR foi aplicado sobre a imagem de entrada e saída, usando a imagem de função image_to_string() do pytesseract, junto com a função Image.fromarray() da biblioteca Pillow, que cria uma imagem a partir de uma matriz numpy. A seguir, vou apresentar os textos que foram encontrados na imagem inclinada e na imagem alinhada:

Para as imagens neg_28.png e pos_24.png originais não foi encontrado nenhum texto antes do alinhamento, porém após o alinhamento foi detectado o seguinte texto, para ambas:

> **Our last argument is how we want to approximate the contour. We use cv2.CHAIN APPROX_SIMPLE to compress horizontal, vertical, and diagonal segments into their end-points only. This saves both computation and memory. If we wanted all the points along the contour, without compression, we can pass in cv2.CHAIN_APPROX_NONE; however, be very sparing when using this function. Retrieving all points along a contour is often unnecessary and is wasteful of resources.**

Pode-se perceber que, o texto foi identificado com sucesso, se compararmos os textos das imagens originais com o texto identificado pelo OCR, é possível notar que são idênticos, o que mostra que a técnica baseada em projeção horizontal foi muito boa para alinhar as imagens, o que permitiu com que o OCR pudesse identificar os textos com sucesso, o que indica que a metodologia em questão é válida.

3.2 Algoritmo baseado na transformada de Hough.

A transformação de Hough é uma técnica comum para detectar o ângulo de inclinação de uma imagem de documento, isso porque é uma técnica que pode ser usada para detectar linhas. Uma linha é representada como y = mx + b, ou na sua forma paramétrica polar ρ = xcosθ + ysinθ, onde ρ é a distância perpendicular da origem à reta, e θ é o ângulo formado por esta reta perpendicular e o eixo horizontal medido no sentido anti-horário.

A detecção de inclinação da imagem baseada na transformada de Hough assume que os caracteres de texto estão alinhados. As linhas formadas pelas regiões de texto são localizadas por meio da transformada de Hough, a qual converte pares de coordenadas (x, y) da imagem em curvas nas coordenadas polares (ρ, θ).

Para a aplicar a técnica, apliquei inicialmente um detector de bordas na imagem de entrada utilizando a função cv2.Canny() para gerar um mapa de bordas, que foi utilizado pela transformada para encontrar linhas na imagem.

Após isso, apliquei a transformada utilizando a função cv2.HoughLinesP(), que implementa uma Transformada de Hough Probabilística, uma otimização da Transformada de Hough. Como parâmetros, utilizei o mapa de bordas descrito no parágrafo anterior, além de definir o comprimento mínimo da linha como 80px e o espaçamento máximo permitido entre os segmentos de reta para considerá-los uma única reta como 10px. Quando a função retorna, ela devolve os dois pontos extremos de cada segmento de reta. Sabendo que o primeiro ponto extremo do segmento é (x_1, y_1) e o segundo é (x_2, y_2), pode-se calcular o ângulo de inclinação desse segmento calculando o arcotangente, o que pode ser feito usando a função np.arctan2() do Numpy, como a seguir:

$$\theta = np.\arctan2(y_2 - y_1, x_2 - x_1)$$

Os ângulos foram armazenados em uma lista, e o ângulo de inclinação foi encontrado através da mediana desses ângulos com a função np.median(), do Numpy. Eu optei por usar a mediana porque, diferentemente da média, ela é menos sensível a outliers.

Apliquei toda essa metodologia no meu código, e executei o programa alinhar.py, considerando o algoritmo baseado na transformada de Hough, e para os arquivos neg_28.png e pos_24.png, da seguinte forma:

```
$ python alinhar.py input/pos_24.png hough alinhada_pos_24.png
```

Primeiro vamos ver como ficou as imagens com as linhas detectadas através da transformada de Hough:

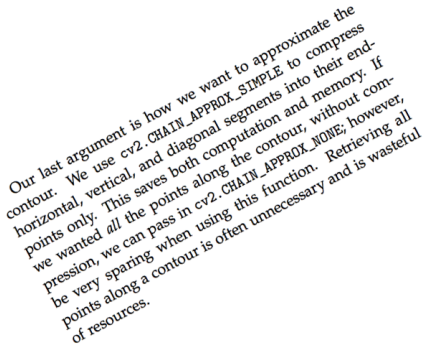


Imagem de entrada neg_28.png

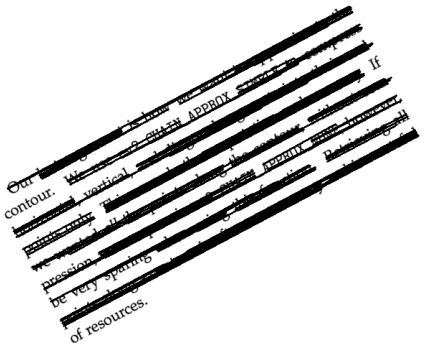


Imagem neg_28.png com linhas detectadas

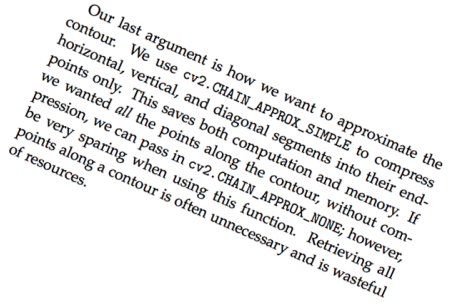


Imagem de entrada pos_24.png

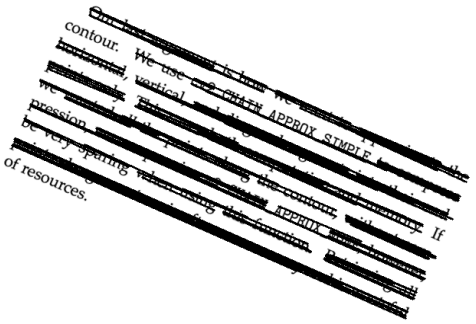


Imagem pos_24.png com linhas detectadas

Analisando as imagens é possível notar que a transformada de Hough realmente conseguiu detectar as linhas do texto, é claro que pode-se notar que ocorreu uma certa redundância, existem múltiplos segmentos de reta sobre a mesma linha de texto, entretanto isso não é um problema para o nosso caso. Como foi obtido os extremos de cada segmento, foi possível calcular a inclinação de cada segmento utilizando o arco tangente e assim obter o valor do ângulo da mediana, garantindo que se pudesse obter o ângulo de inclinação do texto.

A seguir as imagens pos_24.png e neg_28.png após a correção de inclinação, com os ângulo descoberto usando a metodologia de Hough:

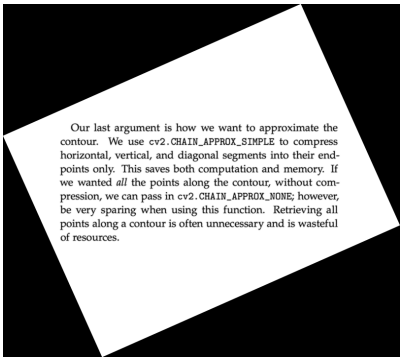


Imagem alinhada de pos_24.png

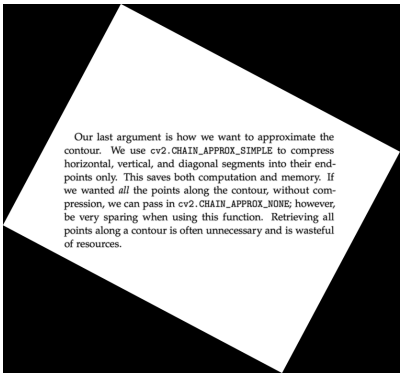


Imagem alinhada de neg_28.png

Nas imagens é possível notar visualmente que as inclinações foram corrigidas, indicando que o algoritmo baseado na transformada de Hough também foi bom para achar a inclinação do texto.

Da mesma forma que foi feito para o algoritmo anterior, o OCR foi aplicado sobre a imagem de entrada e saída para identificar os textos presentes nas imagens. Para as imagens neg_28.png e pos_24.png originais não foi encontrado nenhum texto antes do alinhamento, porém após o alinhamento foi detectado o seguinte texto, para ambas:

> **Our last argument is how we want to approximate the contour. We use cv2.CHAIN_APPROX.SIMPLE to compress horizontal, vertical, and diagonal segments into their end-points only. This saves both computation and memory. If we wanted ail the points along the contour, without com-pression, we can pass in cv2.CHAIN_APPROX_NONE; however, be very sparing when using this function. Retrieving all points along a contour is often unnecessary and is wasteful of resources.**

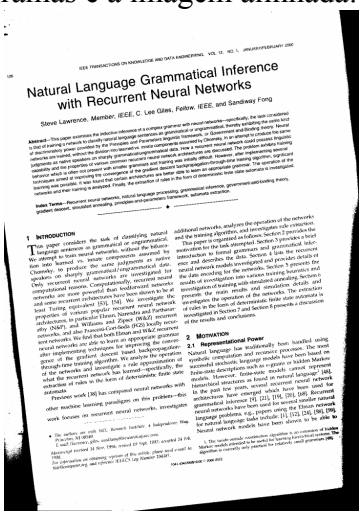
Pode-se perceber que o texto foi identificado com sucesso. Se compararmos os textos das imagens originais com o texto identificado pelo OCR, é possível notar que são idênticos, o que mostra que a técnica baseada na transformada de Hough também foi muito boa para alinhar as imagens, o que permitiu com que o OCR pudesse identificar os textos com sucesso, o que indica que a metodologia em questão também é válida.

3.3 Testes no arquivo sample2.png

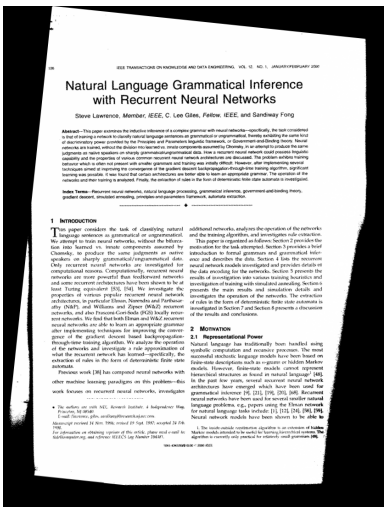
Executei o meu programa para o arquivo sample2.png, uma página de documento de verdade, primeiro para o algoritmo baseado em projeção horizontal, da seguinte forma:

```
$ python alinhar.py input/sample2.png projecao alinhada_sample2.png
```

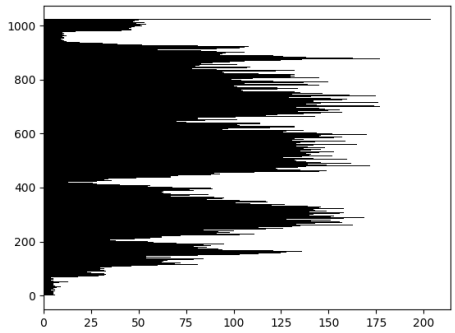
Com a execução do programa obtive um ângulo de inclinação de - 6 graus, vamos ver os histogramas e a imagem alinhada:



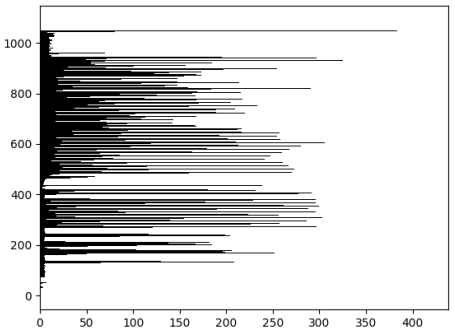
sample2.png original



sample2.png alinhada



Projeção horizontal de sample2.png original



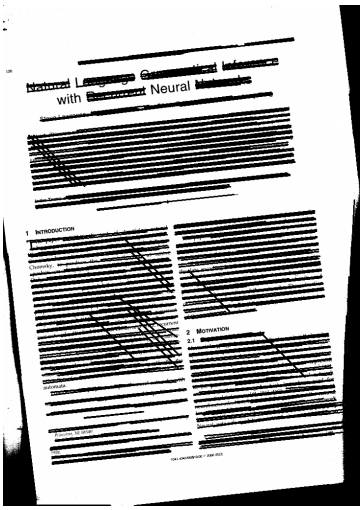
Projeção horizontal de sample2.png alinhada

Através da observação visual da imagem alinhada e dos histogramas, é possível ver que o programa conseguiu alinhar muito bem essa página de documento.

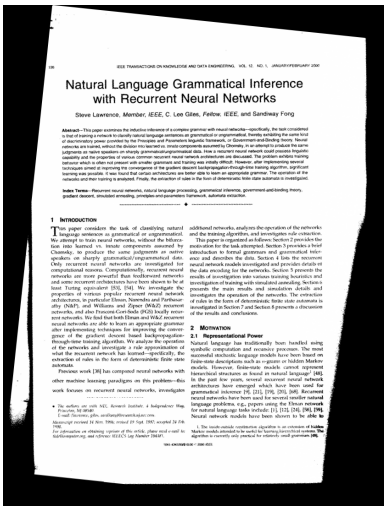
Agora executei o meu programa para o arquivo sample2.png, para o algoritmo baseado na transformada de Hough, da seguinte forma:

```
$ python alinhar.py input/sample2.png hough alinhada_sample2.png
```

Com a execução do programa obtive um ângulo de inclinação de -6,019 graus, vamos ver a imagem alinhada e a imagem com os segmentos destacadas:



sample2.png c/ retas destacadas



sample2.png alinhada

Através da observação visual da imagem alinhada e das retas destacadas, é possível ver que as linhas foram bem identificadas pela transformada de Hough e que o programa conseguiu alinhar muito bem essa página de documento.

Ainda, o OCR não conseguiu identificar bem o texto tanto na imagem original, quanto na alinhada, provavelmente porque é uma imagem com muitas informações espalhadas pelo documento em colunas, parte do texto está centralizada, uma parte à esquerda e uma à direita, os textos podem ser vistos nos arquivos sample2.txt e alinhada_sample2.txt nos diretórios output/projecao e output/hough.

4. Limitações do projeto

A maior limitação do projeto está relacionado a imagem de entrada, ela deve estar em escala de cinza e o texto não pode estar de cabeça para baixo, tanto na técnica da projeção horizontal, quanto na da transformada de Hough, não há diferenças entre a imagem estar de cabeça para baixo, ou não, isso porque os algoritmos estão identificando o ângulo de inclinação das linhas de texto, um texto de ponta cabeça está tão alinhado quanto um texto na orientação normal.

Além disso, lembre-se que o funcionamento do programa está garantido para o python na versão 3 e com as bibliotecas nas versões citadas no tópico 2.1, os programas não foram testados usando o python 2 ou outras versões das bibliotecas.

5. Conclusão

Ambos os algoritmos apresentados se mostraram capazes de fazer o alinhamento de textos de maneira bem precisa, desde que os textos não estejam de cabeça para baixo. Foi possível ver que para textos maiores, como da imagem sample2.png o OCR teve dificuldades de reconhecer o texto, mesmo alinhado, isso porque, como dito anteriormente, a imagem tem muitas informações espalhadas pelo documento em colunas, parte do texto está centralizada, uma parte à esquerda e uma à direita

Esse trabalho ainda ajudou a fixar conceitos relacionados a projeções dos pixels em um dos eixos, bem como a aplicação prática da transformada de Hough, mostrando que esses conceitos podem ser aplicados diretamente no campo de identificação de textos.