# Relatório Trabalho 3

Raphael Adamski
227179

May 16, 2018

Este relatório é parte de uma série de trabalhos da disciplina *MO443 - Introdução ao Processamento de Imagem Digital* lecionada pelo Professor Hélio Pedrini na Universidade Estadual de Campinas no primeiro semestre de 2018.

## 1 Ambiente

O algoritmo desenvolvido necessita de um ambiente com Python 3.x instalado, além das bibliotecas *OpenCV*[1], *Numpy*[2], *imutils*, *PIL* e *Pytesseract*. Este ultimo é apenas uma interface para um comando do sistema operacional (SO), por isso deve estar instalado no SO a biblioteca *Tesseract Open Source OCR Engine* [3].

## 2 Algoritmo

Dado o ambiente como esperado, basta rodar o código fornecido como seguinte uso:

```
alinhar.py [-h] -i INPUT [-o OUTPUT]
[-m {projection,hough}]
[-p {crop,sobel,otsu,contours,gray}]
[-c CROP]
```

em quem o argumento `-i` é o endereço da imagem de entrada para ser encontrado o texto e alinhada, `-o` é o endereço para salvar a imagem de saída, `-m` é a técnica a ser utilizada para encontrar o ângulo de rotação da imagem (Hough ou Projeção Horaizontal), `-p` são os métodos de pré-processamento a serem utilizados (permitindo uma combinação de vários entre filtro de Sobel, Otsu Threshold, recortar, encontrar contornos) e `-c` é parâmetro para definir o tamanho da janela do recorte, caso selecionado no pré-processamento.(*crop*). Utilizando `-h` mostra-se o uso e descrição do programa. [*1*]

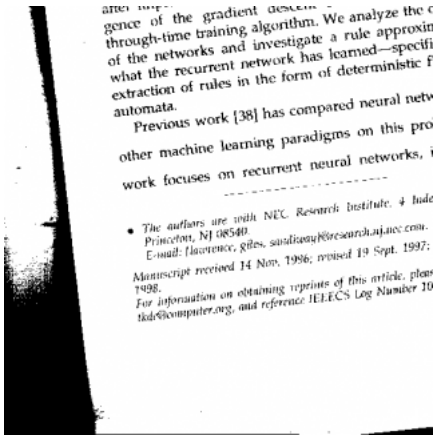
gence of the gradient descent
through-time training algorithm. We analyze the
of the networks and investigate a rule approxi
what the recurrent network has learned—specifi
extraction of rules in the form of deterministic
automata.
Previous work [38] has compared neural net
other machine learning paradigms on this pro
work focuses on recurrent neural networks,
The authors are with NEC Research Institute, 4 Inde
Princeton, NJ 08540.
E-mail: {lawrence, giles, sandiway}@research.nj.nec.com.
Manuscript received 14 Nov. 1996; revised 19 Sept. 1997;
1998.
For information on obtaining reprints of this article, plea
tkde@computer.org, and reference IEEECS Log Number 10

Fig. 1: Imagem de exemplo com bordas pretas fora da imagem.

### 2.1 Pré-processamento

No código foram implementadas quatro técnicas para esta etapa. Vale ressaltar que independentemente dos parâmetros de `-p` passados, sempre será feito uma conversão para escala de cinza (parâmetro `gray`), já que ambos algoritmos para encontrar o ângulo esperam imagens com uma dimensão de profundidade.

As técnicas estão ordenadas na ordem em que são apli-

[1] https://opencv.org/
[2] http://www.numpy.org/
[3] https://github.com/tesseract-ocr/tesseract

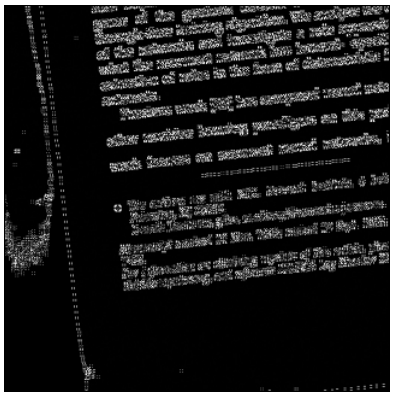

Fig. 2: Aplicação do filtro de Sobel na Figura 1.

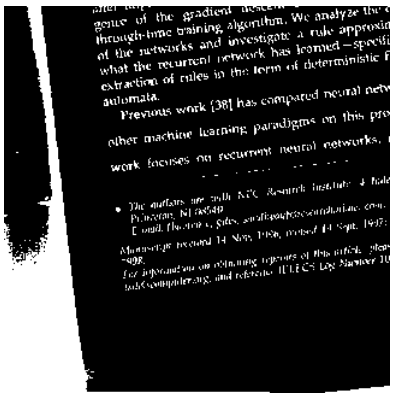

Fig. 3: Aplicação do Otsu threshold na Figura 1.

cadas:

### Recorte

O parâmetro `crop` quando usado permite recortar a imagem com uma janela de 500 pixels por padrão, localizada no centro da imagem. O tamanho da janela é configurável por `-c` seguido de um número inteiro.

### Filtro Sobel

Este filtro detector de bordas é usado quando é passado `sobel` na linha de execução do programa. Basicamente ele realça bordas usando gradiente da imagem. A função utilizada esta presente na biblioteca *OpenCV*. Na Figura 2 é possível visualizar o efeito da técnica.

### Otsu Threshold

Esta é uma técnica de binarização local da imagem, já implementada em *OpenCV* e é ativada usando parâmetro `otsu`. É possível ver seu efeito na Figura 3.

### Reconhecimento de Contornos

Usando `contours` é possível melhorar o processamento identificando previamente formas, como imagens e tabelas (vide 4 e 5). As mesmas são tiradas quando reconhecidas pela função `cv2.findContours` que implementa a técnica proposta por [1].

Quando a imagem esta deslocada e com bordas pretas, é


Fig. 4: Imagem com figuras com Otsu Threshold.


Fig. 5: Aplicação da remoção de contornos na Figura 4.

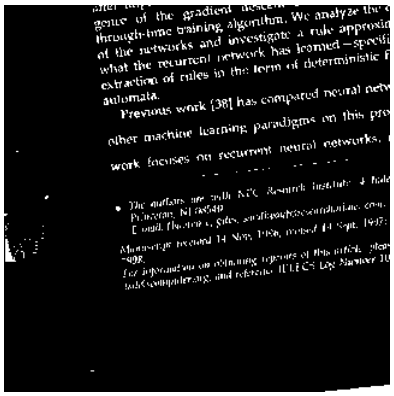

Fig. 6: Aplicação da remoção de contornos na Figura 1.

possível também reconhecer o formato da folha e retirar este fundo preto que poderia interferir no processamento. Como é possível ver comparando a Figura 1 e 6.

## 2.2 Detecção do Ângulo

Como é perceptível nas Figuras 3 e 4 , as imagens que passaram por binarização usando Otsu ou Sobel ficaram com o fundo preto, isso foi feito de maneria proposital pois ambos os métodos explicados a seguir funcionam mais adequadamente com pixels relevantes (neste caso letras) tendo valor, que neste caso seria 255 e cor branca.

### Projeção Horizontal

A técnica se baseia em rotacionar a imagem em 180 graus (pois não é possível prever se o texto esta invertido) e para cada ângulo calcular a soma dos pixels em cada linha, por isso a imagem com representatividade apenas nas letras ( fundo preto, letras brancas). O ângulo com a linha que somasse maior valor é escolhido como ângulo de correção da imagem.
Como é perceptível, fundo com cor igual das letras, como na Figura 1, e imagens tornam esse algoritmo muito suscetível a falhas. Além disso, a função utilizada para rotação (`imutils.rotate_bound`) preenche o fundo da imagem com preto, sendo este outro motivo para inverter a cor das letras e o fundo.

### Transformada de Hough

Esta transformada leva os pontos com intensidade maior que 10, na escala de 0 a 255, para outro plano discretizado por $\rho = x \times cos(\theta) + y \times sen(\theta)$. Usando o valor de $\theta$ da tabela do ponto que obtiver maior número de curvas cruzando, indicará o ângulo de inclinação do texto.
Este método é altamente suscetível a falhas com fundos com cor, como na Figura 1, pois todos os pontos claros (maior que 10) serão transformados para o plano de Hough e irão influenciar na escolha do ângulo. A Figura 7 mostra uma faixa amarela escura que representa a influencia errada do fundo da imagem, e que consequentemente leva a escolha errada do ângulo. Já na Figura 8 vemos claramente que o angulo de aproximadamente 84 graus esta acumulando vários pequenos pontos amarelos, com alta concentração de curvas cruzando, sendo assim a melhor escolha.

## 2.3 Detecção do Texto

É aplicado a técnica de *Optical Character Recognition* (OCR) antes e depois da rotação para mostrar casos em que houve melhoria. A função utilizada da do pacote citado foi `pytesseract.image_to_string`.

# 3 Saída

Durante a execução é mostrado a imagem original ao usuário em uma janela, e em seguida as imagens de cada processo de pré-processamento selecionado. Por fim a imagem original é rotacionada negativamente com o ângulo de inclinação encontrado. O console mostra o texto antes e depois da rotação além do ângulo de rotação para melhorar a imagem.
A imagem rotacionada é salva caso na execução tenha sido passado o argumento `-o` com o caminho para o arquivo.

# 4 Testes

Os testes mostraram que ambas as técnicas de reconhecimento da inclinação têm os mesmos resultados, contudo a Transformada de Hough implementado pelo autor precisa ser otimizada, pois o tempo de execução é muito maior

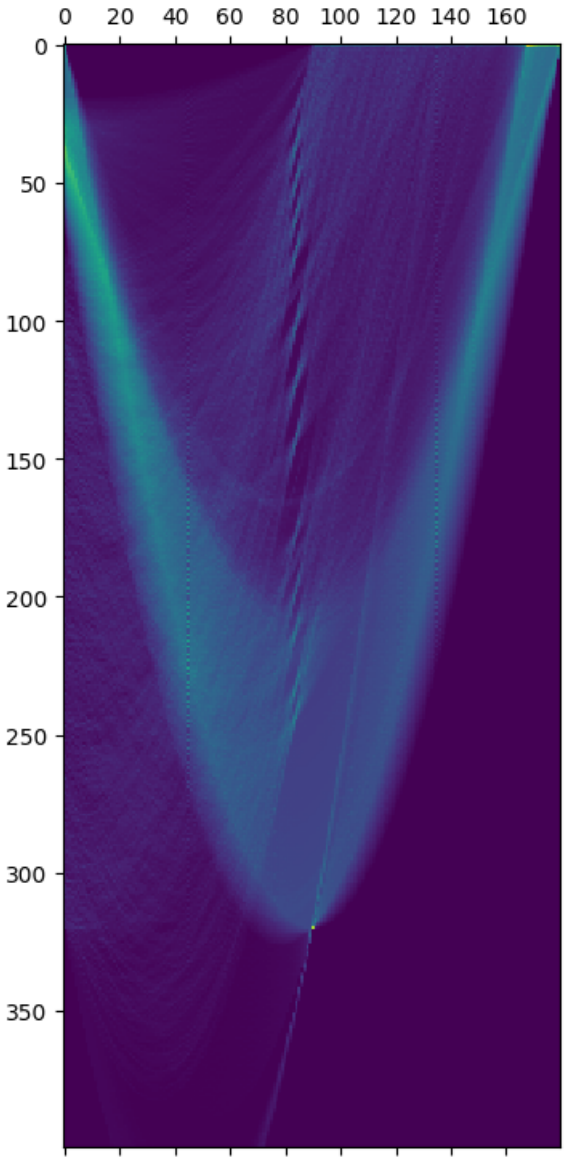


Fig. 7: Transformada de Hough dos pixels da Figura 3 (sem remoção do fundo).

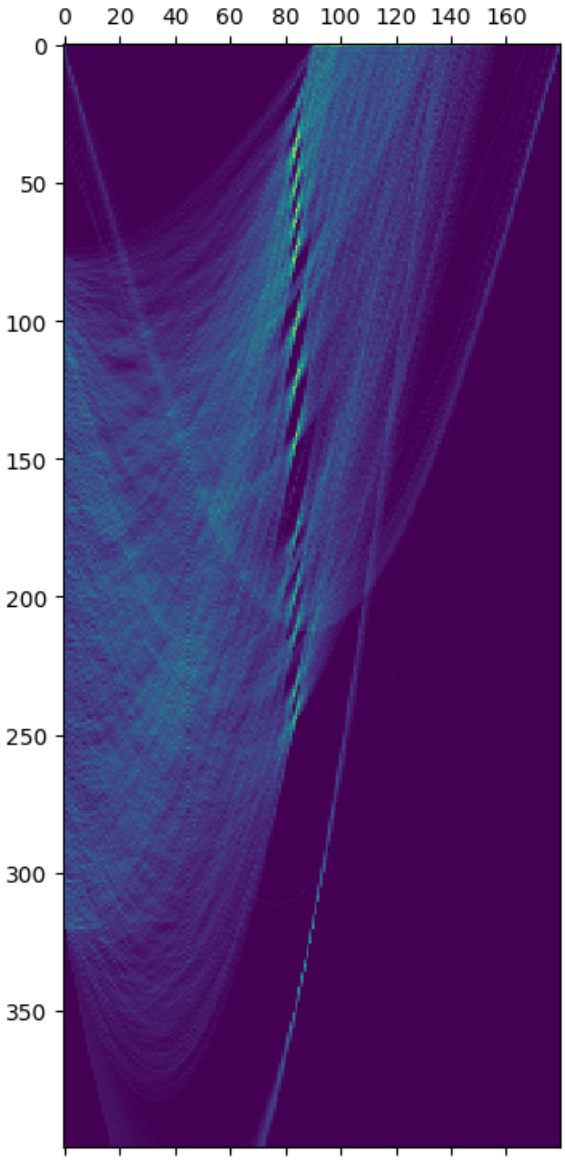


Fig. 8: Transformada de Hough dos pixels da Figura 6 (com remoção do fundo).

que a projeção horizontal.
Apenas o uso de binarização (Otsu ou Sobel) foi bem eficaz em imagens com fundo sólido. Porém em testes com imagens de pior qualidade ou estruturas que não fossem apenas texto, como Figura 1 e 4, o desempenho dos algoritmos ficava distante do ideal.
Aplicando recortes no centro da imagem em alguns casos ajudou a evitar bordas com problemas, contudo caso o recorte concentrasse em uma imagem, como a Figura 4, existia ainda uma dificuldade para processamento.
Então aplicando a técnica da reconhecimento de contorno foi possível obter sucesso nas imagens de teste usando filtro de Sobel e projeção horizontal.

## 5 Condições não tratadas

Além de não serem tratados os casos em que o ambiente não está estabelecido como descrito acima, vale ressaltar que não se verifica se o endereço da imagem passado como parâmetro é válido, o que pode resultar em erros de execução devido a objetos vazios.
A pasta do local de saída da imagem gerada pelo algoritmo de codificação deve existir, caso contrário a mesma não é salva.

## 6 Condições tratadas

Imagens podem ser coloridas ou em escala de cinza e de diferentes extensões, já que a biblioteca trata a maioria dos formatos[4]. O arquivo de saída não precisa existir, pois é criado caso não exista, com a extensão *.png*. Se existir e tiver algum conteúdo, o mesmo é apagado.

## 7 Considerações finais

O arquivo entregue contem um diretório *.git* em que é possível verificar a evolução do código pelos *commits*[5]. Também existem imagens para teste oriundas de https://www.ic.unicamp.br/~helio/imagens_inclinadas_png/.

[4] https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html
[5] https://git-scm.com/docs/gittutorial/2.2.0

## Referências

[1] Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985) 2