

# Tarefa-1 – Segmentação por Área

André Felipe Fuck<sup>1</sup>, Kauan Henrique Werlich<sup>1</sup>,

<sup>1</sup> Universidade do Estado de Santa Catarina - UDESC  
Centro de Ciências Tecnológicas - CCT. Joinville - SC - Brasil CEP: 89.219-710

andre.fuck@udesc.edu.br, kauanhenrique.werlich@gmail.com

## 1. Descrição da tarefa

A presente tarefa tem como objetivo o estudo da aplicação do método de processamento de imagem conhecido como segmentação por área e as operações de pré-processamento necessárias para a sua utilização. Para o estudo foram utilizadas imagens em tons de cinza que apresenta um objeto maior em seu centro, formando uma ilha de pixels de certo brilho dos quais não estão ligados as bordas da imagens (*e.g* Figura 1). O problema consiste em a partir da segmentação por área evidenciar estes elementos centrais maiores nas imagens e definir as coordenadas do centro de massa de massa destes.

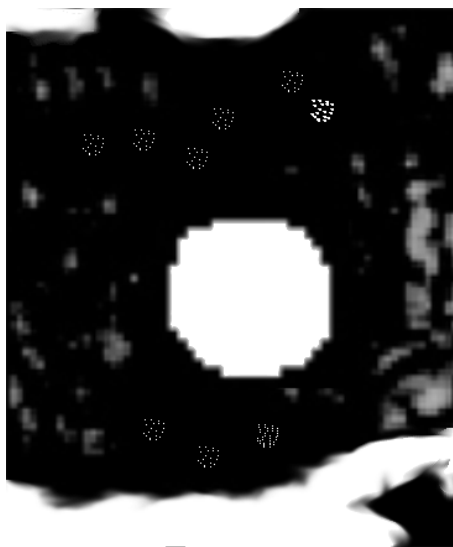


Figure 1. Exemplo de imagem em tons de cinza

No livro [Gonzalez and Woods 2008], é destacado que a segmentação por área geralmente envolve uma série de operações preparatórias, como limiarização, identificação de componentes conexos e rotulação. Essas operações são essenciais para preparar a imagem para a aplicação da técnica de segmentação por área propriamente dita. Além disso, os autores ressaltam a importância de técnicas avançadas, como a determinação automática de limiares, como o método de Otsu, para otimizar o processo de segmentação.

Com base nisso e nas orientações fornecidas pelo professor, para solucionar o problema foi desenvolvido um script em Python, o qual foi construído em 5 passos como evidencia o diagrama na Figura 2. Cada passo realizado gera uma imagem intermediária. A Figura 3 demonstra o mesmo processo com as imagens intermediárias geradas em cada passo, utilizando como entrada a Figura 1. As próximas subseções descrevem cada um dos passos para se obter esse resultado partindo do passo 2.



Figure 2. Diagrama com os passos para se obter o centro de massa



Figure 3. Imagens intermediárias gerada em cada passo

### 1.1. Threshold da imagem

Como primeiro passo do processamento da imagem tem-se o processo de limiarização, que consiste em determinar a intensidade de cinza que separa determinadas partes da imagem (conhecida como limiar), assim agrupando os pixels com intensidades próximas em grupos. Uma das dificuldades em se utilizar esse método é saber qual intensidade de cinza escolher como limiar. Para facilitar esse processo pode-se dispor de métodos que determinem o limiar de forma automática a partir da análise do histograma da imagem. A partir de um histograma, pode-se observar as diferenças de potencial nas distribuições das cores. Nesta atividade foi escolhida a variação do método Otsu, que é capaz de determinar múltiplos limiares. Foi utilizada a biblioteca em Python *scikit-image* com a função *threshold\_multiotsu* para determinar o limiar, o resultado do seu uso é demonstrado na Figura 4. Como é possível reparar na Figura o método Otsu definiu a partir da análise do histograma um limiar que está entre 100 e 150, junto a isso pode-se reparar a predominância de pixels nas duas extremidades, sendo os pixels de cor preta ou próximos os majoritariamente predominantes na imagem.

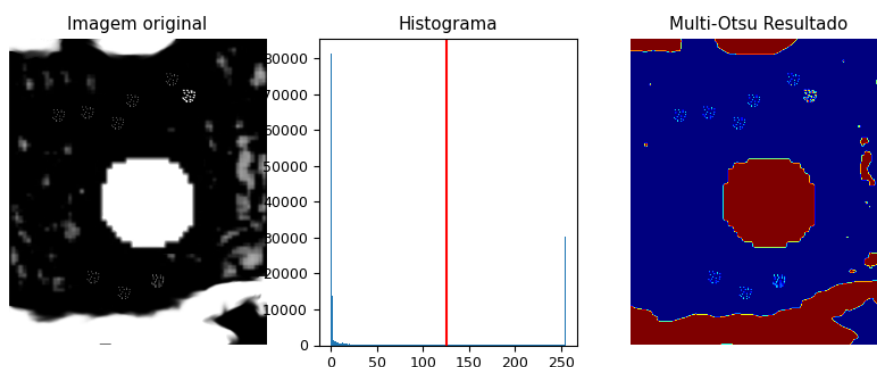


Figure 4. Definição de um limiar usando o método *threshold\_multiotsu*

A função *threshold\_multiotsu* permite que seja possível escolher o número de lim-

iares que se deseja obter no histograma a partir do parâmetro *classes*, permitindo separar a imagem em  $n$  grupos distintos. Como é possível ver na Figura 4 existe apenas um limiar, ou seja,  $classes = 2$ . É possível observar que muitos pixels cinzas nas laterais foram removidos nesse processo. Infelizmente, o resultado do uso desta função devolveu os valores dos pixels em 0 ou 1, sendo que no sistema utilizado os pixels com valor 255 são os que representam a cor branca, não o valor 1. Para facilitar os próximos passos foi aplicada uma função *binary\_threshold* que troca o valor de todos os pixels de valor 1 com o valor 255. O resultado do uso desta função pode ser visto na Figura 5. Pode-se notar agora de maneira mais evidente que muitas das manchas que haviam na lateral da Figura 1 foram removidas nesse processo.

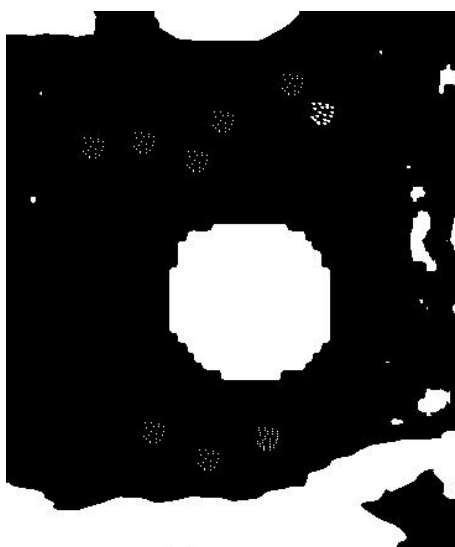


Figure 5. Imagem gerada a partir da função *binary\_threshold*

## 1.2. Rotulação

Este passo tem como objetivo rotular os elementos de acordo com o conjunto conexo do qual fazem parte. Um conjunto conexo de pixels é composto por pixels que são conectados entre si por possuírem a mesma intensidade de tom de cinza – nesse caso por possuírem o pixel na cor branca – por meio de alguma regra de vizinhança dos quais podem ser: conectividade 4 ou conectividade 8. Nesta tarefa foi utilizada a conectividade 8, ou seja, cada pixel procura em seus 8 vizinhos um pixel da mesma intensidade de cinza, como mostra a Tabela 1. Nessa tabela podemos ver como alcançar todos os vizinhos do pixel com as coordenadas  $(i, j)$  a partir do uso de operações aritméticas.

NO $(i-1, j-1)$	N $(i-1, j)$	NE $(i-1, j+1)$
O $(i, j-1)$	$(i, j)$	L $(i, j+1)$
SO $(i+1, j-1)$	S $(i+1, j)$	SE $(i+1, j+1)$

Table 1. Pixel com conectividade 8

A rotulação atribui a cada um desses componentes conexos uma cor distinta, como mostra a Figura 6, do qual usa do método de pseudo cores para facilitar a visualização dos conjuntos. Para realizar a rotulação foi aplicada uma Busca em Profundidade (*Depth-first Search*) sobre cada componente conexo, e cada um deles foi marcado com um rótulo

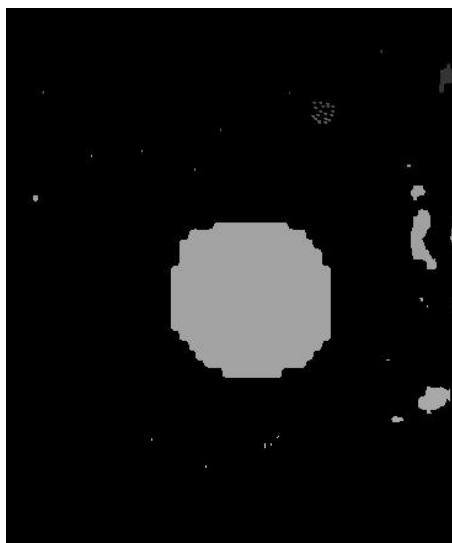
distinto. Com o intuito de facilitar o processo, foi acrescentada temporariamente uma moldura de zeros entorno da imagem para que não seja necessário se preocupar com os pixels que estão nas bordas.



**Figure 6. Imagem gerada a partir do processo de rotulação**

### **1.3. Remoção de elementos nas bordas**

Esta fase tem como objetivo tirar os elementos que estão nas bordas com o intuito de que estes não atrapalhem a busca do maior elemento central da imagem. Esta etapa consiste em percorrer a moldura de zeros ainda presente e definir o conjunto de todos os rótulos que fazem vizinhança com a moldura. A partir deste conjunto é possível remover estes elementos da imagem (substituí-los pelo valor zero), como mostra a Figura 7. É possível ver que os dois elementos que estavam conectados tanto a borda superior quanto a borda inferior foram removidos.



**Figure 7. Imagem sem os elementos conectados às bordas**

#### 1.4. Cálculo do centro de massa

Por fim, o último passo é calcular o centro de massa do maior elemento. Para isso, é necessário encontrar o elemento com a maior quantidade de pixels na imagem, isso é feito percorrendo a imagem e contando quantos pixel cada elemento possui e devolvendo aquele com a maior quantidade de pixels. Após isso, pode-se calcular o centro de massa do elemento, que é a coordenada média (x, y) de todos os pixels do elemento, como descrito na Formula 1.

$$centroDeMassa = (\lceil \sum_{x_i \in X}^n \frac{x_i}{n} \rceil, \lceil \sum_{y_i \in Y}^n \frac{y_i}{n} \rceil) \quad (1)$$

Com X e Y sendo o conjunto de todas as posições que estão no maior elemento e, onde  $n$  é igual a  $|X|$  e  $|Y|$  respectivamente para cada somatório. Pode-se ver o centro de massa do elemento central na Figura 8, para facilitar a visualização foi desenhado uma cruz partindo deste ponto. Como a geometria desse elemento se aproxima a de um círculo, seu centro de massa acabou ficando bem centralizado em relação ao elemento.

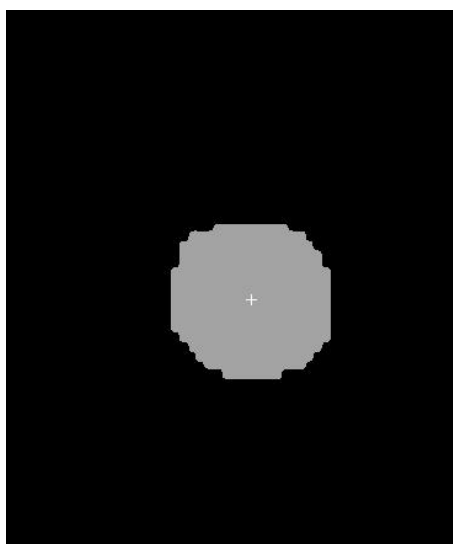


Figure 8. Centro de massa do elemento central

## 2. Instruções para rodar o programa

Para rodar o programa é preciso instalar algumas bibliotecas das quais o script possui dependências, estas são: numpy, matplotlib, skimage e pillow. Segue os comandos para instalá-los usando o gerenciador de pacotes pip.

Comando para instalar numpy:

```
pip install numpy
```

Comando para instalar matplotlib:

```
pip install matplotlib
```

Comando para instalar skimage:

```
pip install -U scikit-image
```

Comando para instalar pillow:

```
pip install pillow
```

Após instalar todas as dependências, pode-se rodar o programa a partir da função *main(path, result\_file\_name)* passando como parâmetros o caminho da imagem (*path*) que deseja calcular o centro de massa e o parâmetro *result\_file\_name* com o nome dos resultados das imagens que serão geradas na pasta *results*. Para rodar o programa pelo terminal basta estar na pasta da tarefa é dar o comando *python3 main.py*.

### 3. Testes

Para verificar se o algoritmo está rodando de maneira adequada, ou seja, conseguindo achar o centro de massa do maior elemento em uma imagem qualquer que possua as mesmas características da Figura 1, o mesmo algoritmo foi testado utilizando outras imagens. Para isso foi criada variações da Figura 1 onde o elemento central foi deslocado para algum outro ponto da imagem. A Tabela 2 mostra o resultado dos testes realizados a partir destas imagens.

Pode-se ver que os resultados obtidos na Tabela 2 são condizentes com o resultado obtido na Figura 8, e que a cruz que representa o centro de massa está localizada no mesmo lugar no elemento.

### 4. Análise dos resultados

Ao analisar os resultados obtidos na Seção 3, é possível observar que o método desenvolvido possui uma robustez no que se trata de encontrar o elemento central em imagens com as características descritas. Também é possível notar que cada passo gera as imagens intermediárias de maneira correta conforme descrita em cada subseção da Seção 1. Assim a partir desses resultados é possível concluir que o método de segmentação por área é uma técnica eficiente para destacar determinados elementos de uma imagem e extrair informações importantes desta, sendo uma ferramenta bastante útil para desenvolvimento de processamentos de imagem mais complexos.

### References

Gonzalez, R. C. and Woods, R. E. (2008). *Digital Image Processing*. Prentice Hall, 3rd edition.


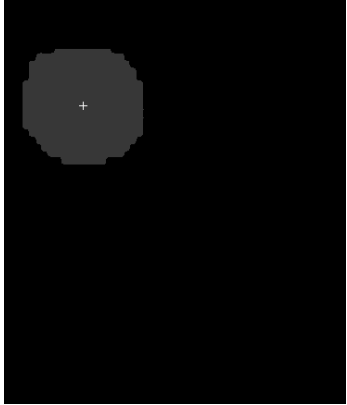

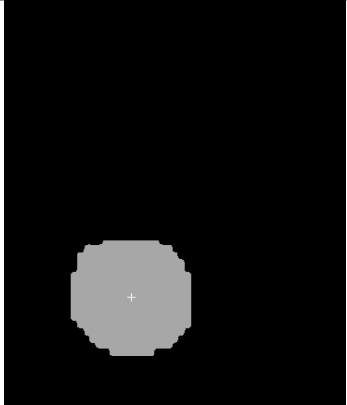
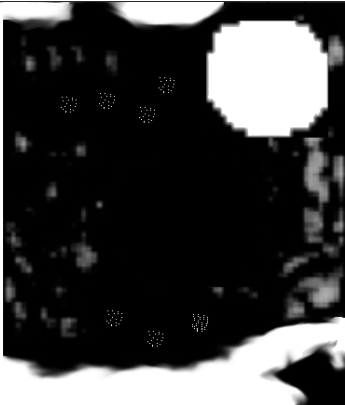
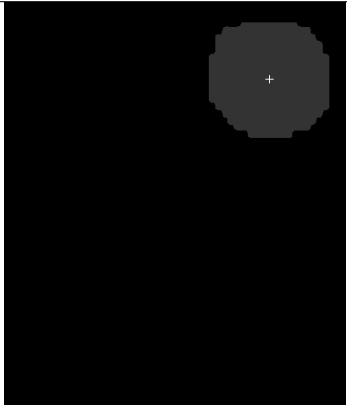
Imagem de entrada	Imagem de saída
	
	
	

Table 2. Testes realizados com diferentes imagens