2024-1-Projeto PIM

César Eduardo de Souza, Lucas Ziemann Ferreira

[1]Departamento de Ciência da Computação - Universidade do Estado de Santa Catarina (UDESC)
Caixa Postal 631 – 89.219-710 – Santa Catarina – SC – Brazil

cesar.souza@edu.udesc.br, lucas.ziemann@edu.udesc.br

Abstract. *This project shows the implementation of image segmentation techniques, including limits, identification of connected components and labeling, using Python.*

Resumo. *Nesse projeto é mostrada a implementação de técnicas de segmentação de imagens, incluindo limiarização, identificação de componentes conectados e rotulação, utilizando Python.*

1. Sobre o trabalho

1.1. Introdução

O trabalho a seguir envolve o estudo da aplicação de segmentação por área e operações preparatórias (limiarização, identificação de componentes conexos, rotulação etc), conforme estudado em sala. Colocando na prática uma implementação na linguagem Python, não utilizando bibliotecas como OpenCv, por exemplo. Porém foram usadas algumas bibliotecas básicas que estarão mencionadas na seção requisitos.

1.2. Requisitos

1. Python 3
2. Numpy
3. Matplotlib
4. PIL

2. Código principal

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math

def threshold(image):
    img_array = np.array(image)
    threshold = np.mean(img_array)
    while True:
        background_pixels = img_array[image <= threshold]
        foreground_pixels = img_array[image > threshold]

        media_background = np.mean(background_pixels)
```

```
        media_foreground = np.mean(foreground_pixels)

        novo_threshold = (media_background +
        ↪  media_foreground) / 2

        if abs(novo_threshold - threshold) < 0.1:
            break

        threshold = novo_threshold
    return threshold

def bfs(start, visited, rows, cols, imagem):
    queue = [start]
    component = []
    visited.add(start)
    while queue:
        x, y = queue.pop(0)
        component.append((x, y))
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < cols and 0 <= ny < rows and (nx,
            ↪  ny) not in visited and imagem[ny][nx] > 0:
                queue.append((nx, ny))
                visited.add((nx, ny))
    return component

def biggest(image):
    visited = set()
    components = []
    image = np.array(image)
    rows, cols = image.shape
    for j in range(cols):
        for i in range(rows):
            if image[i][j] > 0 and (i, j) not in visited:
                component = bfs((j, i), visited, rows,
                ↪  cols, image)
                components.append(component)
    biggest = max(components, key=len)
    return biggest

def gp_img(group):
    ret = [[0 for _ in range(l)] for _ in range(h)]
    for c in group:
        ret[c[1]][c[0]] = 255
    return
    ↪  Image.fromarray((np.array(ret)).astype(np.uint8))
```

```
56
57  def border_cleaner(bimg):
58      rows, cols = bimg.shape
59      visited = set()
60
61      def bfs_border(root):
62          queue = [root]
63          component = []
64          border = False
65          while queue:
66              x, y = queue.pop(0)
67              component.append((x, y))
68              visited.add((x, y))
69              if x == 0 or x == cols-1 or y == 0 or y ==
                ↪  rows-1:
70                  border = True
71              for dx, dy in [(-1, 0), (1, 0), (0, -1), (0,
                ↪  1)]:
72                  nx, ny = x + dx, y + dy
73                  if 0 <= nx < cols and 0 <= ny < rows and
                    ↪  (nx, ny) not in visited and bimg[ny,
                    ↪  nx] > 0:
74                      queue.append((nx, ny))
75                      visited.add((nx, ny))
76          return component, border
77
78      for y in range(rows):
79          for x in range(cols):
80              if bimg[y, x] > 0 and (x, y) not in visited:
81                  component, border_touching = bfs_border((x,
                    ↪  y))
82                  if border_touching:
83                      for (cx, cy) in component:
84                          bimg[cy, cx] = 0
85
86      return bimg
87
88  def center(group):
89      x_all = [c[0] for c in group]
90      y_all = [c[1] for c in group]
91      x_center = np.mean(x_all)
92      y_center = np.mean(y_all)
93      return (x_center, y_center)
94
95
96  path="assets/solda.png"
```

```
image=Image.open(path).convert('L')
l,h=image.size

# encontra o threshold
threshold = threshold(image)

#torna a imagem binária a partir do threshold
bimg = np.where(np.array(image)>threshold, 255, 0)

#usa bfs nas bordas para remover os grupos que encostam
 ↪  nela
bimg = border_cleaner(bimg)

#encontra o maior grupo usando bfs
biggest = biggest(bimg)

#encontra o centro de massa do grupo
c = center(biggest)

#transforma o array em imagem
img = gp_img(biggest)

#define a imagem que será mostrada
plt.imshow(img, cmap='gray')

#plota uma letra O na cordenada do centro de massa definida
 ↪  por c[0](x) e c[1](y)
plt.plot(c[0], c[1], 'ro')

#exibe a imagem
plt.show()
```

3. Breve Explicação do Fluxo do Programa

Na pasta assets se encontra a imagem *solda.png* que será carregada no programa e convertida para o modo ”Luminance”por meio da biblioteca de imagem PIL.

Portanto, é encontrado o *threshold* da imagem pela média entre as intensidades dos pixels da imagem. Com ele é gerada a imagem binária, ou seja, com pixels apenas de 0 ou 255, que facilitará a abordagem para o resto do projeto.

Assim sendo, como solicitado pelo problema, é feita a retirada dos grupos de pixels brancos que atingem os limites da imagem, usando uma função bfs modificada, que guarda a informação de toque na borda de cada pixel.

Por fim, é executada a função padrão de BFS a fim de formar uma lista de grupos dos quais será selecionado apenas o maior, por meio da função *max* usando *key=len* como argumento para selecionar apenas o grupo com maior quantidade de pixels!

E como cereja do bolo é encontrado o centro de massa do grupo usando a função

mean do numpy.

Por fim, é plotada a imagem com um ”o”no centro encontrado, usando a biblioteca matplotlib.

Referências