

UNIVERSIDADE ESTADUAL DE CAMPINAS

**MC920/MO443 – Introdução ao Processamento de Imagem
Digital**

Prof. Dr. Hélio Pedrini

Trabalho 3

**André Amadeu Satorres
231300**

Campinas, maio de 2024

1. Introdução.....	2
2. Entrada e Saída de dados.....	2
3. O Algoritmo.....	2
3.1 Leitura da entrada.....	3
3.2 Projeção Horizontal.....	3
3.3 Transformada de Hough.....	8
3.4 Saída.....	9
4. Análise.....	10
4.1 Imagens iniciais.....	10
4.2 Outras imagens.....	10
5. Considerações.....	12

1. Introdução

Nos dias atuais, a digitalização de documentos é uma prática cada vez mais utilizada. Para documentos pessoais, é muito conveniente tê-lo como foto ou PDF digital, pois viabiliza o fácil acesso, e previne a perda do original, além também de tornar desnecessário carregar consigo uma série de documentos importantes. Com a presença de smartphones, é possível ir ao cinema e apresentar seu RG e até mesmo carteirinha de estudante de forma digital, assim como a Carteira Nacional de Habilitação, o que torna a vida mais prática e segura. Também para documentos acadêmicos ou de trabalho, a digitalização torna possível a preservação e o compartilhamento de tais documentos com várias pessoas, podendo ser provas, livros, receitas, etc.

Todavia, a maior parte dos documentos possuem um alinhamento específico, importante por exemplo para a leitura, ou para verificação. Ao digitalizar documentos, muitos podem ficar desalinhados, o que prejudica a utilização de tal documento, por humanos, ou até mesmo algoritmos de escaneamento. Portanto, a correção de desalinhamento é fundamental para a preservação e futura utilização de documentos digitalizados.

Neste trabalho, serão utilizadas duas técnicas de detecção e correção de desalinhamento, um baseado na projeção horizontal, o outro baseado na Transformada de Hough. Teremos como entrada do problema uma imagem de algum documento digitalizado e desalinhado. O programa deve retornar uma imagem daquele documento, corrigida sua inclinação.

2. Entrada e Saída de dados

No presente trabalho, o algoritmo foi implementado na linguagem de programação Python. Há um arquivo executável: t3.py. Há uma pasta “images”, da qual o programa vai ler todas as imagens PNG e realizar o alinhamento. As imagens de saída serão gravadas no diretório “out/mode/”, onde “mode” é a técnica de alinhamento escolhida: “proj” ou “hough”. Exemplos de execução:

```
Python
python t3.py
Mode: hough (ou proj)
```

3. O Algoritmo

O algoritmo do trabalho é composto essencialmente destes passos:

- 1) Abrir uma imagem de entrada convertida para escala de cinza
- 2) Perguntar ao usuário qual método utilizar
- 3) Usar o método para encontrar o ângulo de inclinação de imagem
- 4) Rotacionar a imagem usando o ângulo descoberto
- 5) Salvar a imagem final

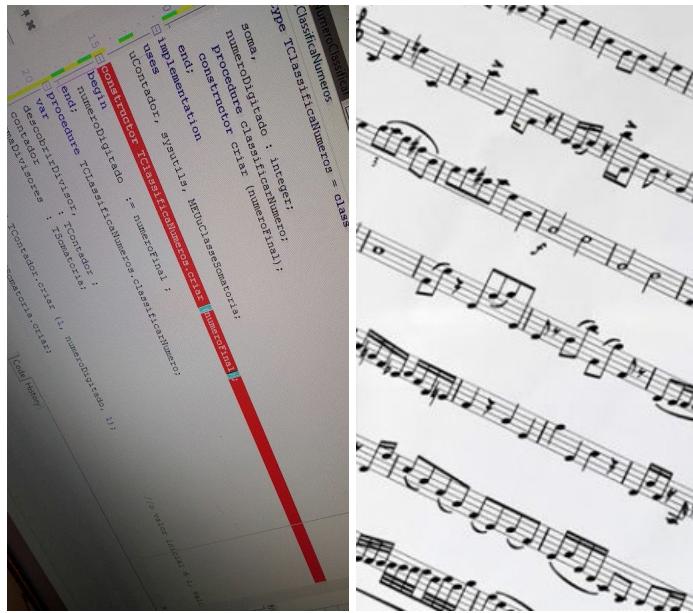
3.1 Leitura da entrada

Iniciamos o programa lendo o método a ser utilizado. Em seguida, para cada imagem PNG do diretório “images”, lemos a imagem. Usamos a biblioteca opencv para tal, e a flag cv.IMREAD_GRAYSCALE para indicar que a imagem deve ser lida como escala de cinza.

Python

```
mode = input('Mode: ')
for f in os.listdir('images'):
    image = cv.cvtColor(cv.imread(f'images/{f}'), cv.COLOR_BGR2GRAY)
if mode == 'proj':
    angle = horizontal_proj_technique(image)
elif mode == 'hough':
    angle = hough_technique(image)
else:
    raise NotImplementedError(f'Mode {mode} not implemented')
```

Há uma série de imagens já presentes em “images”, baixadas de um diretório disponibilizado pelo professor: https://www.ic.unicamp.br/~helio/imagens_inclinadas_png/. Porém, adicionei imagens pessoais também, como esta de uma foto do meu computador em 2017, fazendo meu primeiro programa, no colégio técnico em Pascal, aprendendo a programar. A imagem está bem rotacionada e um pouco apagada também. Abaixo, vemos a foto do programa feito por mim, “tcontador.png”, e “partitura.png”, disponibilizada pelo professor.



Figuras 1 e 2: Imagens de entrada tcontador.png e partitura.png.

3.2 Projeção Horizontal

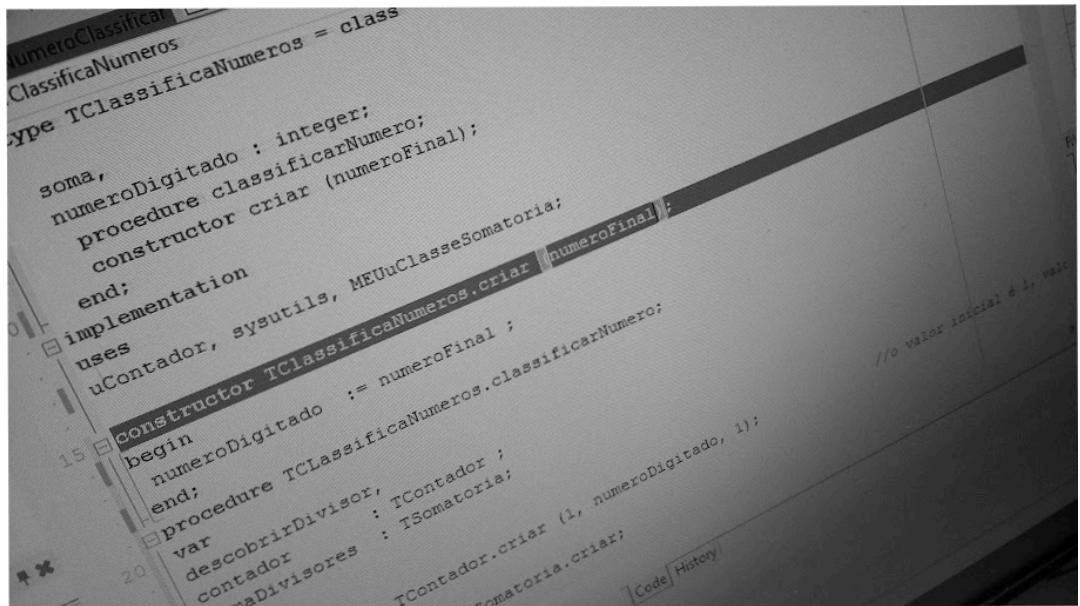
Como primeiro método para corrigir desalinhamento de documentos, utilizaremos a técnica de projeção horizontal. A detecção de inclinação neste método é realizada somando a quantidade de

pixels pretos em cada linha da imagem, para ângulos diferentes. Aquele ângulo que maximiza certa função objetivo, é o ângulo de inclinação que buscamos. A precisão deste método é controlada pela granularidade entre os ângulos testados, ou seja, 10 em 10 graus, 1 em 1 grau, etc.

Python

```
GRANULARITY = 360
def horizontal_proj_technique(img):
    angles = np.linspace(0, 360, GRANULARITY, endpoint=True, retstep=False)
    values = np.zeros_like(angles)
    for i in range(len(angles)):
        profile = np.sum(rotate_image(img, angles[i]), axis=1)
        values[i] = goal_function(profile)

    return angles[np.argmax(values)]
```



Figuras 3 e 4: Imagens após correção pela técnica de projeção horizontal.

Esta parte do algoritmo é composta de alguns passos. Primeiro, obtemos um vetor de ângulos que usaremos para realizar as projeções. Há uma constante: GRANULARITY, que indica a granularidade destes ângulos, ou, quantos ângulos, de 0 a 360, serão utilizados. Escolhemos 360, para testar ângulos de 1 em 1 grau.

Então, para cada ângulo α , rotacionamos a imagem em α , e armazenamos num vetor a soma dos pixels de cada linha da imagem rotacionada, aplicados a uma função objetivo, “goal_function”.

Python

```
def goal_function(profile):
    return np.max((profile[1:] - profile[:-1]) ** 2)
```

O argumento de entrada, “profile”, armazena as somas dos pixels em cada linha. Agora, paramos para pensar um pouco. Num documento bem alinhado, deve haver uma grande variação na soma entre linhas vizinhas, entre a parte que não há texto e a parte que há texto. Portanto, desejamos maximizar a diferença desta soma. Por exemplo, uma imagem na diagonal, vai ter diferenças menores do que uma imagem alinhada na horizontal. Por esse motivo, nossa função objetivo retorna o maior valor do quadrado da diferença entre duas linhas adjacentes. Este vai ser o valor atingido pelo ângulo α .

Python

```
return angles[np.argmax(values)]
```

Ao final da função da projeção horizontal, retornamos o ângulo cujo valor da função objetivo é maior, ou seja, o ângulo onde as diferenças de linhas adjacentes é máximo. Agora, vamos dar uma olhada na função “rotate_image()”.

Python

```
def rotate_image(image, angle):
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv.getRotationMatrix2D(center, angle, 1.0)

    abs_cos = abs(M[0, 0])
    abs_sin = abs(M[0, 1])

    new_w = int(h * abs_sin + w * abs_cos)
    new_h = int(h * abs_cos + w * abs_sin)

    M[0, 2] += (new_w / 2) - center[0]
    M[1, 2] += (new_h / 2) - center[1]

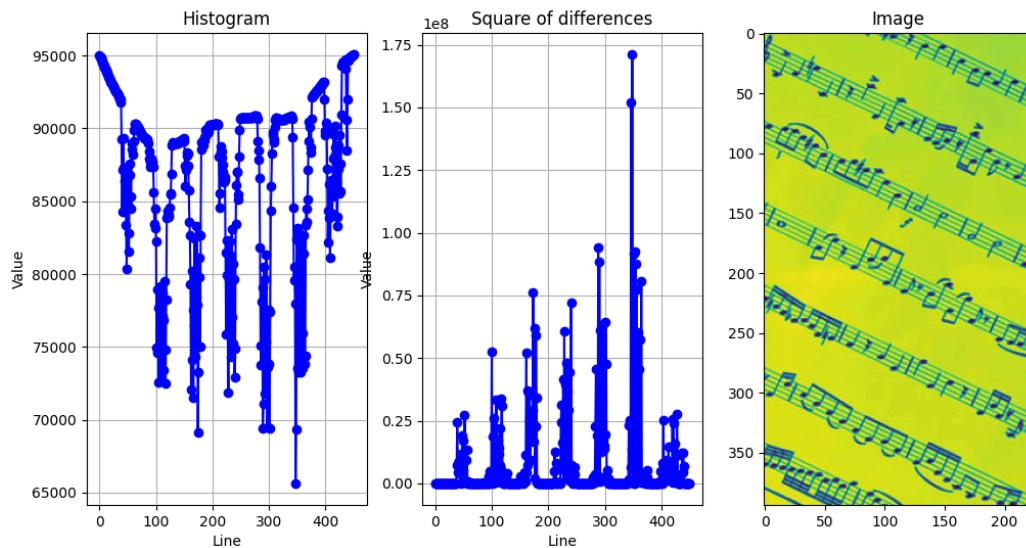
    return cv.warpAffine(image, M, (new_w, new_h), borderMode=cv.BORDER_CONSTANT,
borderValue=(255, 255, 255))
```

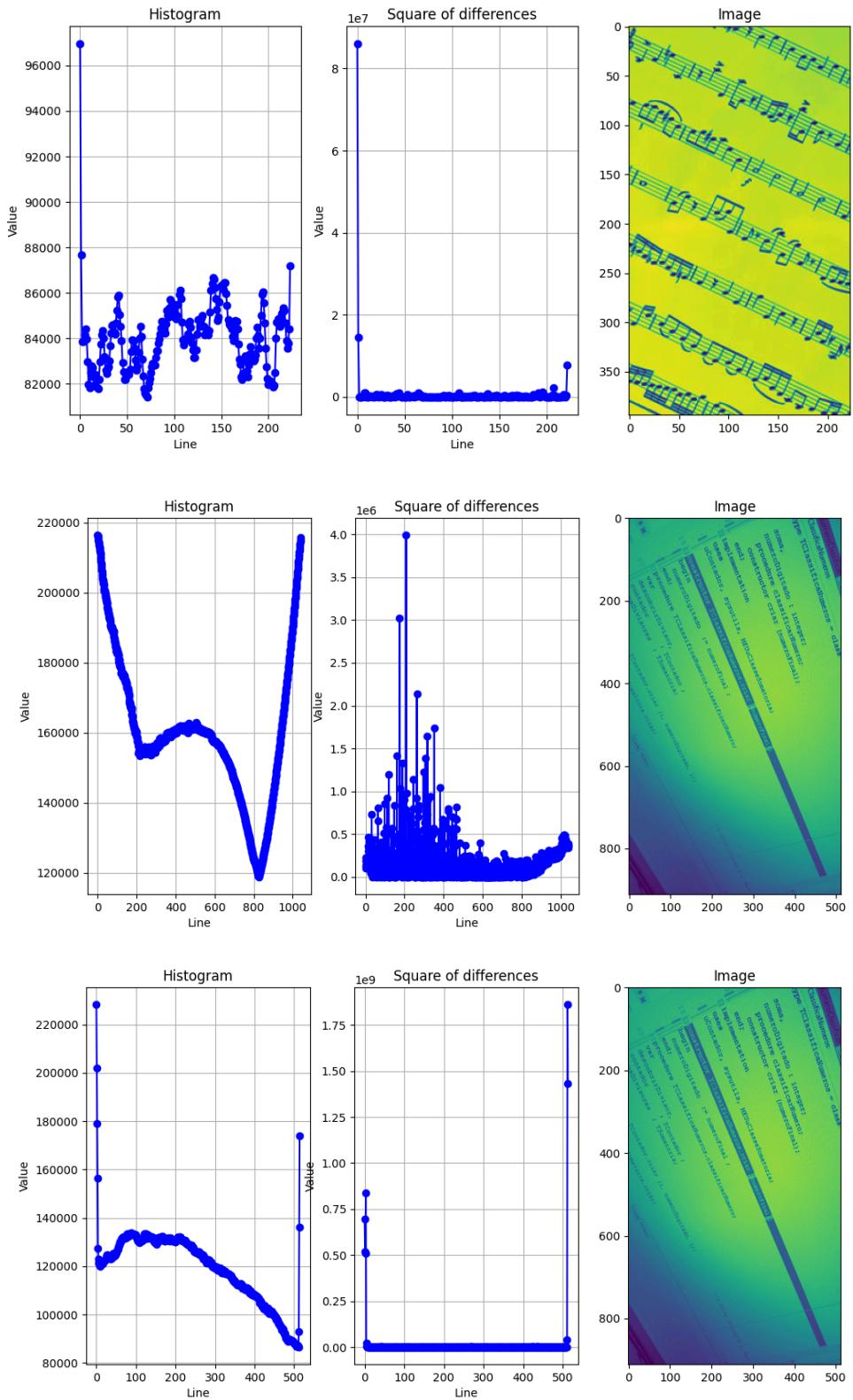
Nesta função, utilizamos uma função do opencv que calcula a matriz de rotação (senos e cossenos) a partir do ângulo, do centro da imagem e escala 1. Calculamos as novas dimensões da imagem a partir de contas matemáticas, usando os senos e os cossenos, pois, aplicaremos um padding na imagem rotacionada. Caso contrário, perderíamos informação. Então, calculamos o novo centro e só aí, usamos a função warpAffine do opencv, para de fato aplicar a rotação na imagem. Com os parâmetros utilizados, as novas bordas da imagem serão brancas.

Durante testes e validações, criei uma função “plot_img_histogram”, para exibir o histograma da projeção horizontal das imagens.

Python

```
def plot_img_histogram(img, hist):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 6))
    # Plot array values
    ax1.plot(np.arange(len(hist)), hist, marker='o', linestyle='-' ,
    color='b', label='Values')
    ax1.set_title('Histogram')
    ax1.set_xlabel('Line')
    ax1.set_ylabel('Value')
    ax1.grid(True)
    ax2.plot(np.arange(len(hist) - 1), (hist[1:] - hist[:-1]) ** 2,
    marker='o', linestyle='-' , color='b', label='Values')
    ax2.set_title("Square of differences")
    ax2.set_xlabel('Line')
    ax2.set_ylabel('Value')
    ax2.grid(True)
    ax3.imshow(img)
    ax3.set_title('Image')
    print(goal_function(hist))
    plt.show()
```





Figuras 5 a 8: Imagens após a exibição desta função. Na primeira, temos o ângulo certo de inclinação da partitura. Vemos que o maior valor da diferença dos quadrados estão na ordem de 10^8 . Num outro ângulo qualquer, o valor máximo é menor, na ordem de 10^7 . Para as imagens do contador, a primeira apresenta um ângulo qualquer, e na segunda, vemos que o valor máximo é extremamente maior. Esta função ajudou a visualizar os resultados e escolher uma função objetivo decente.

3.3 Transformada de Hough

Python

```
def hough_technique(img):
    edges = cv.Canny(img, 100, 150, apertureSize=3)
    lines = cv.HoughLinesWithAccumulator(edges, 1, 2*np.pi/GRANULARITY, 50)
    angle = lines[0, np.argmax(lines[:, :, 2])], 1
    return np.rad2deg(angle) - 90
```

Com um código extremamente conciso, podemos achar o ângulo de inclinação de um documento a partir da técnica da Transformada de Hough. Esta transformada gera uma nova matriz, onde os eixos, ao invés de serem os espaciais x e y, representam o sistema de coordenadas ρ , Θ . A ideia deste método é que retas podem ser representadas em coordenadas polares a partir da equação:

$$\rho = \frac{\rho_0}{\cos(\theta - \theta_0)}$$
, onde ρ , Θ são as coordenadas polares de um ponto da reta, ρ_0 é a distância perpendicular da reta à origem e θ_0 é o ângulo que a perpendicular à reta forma com o eixo polar.

Dessa forma, como a nova matriz é em coordenadas polares, todos os pontos da imagem original que estiverem numa mesma reta serão inseridos numa mesma célula. Cada célula da matriz é um contador, indicando quantos pontos da imagem original pertencem àquela reta. Dado um certo limiar, neste caso, 50, células com acumulador maior ou igual que tal limiar serão consideradas como retas na imagem.

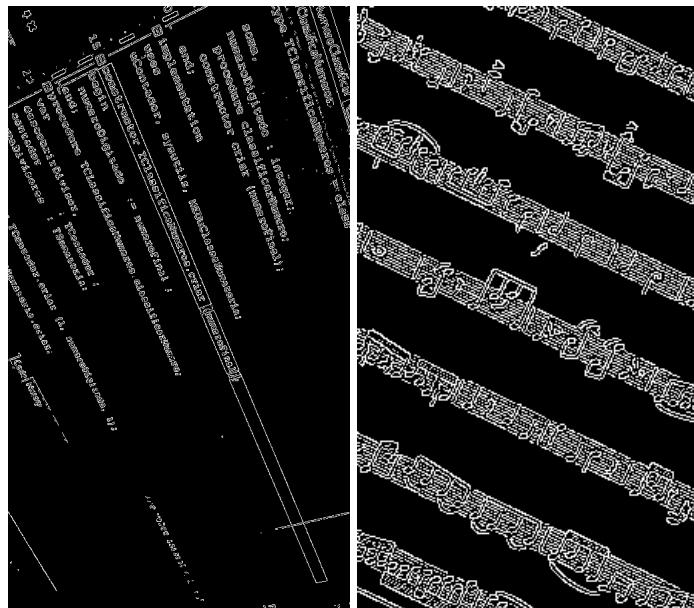
Por fim, um documento bem comportado teoricamente deveria ter uma grande presença de retas com o mesmo ângulo de inclinação, mesmo que ele esteja inclinado. Dessa forma, basta percorrermos a matriz de acumuladores e descobrir qual valor de Θ é mais presente na imagem. Este será o ângulo de inclinação.

Antes de tudo, aplicamos um algoritmo de detecção de bordas na imagem. Vamos utilizar o algoritmo de Canny para isso. A aplicação do algoritmo de Canny antes da Transformada de Hough na detecção de retas em uma imagem é um passo fundamental devido a várias razões. Primeiramente, o algoritmo de Canny é um detector de bordas eficiente que identifica áreas de transição rápida de intensidade na imagem. Estas áreas de transição correspondem às bordas dos objetos na imagem, que são os candidatos ideais para a detecção de retas pela Transformada de Hough.

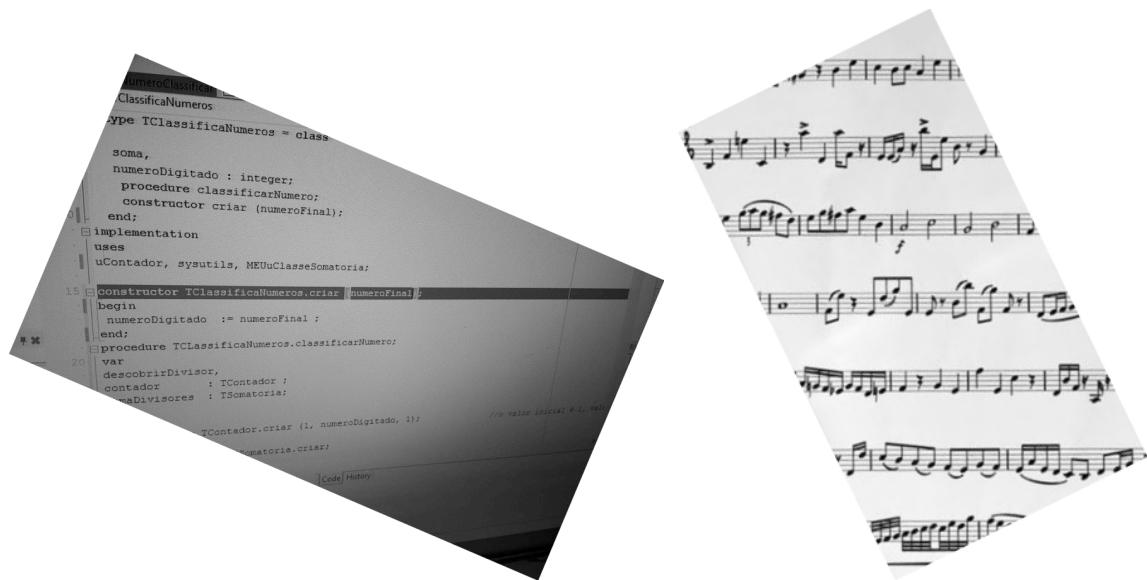
Ademais, o Canny inclui etapas de suavização da imagem, geralmente através de um filtro Gaussiano, para reduzir o ruído. A redução do ruído é crucial porque ele pode causar falsas detecções de bordas, o que levaria a falsas detecções de retas na Transformada de Hough. A Transformada de Hough é uma técnica poderosa, mas computacionalmente intensiva. Ela detecta retas ao transformar a imagem para um espaço de parâmetros onde as retas se tornam pontos de interseção. Aplicar o algoritmo de Canny reduz o número de pixels que precisam ser analisados pela Transformada de Hough, concentrando-se apenas nos pixels que estão nas bordas significativas.

O detector de bordas de Canny utiliza técnicas como a supressão de não-máximos e a histerese para garantir que as bordas detectadas sejam precisas e contínuas. Isso é importante para a Transformada de Hough, que precisa de bordas bem definidas para calcular com precisão os parâmetros das retas (como a inclinação e a interseção). Sem a aplicação prévia de um detector de bordas como o Canny, a Transformada de Hough teria que processar todos os pixels da imagem, o que seria extremamente ineficiente e levaria a um grande número de falsas detecções de retas. O Canny

reduz o conjunto de dados a ser analisado, aumentando a eficiência e a precisão da Transformada de Hough.



Figuras 9 e 10: Imagens apóos o algoritmo Canny de detecção de bordas.



Figuras 11 e 12: Imagens apóos a correção de inclinação pelo método com Transformada de Hough

3.4 Saída

Por fim, tendo o ângulo descoberto por uma das técnicas, usamos a função de rotacionar imagem para corrigir a imagem, e a salvamos na pasta “out”.

```

Python
path = f'out/{mode}'
os.makedirs(path, exist_ok=True)
cv.imwrite(f'{path}/{f}', rotate_image(image, angle))

```

4. Análise

Uma vez apresentada a visão geral do trabalho, vamos analisar os resultados mais a fundo, e verificar alguns casos.

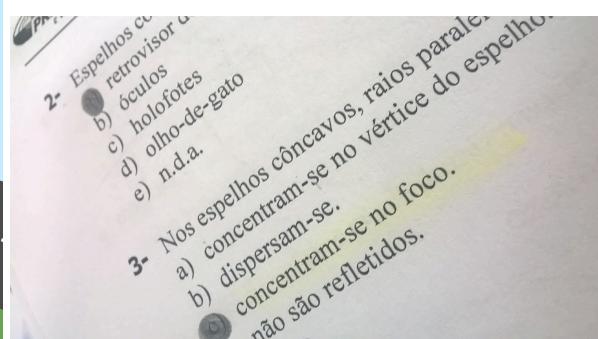
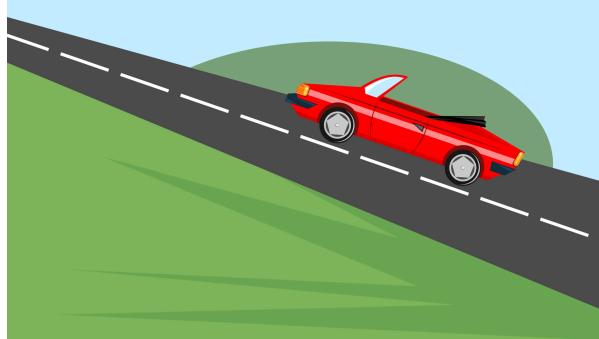
4.1 Imagens iniciais

Para as imagens iniciais apresentadas neste trabalho, “tcontador” e “partitura”, observamos que os resultados, principalmente de “tcontador” são mais certeiros na transformada de Hough. Poderia ser minimizado, talvez escolhendo funções objetivo mais precisas e complexas, porém, isso também complicaria o algoritmo, potencialmente também fazendo demorar ainda mais para executar. Além disso, como “tcontador” possui regiões indevidamente escuras, a soma das projeções horizontais pode não ter sido como esperado.

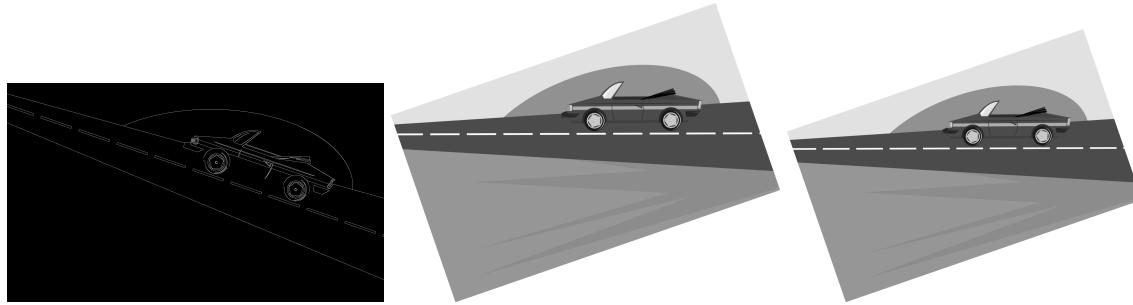
Dado isso, a Transformada de Hough se apresenta como um método bem melhor, pois, além de ser bem mais rápida (utilizando Canny previamente), é muito mais compacta. Na minha máquina pessoal, com 11 imagens de entrada, “proj_horiz” leva cerca de 6 segundos, enquanto “hough” termina em menos de 1 segundo, com resultados melhores. Isso confirma aquilo de que às vezes é melhor você mudar o sistema de coordenadas para conseguir resultados mais poderosos.

4.2 Outras imagens

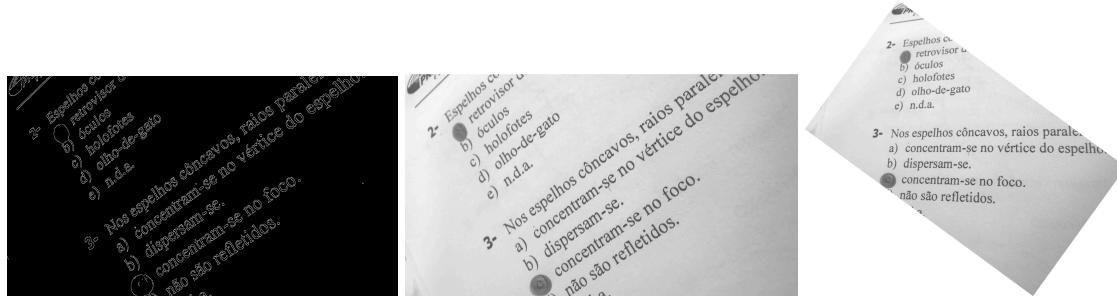
Vamos testar os algoritmos para outras imagens:



Figuras 13 e 14: Outras imagens para teste. Um carro numa rampa e uma prova teste.



Figuras 15 a 17: Da esquerda para a direita, Canny, Projeção Horizontal e Transformada de Hough.

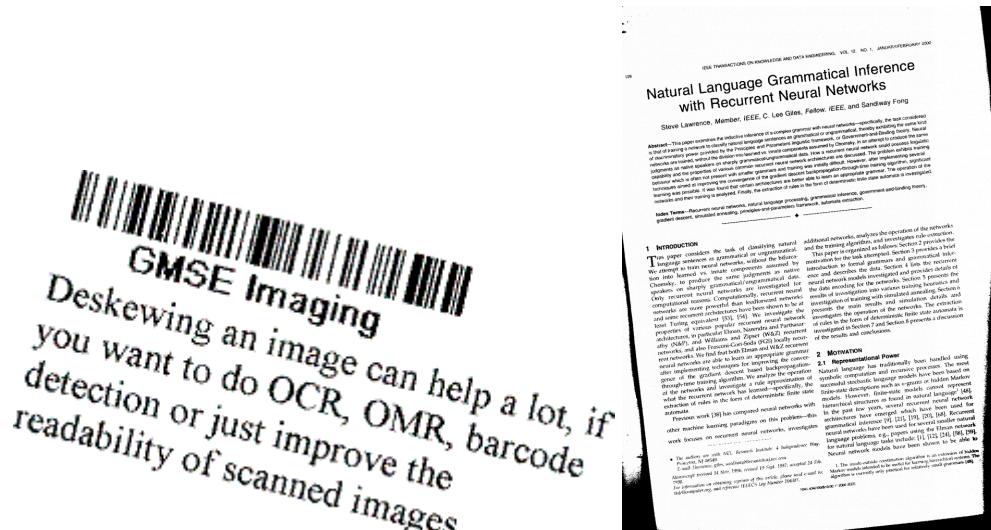


Figuras 18 a 20: Da esquerda para a direita, Canny, Projeção Horizontal e Transformada de Hough.

Podemos observar que ambos os métodos produziram um bom resultado na primeira imagem. Agora, o que antes era uma colina, parece uma estrada reta. Para a segunda imagem, a projeção horizontal não aplicou resultado algum aparentemente. Isso isso devido talvez a ruídos e regiões escuras nas bordas que atrapalharam a projeção. Para Hough, que teve Canny aplicado antes, o resultado foi bem aceitável, além de mais rápido.

4.3 Resultados para sample 1 e 2

No diretório apresentado pelo professor, temos as imagens sample1 e sample2, apresentadas abaixo.



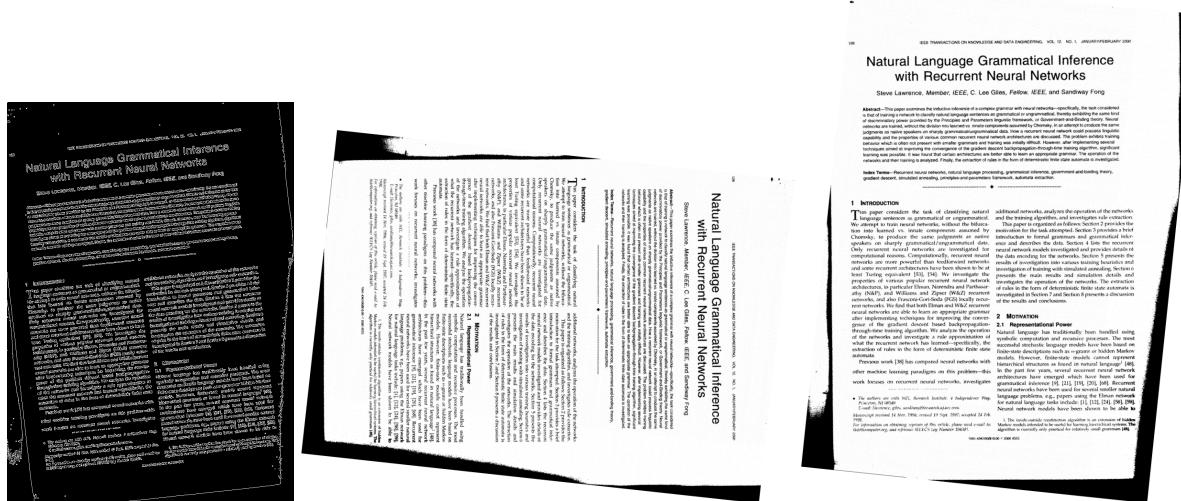
Figuras 21 e 22: Da esquerda para a direita, sample1 e sample2.

O desafio destas imagens é que, uma possui fortes retas verticais, do código de barras. Mesmo assim, o alinhamento deve seguir o texto, e não o código de barras. Na segunda, é uma folha inteira, com um fundo preto após a digitalização.



Figuras 23 a 25: Da esquerda para a direita, Canny, Projeção Horizontal e Hough.

Para sample1, vemos resultados bem satisfatórios, então a função objetivo não foi afetada pela presença do código de barras, e deu o resultado esperado. Hough também foi bem consistente. Vamos ver para sample2.



Figuras 26 a 28: Da esquerda para a direita, Canny, Projeção Horizontal e Hough.

Nesta imagem, para a projeção horizontal, a imagem ficou rotacionada. Isso não seria desejado, mas, o programa de fato alinhou a imagem. Porém, ficou desse modo pela presença do fundo preto. Há uma parte da imagem quase toda preta, o que faz com que a função objetivo maximize a rotação onde esses pontos se alinham, fazendo com que fique deitada. Para Hough, isso não foi um problema. Como o Canny removeu tais pontos desnecessários, as retas nem foram calculadas, resultando então no valor esperado de ângulo de inclinação.

5. Considerações

Por fim, observamos que é possível realizar a correção de inclinação de objetos digitalizados, de forma simples e compacta, produzindo resultados satisfatórios num tempo excelente. Entre os métodos abordados, a Projeção Horizontal é mais simples de entender teoricamente, porém é menos eficaz. Já a Transformada de Hough, tem uma teoria complicada, mas é simples de utilizar e bem eficiente, ainda mais combinada a outros métodos de processamento de imagens, como a detecção de bordas por Canny, por exemplo.

Observamos cada vez mais os grandes potenciais de aplicações de processamentos simples de imagens, que produzem resultados muito interessantes, que pareciam ser complexos e quase impossíveis de resolver. No começo do semestre, não sabíamos responder nem um método eficiente de se achar bordas e retas numa imagem. Agora, implementamos dois algoritmos de correção de inclinação de imagens digitais, o que é muito interessante e motiva os alunos, facilitando a assimilação da teoria com resultados práticos muito intrigantes.