Documentação Trabalho 3 - IPI

Bruna Azambuja
Departamento de Ciência da Computação
UnB
Brasília, Brasil
bubu.azambuja@hotmail.com

Keywords - processamento, imagens.

I. Resumo

Este trabalho teve como objetivo principal aplicar os conhecimentos obtidos em sala de aula sobre processamento e características gerais de uma imagem ou vídeo; tais como algoritmos de segmentação de objetos numa imagem, e manipulação de vídeos. Os códigos foram desenvolvidos em MatLab 2019, e testados com imagem/vídeo específico, como será explicado melhor na seção de Resultados.

O exercício consiste em duas partes: A primeira testa a capacidade de segmentar uma imagem, ou seja, separar pequenos objetos do fundo e entre si, de acordo com uma determinada característica de semelhança entre as regiões segmentadas.

Já a segunda parte do trabalho tem como objetivo criar um programa que leia um arquivo de formato diferente do usual – nesse caso, em formato YUV, e utilizá-lo para obter o primeiro quadro de um vídeo, no qual deve-se fazer a segmentação do que é pele humana neste quadro.

II. Introdução

A. Primeiro Exercício: Segmentação

Neste exercício foi dada uma fotografia microscópica de células em níveis de cinza. A saída gerada pelo programa deve ser uma imagem em que as células estejam segmentadas, ou seja, separadas entre si e do fundo. Para isso deve-se usar funções de preenchimento de buracos, binarização, e etc, para que no fim o WaterShed possa ser aplicado com eficiência.

B. Segundo Exercício: Leitor YUV e Segmentação de Pele

Já na segunda parte do trabalho, tem-se um arquivo YUV e deve-se criar um programa que receba esse arquivo e leia os N primeiros quadros de resolução AxB, retornando N matrizes de inteiros de 8 bits sem sinal.

Após implementada, devemos usar essa função para ler o primeiro quadro de um vídeo em formato YUV, e utilizar uma máscara de segmentação que nos indique a pele humana presente nesse quadro.

III. Metodologia

Dado o resumo do objetivo de cada exercício, será feita a explicação sucinta de como foi alcançado tal objetivo. Quais funções foram usadas e quais algoritmos foram implementados para que o programa funcionasse do jeito esperado.

Lembrando aqui que os nomes de funções utilizadas são referentes ao MatLab 2019 - *nota: funções utilizadas no exercício 2 só estão disponíveis a partir de versões do Matlab 2018, versões anteriores não apresentam tais funções!*

A. Primeiro Exercício:

Foi recebida uma imagem microscópica no formato .jpg de células. Primeiramente, foi feita a transformação da imagem para sua versão em escala de cinza, com a função *rgb2gray*, depois foi realizada a binarização da imagem, utilizando a função *imbinarize* e *graythresh* do MatLab, que calcula o limiar Threshold ideal.

Depois, utilizou-se a função *bwareaopen* com limiar 70, para preencher espaços descontínuos, e a função *imfill* com parâmetro '*holes*', já no negativo da imagem, para preenchimento dos buracos de dentro das células.

Após feito o pré-processamento da imagem, podemos finalmente calcular, a partir da função *bwdist*, a Transformada de distância Euclideana do complemento da imagem. Essa função, em sua essência, calcula a distância do pixel atual ao pixel não zero mais próximo, no que resulta, visualizando o nível de brilho como uma terceira dimensão, em buracos de pixels zero no centro das células. Ou melhor, se fizermos o negativo desse resultado, geraríamos picos de pixels brancos nos centros das células.

Fazemos então o complemento dessa imagem de distâncias gerada, e forçamos os pixels que não pertencem ao objeto a obterem valor Infinito. Nesta imagem de distâncias, foi aplicado uma operação morfológica de abertura com elemento estruturante disco de raio 3, para eliminar os pontos de encontro entre duas células.

Então, finalmente, basta aplicar o WaterShed, e voltar o valor dos pixels setados com Infinito para zero. Apenas para melhor visualização, vamos criar uma imagem no formato rgb, utilizando a função *label2rgb*.

B. Segundo Exercício

Nesta questão foi recebido um arquivo de vídeo no formato YUV; primeiramente, definido o tamanho exato de cada quadro, percorremos o arquivo inteiro, obtendo quadro por quadro e os armazenando no formato certo, redimensionado.

©2019 IEEE

Como sabemos que o arquivo é YUV 4:2:0, tem-se que para cada 4 pixels da componente Y, tem-se 2 de U e 2 de V. Ou seja, na essência, enquanto Y tem tamanho MxN, as componentes U e V têm tamanho M/2xN/2.

Portanto, devemos percorrer o vídeo em quadros, sendo que cada quadro tem uma componente Y – MxN, U – M/2xN/2 e V – M/2xN/2. Separando cada componente como sendo uma dimensão de um quadro, e redimensionando as componentes U e V para terem o mesmo tamanho da imagem original MxN, temos que um único quadro agora poderá ser mostrado como sendo uma imagem de três dimensões simples MxNx3.

Feito isso, basta chamar a função implementada *ler_yuv* com os parâmetros M, N – que são a largura e a altura de cada quadro, respectivamente, e o número de quadros que queremos mostrar.

Após feito tal processo, devemos obter o primeiro quadro do vídeo, e fazer uma máscara de segmentação que determine a região de pele humana na imagem.

Para isso, utilizamos a componente V, já que é uma das componentes que diz respeito à cor; e, analisando a imagem, um bom jeito de diferenciar regiões é pelo aspecto da cor da pele ser quase toda homogênea. Logo, se fizermos a segmentação a partir da componente V, a pele estará mais ou menos homogênea em relação ao resto da imagem, e será mais facilmente segmentada.

Utilizou-se então o algoritmo Kmeans, que realiza a segmentação da imagem em **k** regiões de aspectos semelhantes. A partir da função *imsegkmeans* com parâmetro 2, dividiremos então a imagem em duas regiões – no nosso caso, o que é pele e o que não é pele. Foi aplicado também uma operação morfológica de fechamento na imagem, com elemento estruturante quadrado de lado 15, para que retiremos as descontinuidades da máscara.

Então, apenas para uma melhor visualização, utilizou-se a função *labeloverlay* que aplica a máscara de segmentação encontrada na própria imagem original, para que possamos ter uma melhor noção do resultado obtido.

IV. Resultados

Após uma noção geral de cada função dos exercícios e como foi implementada cada uma, será dada nessa seção a saída gerada de cada um dos programas com imagens pré-definidas.

A. Primeiro Exercício:

A imagem de entrada em nível de teste foi:

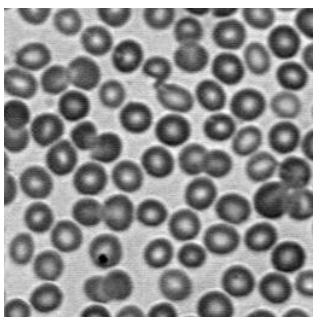

Fig.1. Imagem de entrada

Feita a binarização da imagem, obtendo:

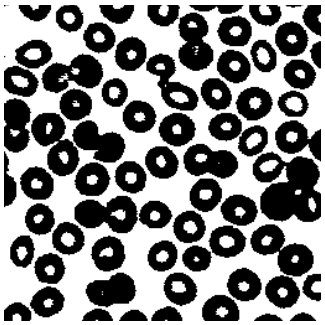

Fig.2. Imagem Binária

Para preenchimento de espaços desconectados, utilizou-se a função *bwareaopen* no complemento da imagem:

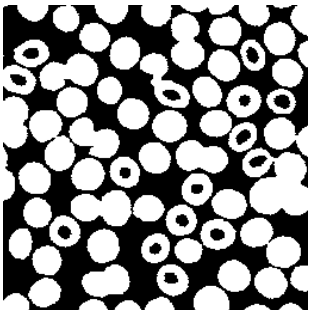

Fig.3. *bwareaopen*

Para o preenchimento dos buracos restantes, utilizou-se a função *imfill*:

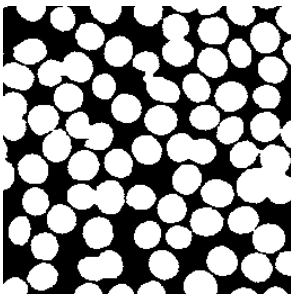

Fig.4. Buracos preenchidos

Calculada a Transformada de Distância Euclideana, obtemos:

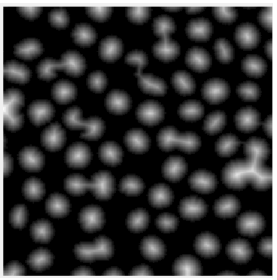

Fig.5. Imagem de Distâncias

Finalmente, aplicou-se primeiramente uma abertura para evitarmos o encontro de duas ou mais células. E aplicando-se o WaterShed e a transformação para RGB para melhor visualização, obtém-se a imagem segmentada:

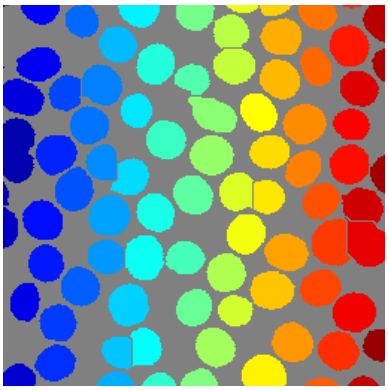

Fig.6. Imagem de Saída – Segmentada

B. *Segundo Exercício*

Como o arquivo de entrada é um vídeo, não será possível mostrar a entrada do programa, porém, utilizando a função *ler_yuv* consegue-se obter o primeiro quadro do vídeo:

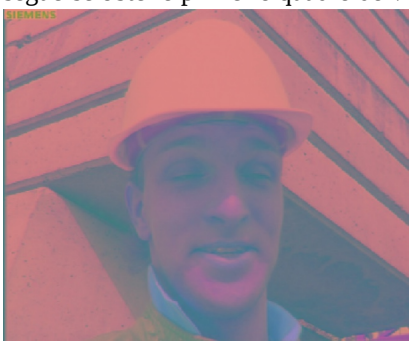

Fig.7. Primeiro Quadro – Foreman.yuv

Pegando apenas a componente V do primeiro quadro, para realizar uma melhor segmentação, tem-se:

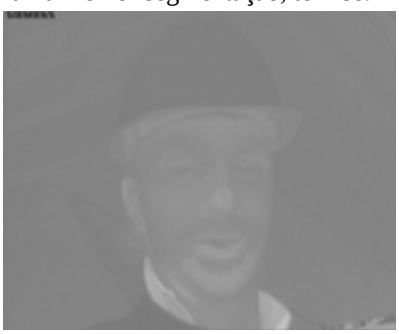

Fig.8. Componente V

Nota-se que apesar de não ser muito equalizado, fica nítida a diferença de tom entre a pele e o resto da imagem.

Após feita a aplicação do Kmeans, com duas regiões de semelhança, o fechamento, e o *labeloverlay* para visualização, obteve-se:

Fig.9. Imagem de Saída – Pele Segmentada

V. Conclusão

Neste trabalho pudemos analisar de perto como é feita a segmentação de imagens; desde tarefas simples como segmentar células entre si e do fundo, até tarefas mais complexas, como fazer uma máscara de segmentação da pele de uma pessoa.

Tivemos também que trabalhar com vídeos em extensão YUV, analisando o formato 4:2:0 e como ele é disposto. Bem como as componentes do YUV em si, e como elas podem ser utilizadas para realizar a segmentação de partes específicas da imagem.

Referências Bibliográficas

[1] Slides do professor Bruno Macchiavello.
[2] Site de documentação do MatLab – MathWorks.