MC920 - Trabalho 3

Tobias Conran Zorzetto, RA: 166214

April 2023

1 Introdução

Esse trabalho tem como objetivo aplicar uma série de operações morfológicas para segmentação de regiões compreendendo *texto* e *não texto* em uma imagem de entrada. Nesse contexto, esse relatório tem como objetivo apresentar cada passo e de cada operação feita sobre a imagem, trazendo com si uma análise a respeito das imagens resultantes.

Assim, o texto está dividido em 3 seções: **Introdução**, **Execução**, onde será explicado de maneira geral como foi feito o programa, sua estrutura e as bibliotecas utilizadas, e **Passos e Resultados**, onde serão apresentados os resultados de cada passo pedido pelo trabalho.

2 Execução

Para a resolução dos passos enunciados no trabalho 3, alguns padrões foram adotados. Primeiramente, todos os passos foram realizados em um único Júpiter Notebook entitulado `trabalho3.ipynb`. Além disso foram utilizadas 3 bibliotecas em geral para a resolução: a biblioteca *imageio* foi utilizada para ler a imagem original, salvando-a em um vetor do *numpy*; a bibliotecta *opencv* foi utilizada para realizar as operações morfológicas na imagem; por fim, a biblioteca *matplotlib.plt* foi utilizada para apresentar as imagens resultantes de cada passo.

A imagem original utilizada foi a imagem `bitmap.pbm`, que pode ser vista na figura 1. È importante dizer, porém, que essa imagem foi invertida para melhor funcionamento do programa, já que a biblioteca *opencv* entende os pixels pretos como fundo e os pixels brancos como objeto.

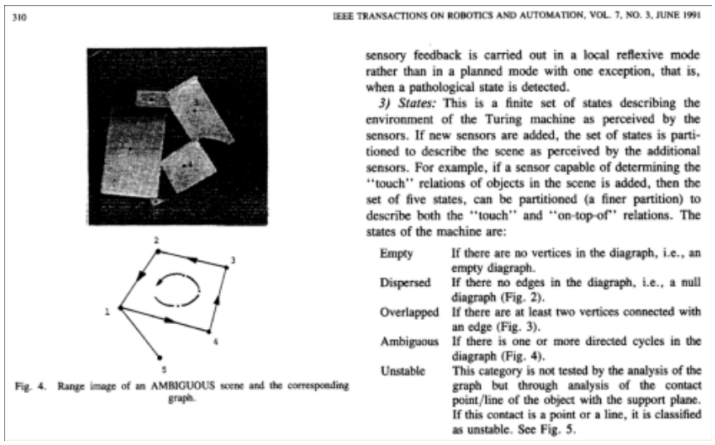

Figura 1: bitmap.pbm

3 Passos e Resultados

3.1 Passo 1

O primeiro passo consistiu na dilatação da imagem original com um elemento estruturantes de 1 pixel de altura e 100 pixels de largura. Isso pôde ser realizado com a função `dilate()` do *opencv*.

A imagem resultante pode ser vista na figura 2. Vê-se que com essa dilatação com elemento estruturante longo, ela já foi capaza de segmentar de forma mais "grossa"as linhas e figuras que compôe a imagem.

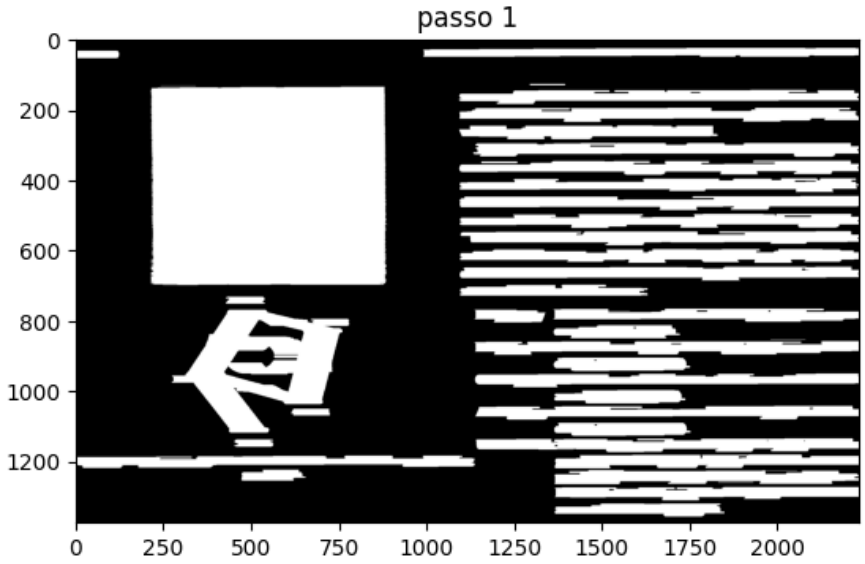

Figura 2: imagem resultante do passo 1

3.2 Passo 2

O segundo passo consistiu da erosão da imagem resultante do passo 1 com mesmo elemento estruturante. Isso pôde ser realizado com a função `erode()` do *opencv*.

A imagem resultante pode ser vista na figura 3. A erosão seguida realizada sobre a dilatação anterior faz uma operação de fechamento nas imagens. Assim, têm-se como resultado a segmentação das linhas e imagens na figura, porém com maior definição de espaço ocupado que no passo anterior.

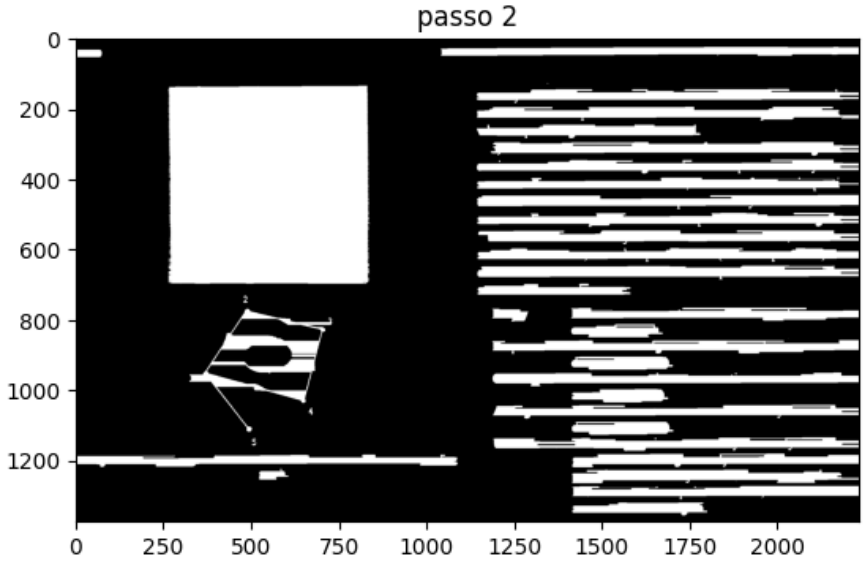

Figura 3: imagem resultante do passo 2

3.3 Passo 3

O terceiro passo consistiu na dilatação da imagem original com um elemento estruturantes de 200 pixels de altura e 1 pixel de largura.

A imagem resultante pode ser vista na figura 4. È possível ver um alongamento vertical dos objetos na imagem, enquanto, em comparação com a imagem resultante do passo 1 que faz um alongamento horizontal. Como o objetivo é encontrar textos, tal operação parece ser menos efetiva que a do passo 1.

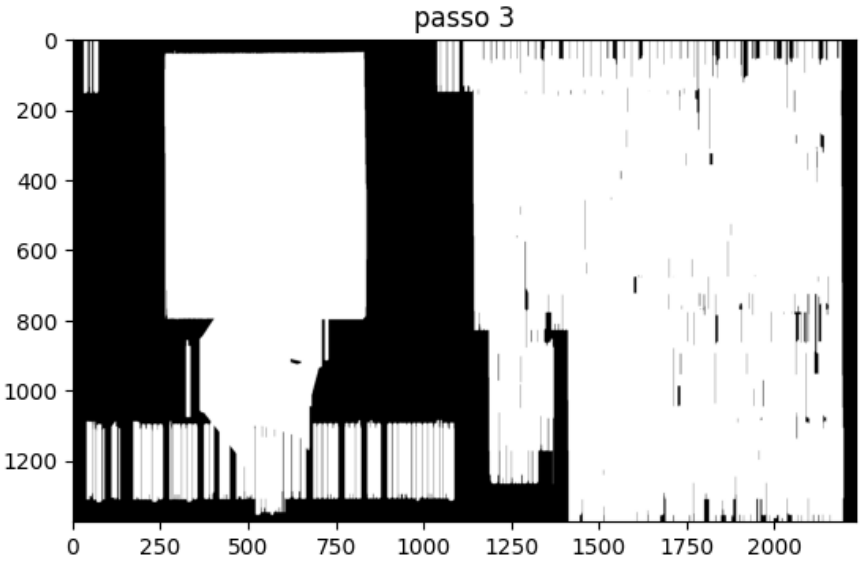

Figura 4: imagem resultante do passo 3

3.4 Passo 4

O quarto passo consistiu da erosão da imagem resultante do passo 3 com mesmo elemento estruturante.

A imagem resultante pode ser vista na figura 5.

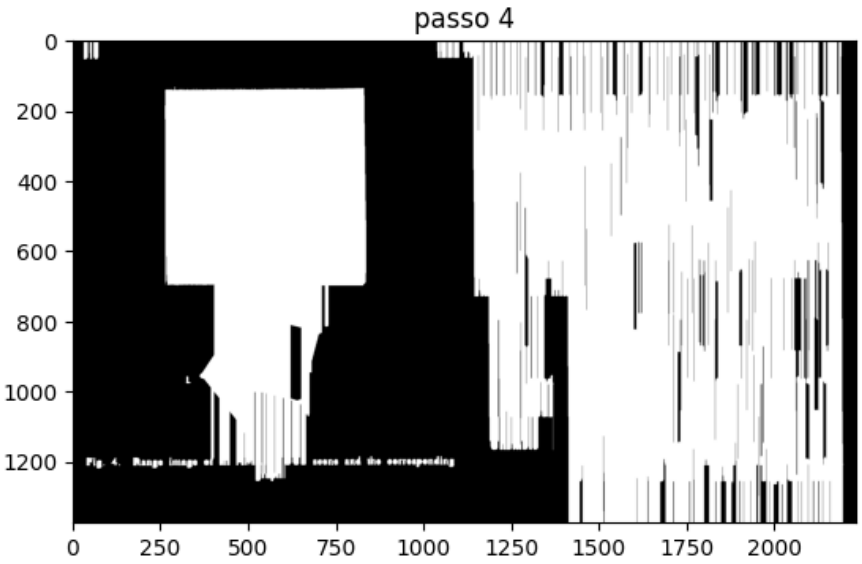

Figura 5: imagem resultante do passo 4

3.5 Passo 5

O passo 5 consistiu na itersecção das imagens resultantes do passos 2 e 4 (figuras 3 e 5). Isso pôde ser realizado com a função `bitwise_and()` do *opencv*.

A imagem resultante pode ser vista na figura 6. Com essa intersecção pode ser vista uma maior definição das regiões com textos e figuras na imagem, superior às obtidas nos passos individuais que formaram essa nova imagem e definidas em todas as dimensões, apesar disso, o resultado ainda parece ruidoso, sendo difícil identificar linhas de texto por completo.

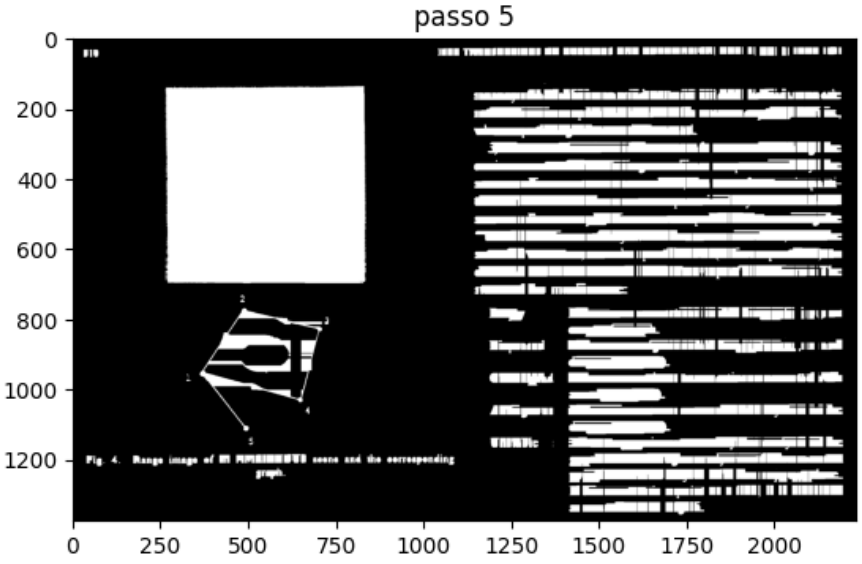

Figura 6: imagem resultante do passo 5

3.6 Passo6

O passo 6 consistiu no fechamento da imagem obtida no passo anterior, a partir de um elemento estruturante de 1 pixel de altura e 30 pixels de largura.

A imagem resultante pode ser vista na figura 7. Com esse fechamento, parece ter sido resolvido o problema da imagem anterior de identificação das linhas por completo. Assim, as linhas de texto parecem estar bem conexas.

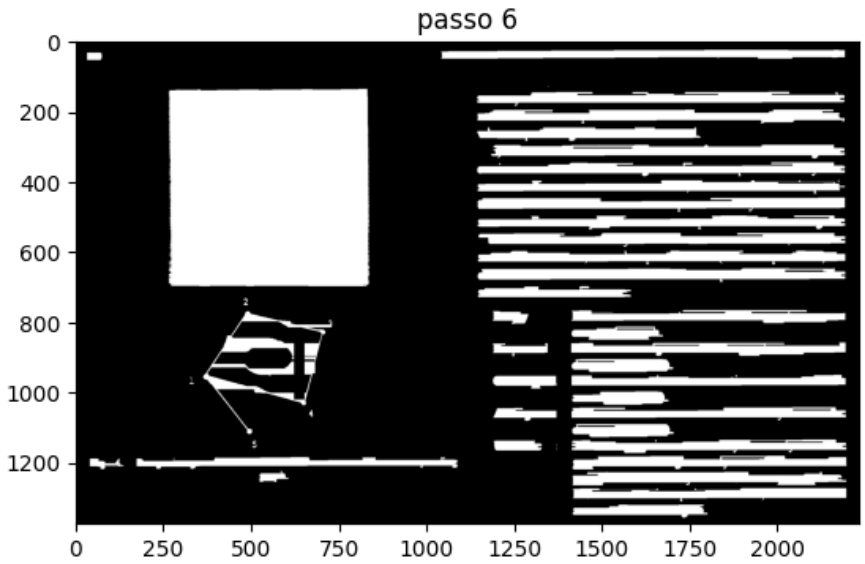

Figura 7: imagem resultante do passo 6

3.7 Passo 7

Para o sétimo passo, aplicado um algoritmo de identificação de componentes conexos sobre a imagem do passo anterior.

Para isso, foi tuilizada a função `connectedComponentsWithStats` do *opencv*, que retorna o número de componentes, uma imagem rotulada com os labels, e varias estatísticas como área, centroide, posições de origem e altura e largura. Dado isso, para identificar os componentes em uma imagem, basta percorrer pelos componentes e utilizar as estatísticas para calcular onde deveria estar o retângulo que o delimita. Fazendo esse cálculo, basta desenhar cada retângulo em uma cópia da imagem com o comando `rectangle`, também do *opencv*.

A imagem resultante pode ser vista na figura 8.

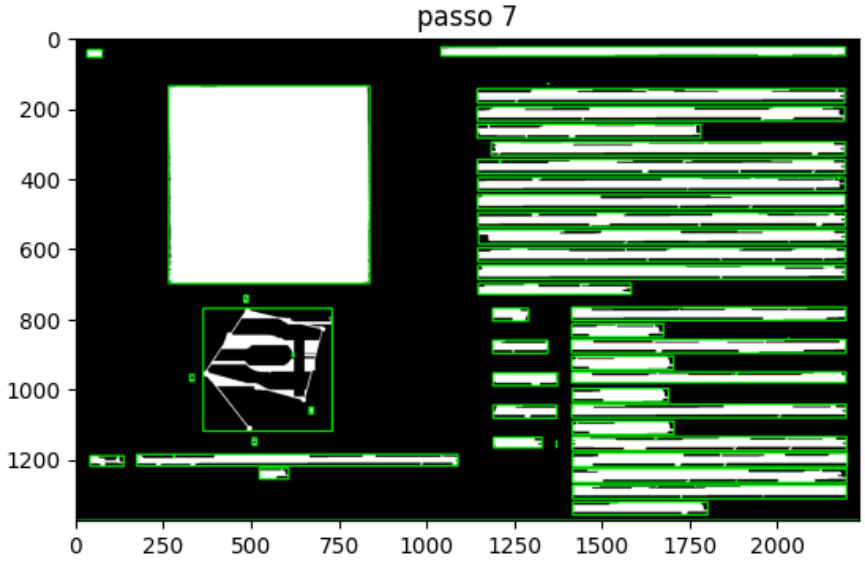

Figura 8: imagem resultante do passo 7

3.8 Passo 8

Para o oitavo passo, foram calculadas: a razão entre os números de pixels que representam um objeto e o número total de pixels e a razao entre o número de transições verticais e horizontais de *objeto para não objeto* e o numero total de pixels de objeto.

Para o primeiro, bastou utilizar a área de cada componente dada nativamente pelas estatísticas do comando utilizado no passo anterior e dividir pela área do retângulo, que pode ser calculada pela altura do componente vezes a largura do componente. Assim, a a figura 9 apresenta as razões obtidas para cada componente.

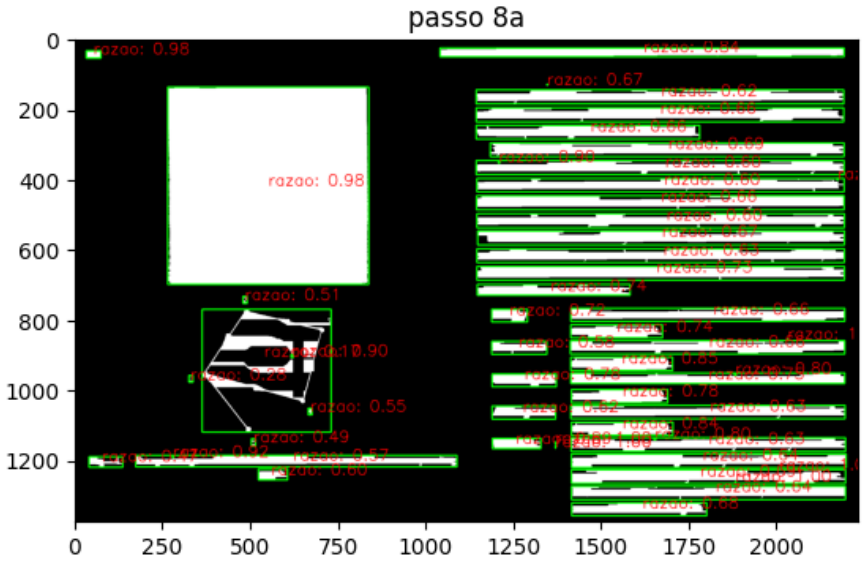

Figura 9: imagem resultante do passo 8 com as razões de pixel que representam objeto de cada componente

Para a segunda razão basta percorrer todos os pixels da imagem, incrementando uma contagem toda vez que o próximo pixel for maior que o anterior, tanto nas colunas quanto nas linhas. Assim, a razão de transição pode ser calculada dividindo essa contagem pela altura do componente vezes a largura do componente. Assim, a a figura **??** apresenta as razões de transição obtidas para cada componente.

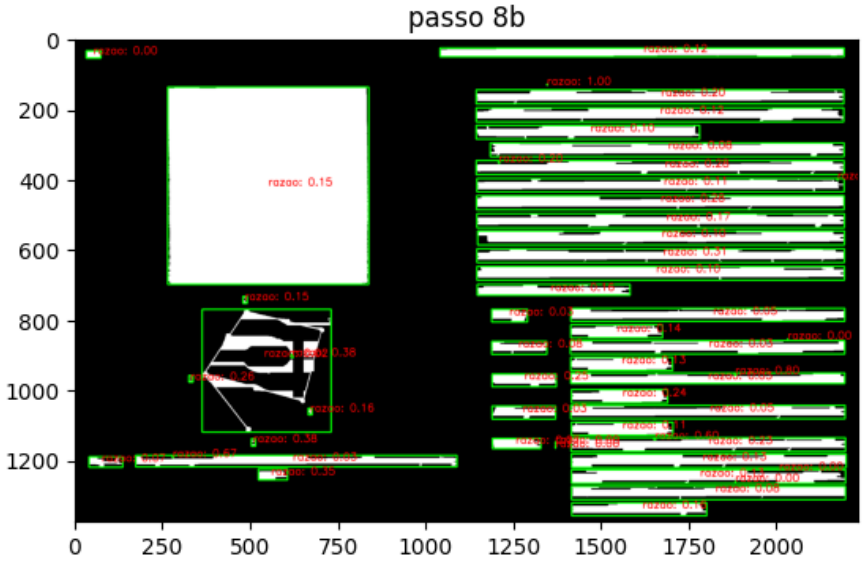

Figura 10: imagem resultante do passo 8 com as razões de transição de cada componente

3.9 Passo 9

Para o nono passo, deve-se criar uma regra para classificar cada componente conexo em texto ou não texto. Isso pode ser feito a partir da análise das imagens resultantes do passo 8, analisando as diferentes razões para cada componente.

A partir dessa análise, pode-se criar uma regra de que a razão de pixels de objeto deve estar entre 0.47 e 0.9. Também pode-se utilizar da razão de transição para eliminar ruídos da seguinte forma: $0 < razão_{transição} < 1$. Por fim, também podemos exigir que para que seja considerado um texto, o componente deve ter uma área maior que 200 pixels.

Assim, conseguimos determinar os componentes que representam texto. Assim, utilizamos novamente a imagem original para demonstrar como essa classificação se encaixa à imagem original. O resultado disso está na figura 11. Foram encontradas 34 linhas.

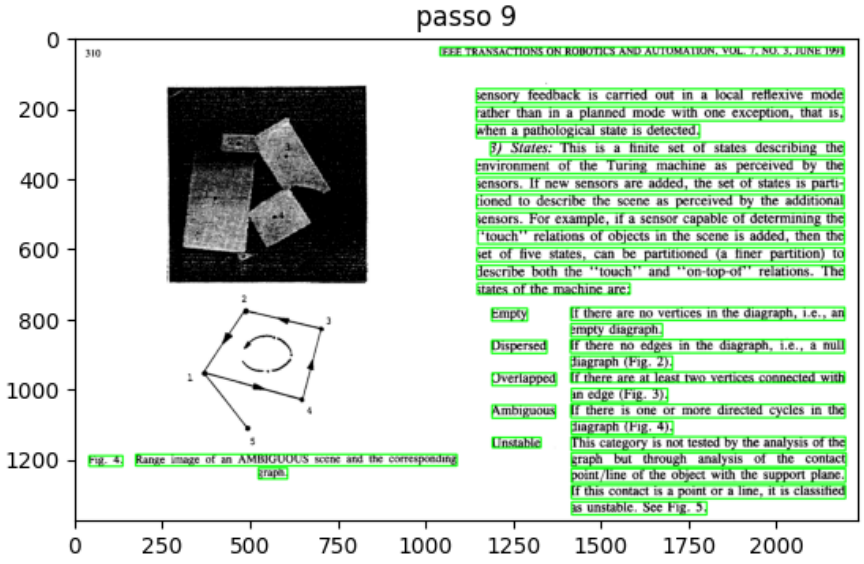

Figura 11: imagem resultante do passo 9

3.10 Passo 10

Para o último passo, foi pedido que agora se segmente a imagem em blocos de palavras.

Para isso, pode-se fazer uma síntese do que foi feito nos passos anteriores, porém alterando os parametros para segmentação de palavras no lugar de linhas. Para isso, faz-se primeiramente um fechamento com um elemento estruturante de 1 pixel de altura e 10 pixels de largura, para juntar os objetos que estão separados por menos de um espaço em branco. Depois, em cima da imagem, faz-se um fechamento com um elemento estruturante de 10 pixels de altura e um pixel de largura para conectar acentos às palavras respectivas.

Depois disso, com os componentes conexos, basta identificá-los e aplicar uma regra utilizando as razões descritas no passo 8. Para palavras, as condições que se mostraram apropriadas para classificar um componente conexo como um bloco de palavras foram $0.2 < razão_{pixelspretos} < 0.95$ e $0 < razão_{transições} < 1$, além de uma área maior que 70 pixels, para eliminar ruídos menores que o tamanho de uma letra.

O resultado dessas operações pode ser visto na imagem da figura 12. Vê-se que em geral o programa foi capaz de reconhecer blocos de palavras, errando com uma frequência muito pequena. Foram encontradas 244 palavras.

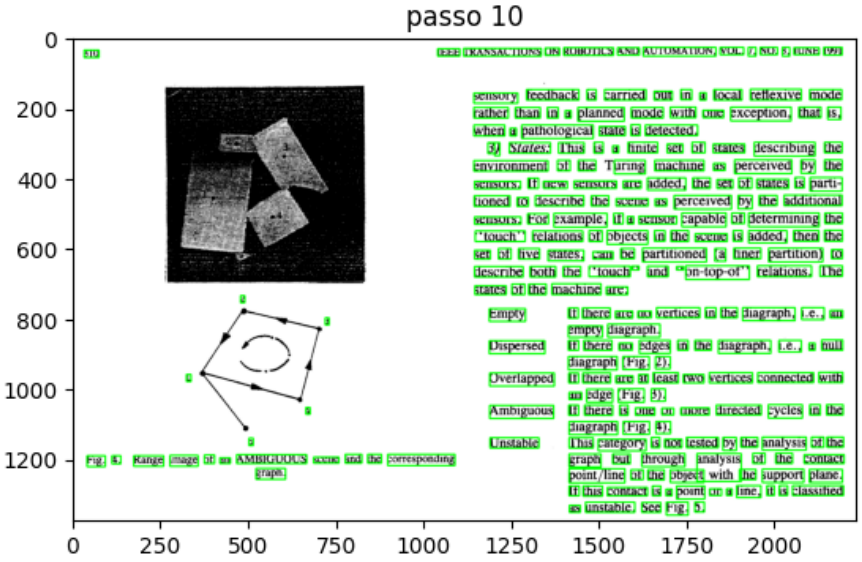

Figura 12: imagem resultante do passo 10