Universidade Estadual de Campinas
Instituto de Computação

Introdução ao Processamento Digital de Imagens (MC920 / MO443)
Professor: Hélio Pedrini

Trabalho 3

Julio Vinicius Amaral Oliveira – 230537

Data de Entrega: 20 de maio de 2025

Resumo

O objetivo deste relatório é mostrar a utilização de 2 algoritmos para o alinhamento de imagens, utilizamos o algoritmo de projeção horizontal e a transformada de Hough. Neste relatório, iremos detalhar o métodos, escolhas realizadas, testes e resultadoss

1 Introdução

O desalinhamento de documentos durante o processo de digitalização é um problema comum que afeta o desempenho de sistemas de Reconhecimento Ótico de Caracteres (OCR) e a utilização de algoritmos para alinhar textos é uma etapa fundamental para que modelos mais tarde possam ser utilizados com maior eficiência. O objetivo do trabalho é implementar dois algoritmos para a detecção e correção automática do alinhamento: um baseado em projeção horizontal e outro baseado na transformada de Hough.

2 Especificação do Problema

O objetivo principal é desenvolver algoritmos que identifiquem o ângulo de inclinação de uma imagem de documento e rotacionar esta imagem para corrigir essa inclinação. Iremos discutir sobre duas implementações utilizadas para resolver esse problema

3 Técnicas de Alinhamento Implementadas

3.1 Técnica Baseada em Projeção Horizontal

A primeira técnica utilizada foi a projeção horizontal, que utiliza um algortimo simples: percorremos um angulo de θ de -90 a 90 graus. Utilizando algumas funções do próprio openCV como `getRotationMatrix2D` e `warpAffine`, rotacionamos a imagem para cada angulo e calculamos a soma dos pixels de cada linha. Daí, calculamos a diferença quadrática entre a soma dos pixels de linhas linhas adjacentes e somamos esses valores em um somador. Por fim, escolhemos o ângulo que maximiza essa soma.

3.2 Técnica Baseada na Transformada de Hough

A segunda técnica utilizada foi a transformada de Hough, para isso primeiramente começamos aplicando um detector de bordas na imagem, já que o conteúdo de interno de cada letra não seria útil para encontrar o melhor ângulo e ao diminuir a quantidade de pixels pretos, conseguimos aumentar a performance do algoritmo. A princípio, foi aplicado um filtro de Sobel, mas percebeu-se que as bordas ainda estavam muito grossas, então foi aplicado um filtro de Canny. Após isso, foi aplicada a transformada de Hough para cada pixel preto

da imagem. Com o valor de ρ retornado, conseguimos montar uma matriz de $\theta X \rho$ cujos valores eram atualizados quando um pixel dentro do intervalo de ρ e θ era encontrada (foi utilizado um bin de tamanho 3 escolhido de forma arbitrária para os valores de ρ). Após isso, bastou-se encotrar qual havia sido o elemento da matriz com o maior valor. Como o ângulo encontrado é o ângulo perpendicular à linha, foi necessário subtrair 90 graus do ângulo encontrado.

3.3 Comparação entre as Técnicas

As duas técnicas foram testadas nas imagens de teste e obteram resultados praticamente iguais, porém a técnica de projeção horizontal possuia tempo de execução visivelmente menor do que a técnica da transformada de Hough. Oque é explicado já que a técnica de projeção horizontal teve o auxilio da função `np.sum` que com certeza impactou positivamente no tempo de execução. Enquanto a técnica da transformada de Hough não teve vetorizaçào envolvida. Segue abaixo os resultados de tempo e performance para cada entrada:

Imagem	**Técnica**	**Ângulo°**	**Entrada**	**Saída**	**Tempo (s)**
neg_4	P. Horizontal	-4	91.93	95.00	0.25
neg_28	P. Horizontal	-28	0.00	95.61	0.39
partitura	P. Horizontal	26	0.00	0.00	0.25
pos_24	P. Horizontal	24	0.00	95.54	0.37
pos_41	P. Horizontal	41	0.00	95.45	0.43
sample1	P. Horizontal	14	0.00	87.64	0.13
sample2	P. Horizontal	-6	36.69	64.76	0.89
neg_4	T. Hough	-4	91.93	95.00	1.80
neg_28	T. Hough	-28	0.00	95.61	1.85
partitura	T. Hough	30	0.00	0.00	1.12
pos_24	T. Hough	24	0.00	95.54	1.83
pos_41	T. Hough	41	0.00	95.45	1.86
sample1	T. Hough	14	0.00	87.64	0.57
sample2	T. Hough	-6	36.69	64.76	7.61

Tabela 1: Resultados dos testes de alinhamento: ângulo estimado, confiança do OCR antes e depois, e tempo de execução.

Podemos notar que os resultados foram idênticos, e apenas o tempo de execução que foi bem superior no caso da transformada de Hough.

4 Escolhas de estruturas de dados e implementação

Durante o desenvolvimento do projeto, fizemos algumas escolhas de estruturas de dados que tiveram como motivação principalmente a performance.

4.1 Projeção Horizontal

No caso da projeção horizontal, não utilizamos nenhuma estrutura muito complexa, apenas utilizamos variáveis simples para controlar qual era o ângulo que gerava o maior somador, também utilizamos uma lista para armazenar os valores de cada linha da imagem e o operador `np.sum` para calcular a soma dos pixels de cada linha, que foi motivado principalmente pela performance.

4.2 Transformada de Hough

No caso da transformada de Hough, usamos algumas estruturas de dados, primeiramente utilizamos uma matriz para guardar os resultados obtidos a partir de cada transformada realizada, com bins de tamanho 3 escolhidos de forma arbitrária. Também utilizamos dicionários para guardar os valores de cos e sin de cada ângulo, ajudou a não precisar recalcular esses valores para cada pixel da imagem.

4.3 Estrutura do Código & Funções auxiliares

O código foi estruturado pensando em receber os parâmetros indicados como sugestão no enunciado do relatório, então ele recebe o caminho da imagem de entrada, o modo de alinhamento 'projecao_horizontal' ou 'transformada_hough' e o caminho da imagem de saída. Então dividimos a função principal em 2 partes, cada parte sendo responsável pelo respectivo modo de alinhamento. Ao longo do código utilizamos algumas funções auxiliares para facilitar a leitura e também utilizamos uma função que utilizava o `pytesseract` para calcular a confiança do OCR e assim ter uma medida quantitativa de quão alinhada estava a imagem final. Como foi mostrado na tabela 1, o OCR não conseguiu ler quase nenhuma imagem desalinhada, porém após a aplicação dos algoritmos, o OCR obteve um resultado bem satisfatório. Para mostrar a imagem final utilizamos uma função que aplicava um fundo branco a imagem, para evitar situações em que uma imagem retangular ficasse de alguma forma distorcida ou perdesse o seu conteúdo após a rotação, como havia acontecido com o exemplo de teste da partitura.

5 Forma de Execução

Para executar o código é necessário ter o pytesseract, numpy e openCV instalados e basta executar o seguinte comando: `python alinhar.py <caminho da imagem de entrada> <modo de alinhamento> <caminho da imagem de saída>`. O modo de alinhamento podendo ser 'projecao_horizontal' ou 'transformada_hough'.

6 Conclusão

As implementações dos algoritmos de alinhamento propostos foram bem eficazes para alinhar as imagens, ambos conseguiram encontrar o ângulo de inclinação corretamente, gerando uma imagem final bem alinhada. Os resultados foram percebidos tanto de forma qualitativa ao olhar o output final, conseguiamos perceber claramente que a imagem estava alinhada, quanto de forma quantitativa, já que o OCR conseguiu nos retornar um nível de confiança atrelado à imagem final com valores consideravelmente maiores do que o nível de confiança relacionados a imagem inicial. A única consideração que podemos fazer é quanto ao tempo de execução que se mostrou bastante lento, principalmente para imagens maiores, apesar dos esforços para tentar otimizar o algoritmo. Uma forma de reduzir isso seria pensar em substituir os loops por uma operação vetoriza do numpy.