PROJETO 3 – Componentes Conectados

João Vitor Rodrigues Baptista
15/0013329
Universidade de Brasilia
Faculdade Gama
Brasilia, Brasil
jvrbaptista@live.com

***Abstract*— O presente relário esta exposto as atividades teoricas e computacionais da materia de processamento digital de imagens da faculdade Gama com respeito a implementação de rotulação de regiões.**

Keywords—Imagens, Computational, rotulação, vizinhança.

I. INTRODUÇÃO

A avaliação de imagens digitais é possivel aplicar técnicas que visa a avaliação da relação entre pixel vizinhos. Conceitos pode ser separados em vizihança de pixel, adjacência, conectividade, regiões e fronteiras. Dentre as ténicas de avaliação de vizinhança existem os metodos de 4-vizinho e 8-vizinho, que basicamente a valia respectimante os 4 vizinhos – superior, inferior, esquerdo e direito – do pixel, já a tecnica de 8-vizinhos considera os 4-vizinho mais os pixel diagonal em relação do pixel de interesse.

Baseado nesses conceitos, é possivel avaliar regiões de interesse em uma imagem que são delimitados por fronteiras. As fronteiras, basicamente, é o local onde um pixel de interesse tem uma intensidade muito discrepente em relação ao pixel vizinho, resultando de duas regiões diferentes.

Essa tecnica pode ser utlizada para segmentar e separar regiões em uma imagem de interesse para que se possa analisar as fronteiras de interesse. Para o presente relatório foi desenvolvida tecnicas para a contagem de componentes e buracos de uma imagem baseada nos 8-vizinhos do pixel que esta contido na imagem.

II. EXPERIMENTO

No presente relatório esta exposto a teoria e a simulações computacionais referentes a rotulação de componentes por aldoritimos de identificação de vizinhanças.

Todas as simulações computacionais foram desenvolvidas em python 3 utilizando diversas bibliotecas de manipulação e plote de imagens e o editor Spyder. Foi desenvolvidas funções de rotulação de imagens por diferença de cores em cada região.

III. RESULTADOS

Os resultados obtidos foram conforme o esperado apresentado, contudo por falta de conhecimento tecnico não foi possivel desenvolver um algoritimo para classificar os tipos de manchas em tipo A ou tipo B. Sera apresentado nas seções apropriadas com uma breve discurção da teoria utilizada para o desenlvimento da metodogia e as particularidades de implementação computacional.

Foi desenvolvido uma classe com sete funções basicas de manipulação de imagens. Em seguida foi criado uma função que aplica de forma apropriada todos os metodos da classe. Enfim, basta carregar as imagens setar um limear para a binarização e aplicar na função principal o retorno dessa função é a imagem rotulada.

A. Resultados para a fig1.jpg

Aplicado a imagem fig1.jpg na função previamente criada, porem antés de aplicar e feita a binarização da imagem o resultado retornado da função é a imagem a seguir:

Figura 1 - Resultado da rotunação da fig1.jpg

É possivel notar que a função fez a distinção entre os componentes da imagem com cores diferentes. Para automatizar o numero de componentes de uma imagem baste verificar o numero de intensidades distintas de pixel na imagem rotulada e subtrair 1, o nivel de intensidade do fundo da imagem.

Na implentação é feita uma conversão do objecto Imagem para Array e usando uma função para verificar o numero de elementos distintos e subtrair de 1, assim pode-se saber o numero de componentes uma imagem tem.

```
[ ]  len(uniqueValues) - 1

     3
```

Figura 2 - Numero de componentes diferentes

Para fazer a contagem de buracos na imagem foi feito primeiro o negativo da imagem e então aplicada na função de rotunação. A função devolve os componentes com a mesma cor, porem os buros com cores diferentes, então para contar o numero de buracos na imagem é feita a conversão de objecto Imagem para Array e então contado o numero distinto de cores que a imagem possuir e subtraido de 2, pois tem a cor do fundo e a cor dos componentes.

Na imaplentação foi observado o seguinte comportamento.

Figura 3 - Imagem com rotulação nos buracos e componentes de mecom a mesma cor

```
[13] uniqueValues_holes = np.unique(gray_holes)
     len(uniqueValues_holes) - 2

     3
```

Figura 4 - Numero de buracos na imagem

Não foi implementada a classificação dos componentes, porem a sobreposição das imagem rotuladas produz uma nova imagem com componentes e buracos de cores distintas.

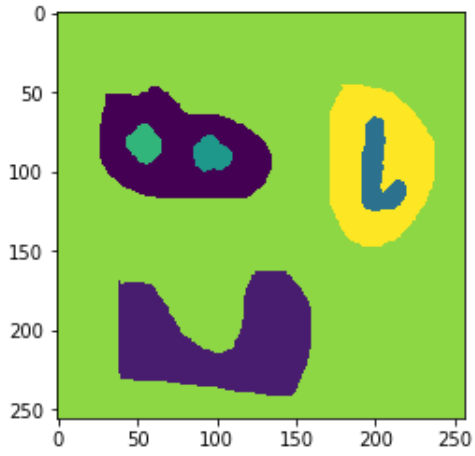

Figura 5 - Sobreposiçaõ das imagems de rotulação

Vizualmente é percepitivel que a imagem de um componente de tipo A que não possuir buracos e dois componentes de tipo B, um componente com dois burados e outro com um.

B. Resultados para a fig2.jpg

Foi feito o mesmo procedimento para a fig2. A seguir será apresentado os resultados.

Foi feita as mesmas conversões e aplicou-se as mesmas funções da imagem anterior.

Figura 6 - Rotulação das figura 2 é possivel verificar quatros cores diferentes

A imagem de rotulação foi convertida para Array e aplicando uma função para saber o numero de intensidades de pixel distintas tem a imagem e subtraindo uma das intensidades, já que a imagem possui fundo.

```
[76] len(uniqueValues) - 1

     4
```

Figura 7 - Numero de componentes da figura 2

Fazendo o negativo de imagem é possivel fazer a distinção entre os buracos da imagem. Convertendo a imagem para Array e aplicando a função de niveis de pixel distinto e subtraindo de dois, pois a imagem de fundo e as cores dos componentes. A seguir são apresentados os resultados.

Figura 8 - Imagem rotulada da figura 2 para identificar os buracos

```
[123] uniqueValues_holes = np.unique(gray_holes)
      len(uniqueValues_holes) - 2

      3
```

Figura 9 - Numero de buracos distintos

Foi feita a sobreposição das imagens rotuladas o que produz uma imagem rotunada com todos os elementes distintos e o fundo.

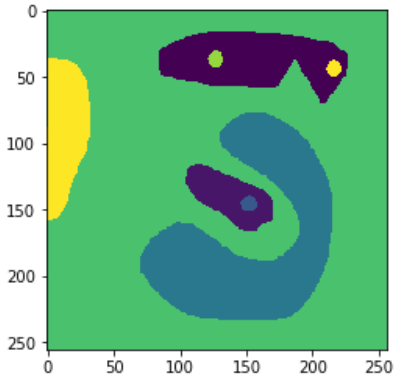

Figura 10 - Sobreposição das imagens rotuladas.

Fica fazil notar que essa imagem tem dois componentes de tipo A e dois componentes de tipo B, sendo um com dois buracos e outro com apenas um.

C. Resultados para a fig3.jpg

Foi feito as mesmas considerações feitas nos sub itens anterior. Segue os resultados:

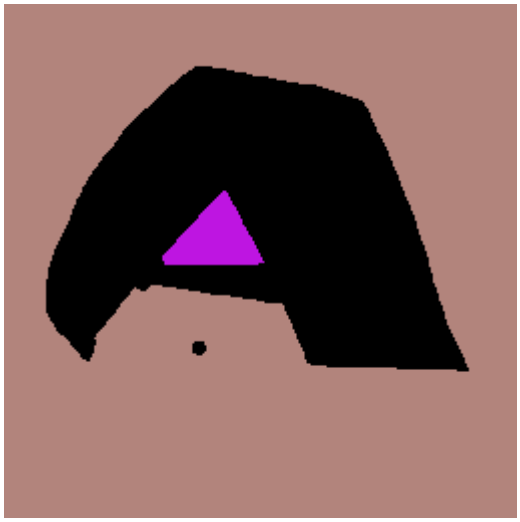

Figura 11 - Imagem rotulada com buracos de cores distintas

Figura 12 - Imagem rotulada com componentes de cores distintas

```
[138] len(uniqueValues) - 1

      2
```

Figura 13 - Numero de componentes distintos baseado nas cores

```
[149] uniqueValues_holes = np.unique(gray_holes)
      len(uniqueValues_holes) - 2

      1
```

Figura 14 - Numero de buracos distintos baseados nas cores

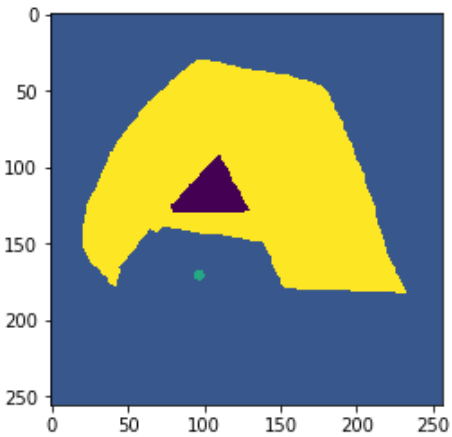

Figura 15 - Sobreposição das imagens rotuladas

Fazendo a sobreposição das imagens rotuladas o resultado é uma nova imagem onde todos os elementos vizinhos distintos possuir cores diferentes. Fica facil notar que existe dois componentes um do tipo A e outro do tipo B.

D. Resultados para a fig4.jpg

Foi feitos as mesmas considerações do passo anterior para a figura 4. A seguir os resultados

```
[162] len(uniqueValues) - 1

      6
```

Figura 16 - Numero de componentes distintos

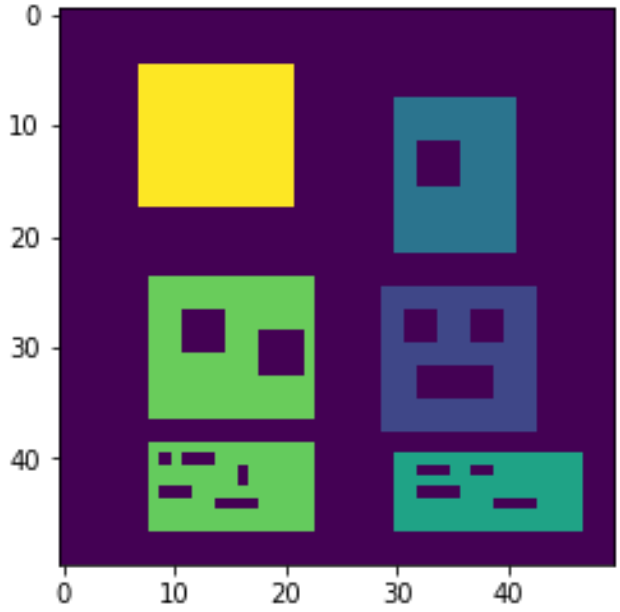

Figura 17 - Imagem rotulada da figura 4

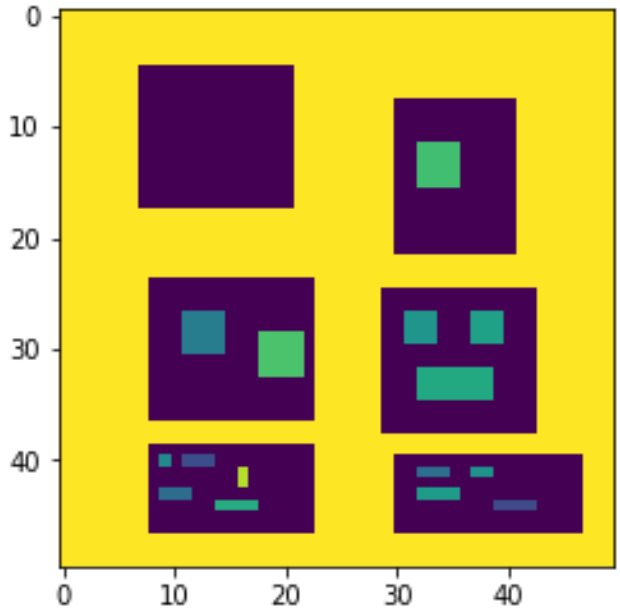

Figura 18 - Imagem rotudala dos buracos

```
[171] uniqueValues_holes = np.unique(gray_holes)
      len(uniqueValues_holes) - 2

      15
```

Figura 19 - Numero de buracos distintos na imagem

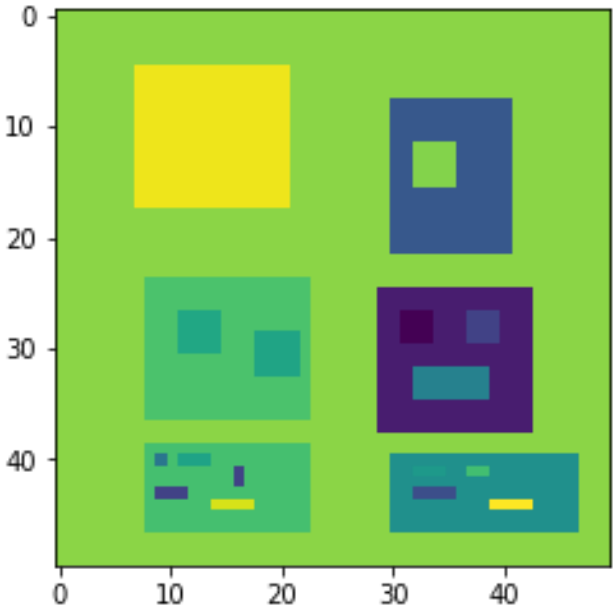

Figura 20 - Sobreposiçaõ da imagens rotuladas

É possivel notar que a imagem possui mais buracos e componentes que as imagens anteriores. Existe uma imagem de tipo A e o restante é do tipo B, com respectivamente da esquerta para direita e de cima para baixo, um buraco, dois buracos, três buracos, cinco buracos e quatro buracos.

E. Quadro de resumo dos resultados

Tabela 1 – Quadro de resumo dos resultados

	NM	NB
Fig1.jpg	3	3
Fig2.jpg	4	2
Fig3.jpg	2	1
Fig4.jpg	6	15

DISCURSÃO

Apensar de não ser implementada a classificação dos componentes os resultados de contagem de componentes e buracos esta conforme o esperado. É possivel notar que essa ténica pode ser utilzada para fazer contagem de elementos desejados de uma imagem apartir da diferênciação em cores.

Computacionalmente são operações de varredura e atribuização de pesos em matrizes pequenas, porem para larga escala a tecnica de 8 vizinhos pode necessitar de mais recursos computacionais comparada com usando a de 4 vizinhos.

Para melhorar a identificação de buracos seria melhor uma tecnica para excluir a borda dos componentes. Isso poderia ser alcançado fazer operações entre as imagens.

REFERENCIAS

[1] Gonzalez, Rafael C. Processamento digital de imagens / Rafael C. Gonzalez e Richard C. Woods ; revisão técnica : Marcelo Vieira e

Mauricio Escarpinati ; [tradução Cristina Yamagami e Leonardo Piamonte]. -- 3. ed. -- São Paulo : Pearson Prentice Hall, 2010..

CODIGO FONTE

```
from PIL import Image, ImageDraw
import sys
import math, random
from itertools import product

class UFarray:
    def __init__(self):
        self.P = []
        self.label = 0

    def makeLabel(self):
        r = self.label
        self.label += 1
        self.P.append(r)
        return r

    def setRoot(self, i, root):
        while self.P[i] < i:
            j = self.P[i]
            self.P[i] = root
            i = j
        self.P[i] = root

    def findRoot(self, i):
        while self.P[i] < i:
            i = self.P[i]
        return i


    def find(self, i):
        root = self.findRoot(i)
        self.setRoot(i, root)
        return root

    def union(self, i, j):
        if i != j:
            root = self.findRoot(i)
            rootj = self.findRoot(j)
            if root > rootj: root = rootj
            self.setRoot(j, root)
            self.setRoot(i, root)

    def flatten(self):
        for i in range(1, len(self.P)):
            self.P[i] = self.P[self.P[i]]

    def flattenL(self):
        k = 1
        for i in range(1, len(self.P)):
            if self.P[i] < i:
                self.P[i] = self.P[self.P[i]]
            else:
                self.P[i] = k
                k += 1

def run(img):
    data = img.load()
    width, height = img.size

    uf = UFarray()

    labels = {}

    for y, x in product(range(height), range(width)):

        if data[x, y] == 255:
            pass

        elif y > 0 and data[x, y-1] == 0:
            labels[x, y] = labels[(x, y-1)]

        elif x+1 < width and y > 0 and data[x+1, y-1] == 0:

            c = labels[(x+1, y-1)]
            labels[x, y] = c

            if x > 0 and data[x-1, y-1] == 0:
                a = labels[(x-1, y-1)]
                uf.union(c, a)

            elif x > 0 and data[x-1, y] == 0:
                d = labels[(x-1, y)]
                uf.union(c, d)

        elif x > 0 and y > 0 and data[x-1, y-1] == 0:
            labels[x, y] = labels[(x-1, y-1)]

        elif x > 0 and data[x-1, y] == 0:
            labels[x, y] = labels[(x-1, y)]

        else:
            labels[x, y] = uf.makeLabel()

    uf.flatten()

    colors = {}

    output_img = Image.new("RGB", (width, height))
    outdata = output_img.load()

    for (x, y) in labels:

        component = uf.find(labels[(x, y)])

        labels[(x, y)] = component

        if component not in colors:
            colors[component] = (random.randint(0,255),
random.randint(0,255),random.randint(0,255))

        outdata[x, y] = colors[component]

return (labels, output_img)

# Inicializar a imagem
img = Image.open('/content/fig4.jpg')
img.show()
```

```
img = img.point(lambda p: p > 190 and 255)
img = img.convert('1')

(labels, output_img) = run(img)

output_img.show()

output_img

type(output_img)

type(output_img)

pix.shape

uniqueValues = np.unique(pix)
uniqueValues

import cv2
import matplotlib.pyplot as plt
gray = cv2.cvtColor(pix, cv2.COLOR_BGR2GRAY)
plt.imshow(gray)

gray.shape

uniqueValues = np.unique(gray)
uniqueValues

len(uniqueValues) - 1

# Open the image
img = Image.open('/content/fig4.jpg')
img.show()
# img = np.array(img)
# img = cv2.bitwise_not(img)
# img = Image.fromarray(img)

# Threshold the image, this implementation is designed to process b+w
# images only
img = img.point(lambda p: p <= 190 and 255)
img = img.convert('1')

# labels is a dictionary of the connected component data in the form:
#     (x_coordinate, y_coordinate) : component_id
#
# if you plan on processing the component data, this is probably what you
# will want to use
#
# output_image is just a frivolous way to visualize the components.
(labels, output_img_holes) = run(img)

output_img_holes.show()

output_img_holes

import numpy as np
import cv2
import matplotlib.pyplot as plt

pix_holes = np.array(output_img_holes)
gray_holes = cv2.cvtColor(pix_holes, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_holes)

uniqueValues_holes = np.unique(gray_holes)
len(uniqueValues_holes) - 2

new_img = cv2.addWeighted(gray_holes, 1, gray, 1, 0)
plt.imshow(new_img)
```