Universidade Federal de Santa Catarina
Centro Tecnológico - Departamento de Engenharia Mecânica
Programa de Pós-graduação em Engenharia Mecânica
EMC410035 - Fundamentos da Visão Computacional
Prof. Dr. Armando Albertazzi
Alunos: Alex Dueñas Salazar
Rafael Bidese Puhl

Fundamentos da Visão Computacional

Imagens Digitais e Noções de Processamento de Imagens

Rafael Bidese, Alex Dueñas Salazar

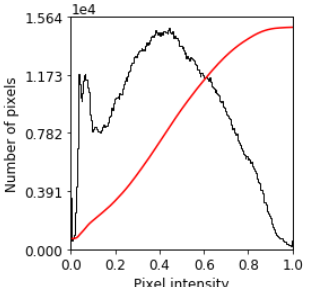
Resumo: Nesta aula prática, serão apresentados conceitos e exemplos de temas abordados na aula teórica. Os principais temas que serão aqui abordados são: histograma, filtros de convolução e morfologia, transformada de fourier em 2D e aplicações práticas utilizando essas e outras ferramentas de processamento de imagem.

Histograma

Em uma imagem, a quantidade de vezes que uma determinada cor se repete pode ser representada por um histograma. Como há várias possibilidades de cores, este tipo de histograma geralmente é gerado com base em uma foto em tons de cinza.

Por meio da equalização do histograma, ou seja, fazendo com que todas as intensidades de pixels tenham a mesma mesma frequência é possível melhorar o contraste de uma imagem. Uma imagem equalizada, tem uma função de distribuição acumulativa aproximadamente linear. Para ilustrar esse conceito, a Figura 1 apresenta uma imagem de baixo contraste. A Figura 1 passa pelo processo de equalização de histograma e o resultado pode ser visto na Figura 2.

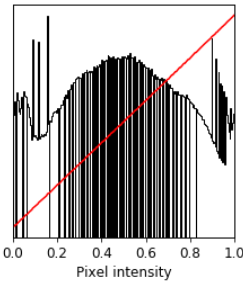
(a)



(b)

Figura 1: Exemplo de imagem com baixo contraste. (a) imagem de baixo contraste. (b) em preto, o histograma e em vermelho, a função de distribuição acumulativa.

(a)



(b)

Figura 2: Exemplo de imagem com histograma equalizado. (a) imagem equalizada. (b) histograma equalizado, em preto o histograma e em vermelho a função de distribuição acumulativa.

Histograma na prática

Um exemplo prático, programado em linguagem Python pode ser acessado no endereço eletrônico: https://github.com/rafaelbidese/fvc-practice, no arquivo *histogram.ipynb*.

Três abordagens são utilizadas para fazer a equalização do histograma utilizando o pacote *scikit-image*:

- Re-escalonamento de intensidade

```
img_rescale = exposure.rescale_intensity(img, in_range=(p2, p98))
```

- Equalização de histograma

```
img_eq = exposure.equalize_hist(img)
```

- Equalização adaptativa

```
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
```

Outro exemplo iterativo pode ser visto na aplicação web para edição de imagens Pixlr disponível em https://pixlr.com/editor/. Na opção Adjustment>Curves. Interaja com o *software* e tente reproduzir a função de distribuição acumulativa vista na equalização adaptativa do exemplo anterior.

Filtros de convolução e morfologia matemática (kernels)

Convolução é uma operação fundamental no processamento de imagens que consiste basicamente em aplicar um operador matemático a cada pixel numa imagem e mudar esse valor de pixel de alguma maneira. Para aplicar os diferentes operadores matemáticos, é usada uma outra matriz que é chamada de kernel. Geralmente os kernels são definidos como matrizes de tamanho 3x3, mas em muitas ocasiões varia. Assim, o kernel é ubicado de tal forma que no centro dele fique o pixel a ser considerado. Depois, é feita uma multiplicação de cada valor na matriz do kernel com os valores correspondentes na imagem e, em seguida, todos os valores são somados. Este novo valor corresponderá ao novo valor do pixel que está sendo analisado.

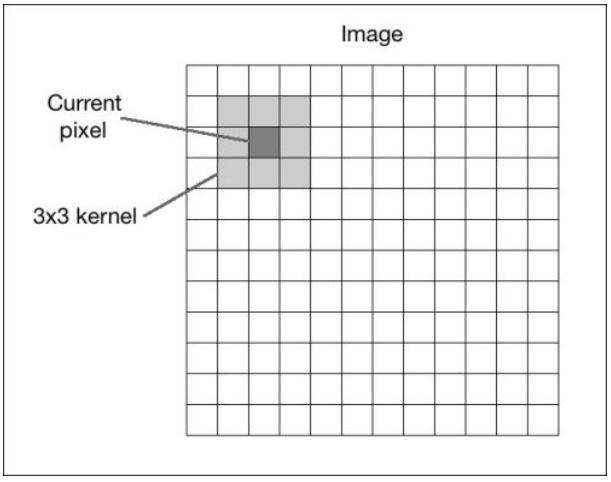


Figura 3. Exemplo de kernel

Dessa forma, o kernel é chamado de “filtro da imagem”, o processo de aplicar o kernel é chamado de “filtrado de imagem”, por último, a saída obtida é chamada de imagem filtrada.

Dependendo dos diferentes valores de kernel podem ser feitas operações como filtragem de imagens para minimizar ruídos, detecção de bordas, morfologias matemáticas, etc Na figura 4 é ilustrado o kernel usado pelo operador Sobel para encontrar as bordas tanto horizontais, quanto verticais numa imagem.

$\Delta x =$

-1	-2	-1
0	0	0
1	2	1

$\Delta y =$

-1	0	1
-2	0	2
-1	0	1

. Figura 4. Detector de bordas Sobel

Por outra parte, algumas operações que usam kernels são as chamadas de operações morfológicas. Aqui o kernel é chamado de elemento estruturante. As duas operações

morfológicas fundamentais são a erosão e a dilatação. Dessas duas operações derivam-se outras tal como a abertura, a fechadura, gradiente, Top Hat e Black hat.
Os elementos estruturantes mais usados nestas operações são: Retangular, elíptico e cruz.

```
# Rectangular Kernel
>>> cv.getStructuringElement(cv.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Figura 5. Tipos de elementos estruturantes

Na erosão, no momento de ser feita a convolução, o pixel será 1 só se todos os pixels que ficam embaixo do kernel também são 1.

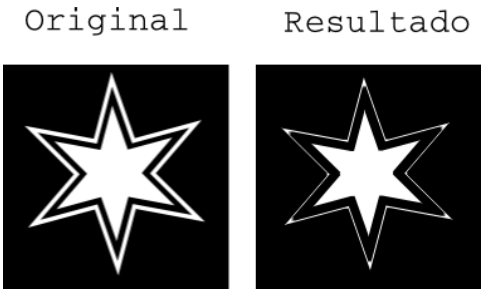


Figura 6. Erosão.

Na dilatação, pelo contrário, o pixel será 1 se pelo menos um pixel que fique embaixo do kernel também é 1.

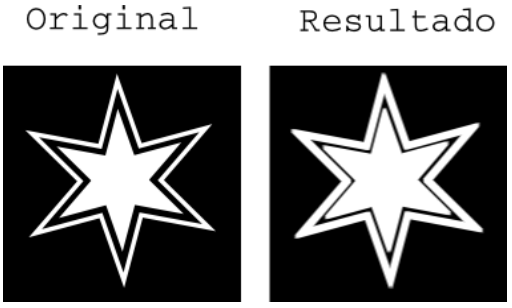


Figura 7. Dilatação.

Gradiente é a diferença entre dilatação e erosão, bom para encontrar bordas.

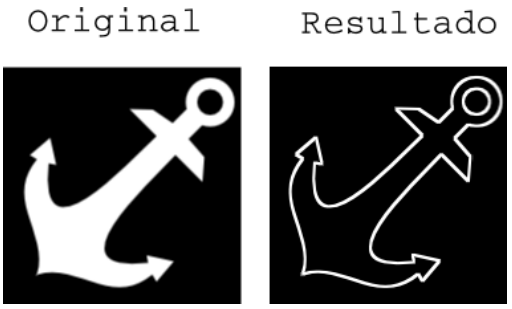


Figura 8 . Gradiente

Abertura é uma erosão seguida de uma dilatação.
Fechamento é uma dilatação seguida de uma erosão.
Top Hat é a diferença entre a imagem original e a sua abertura.
Black Hat é a diferença entre o fechamento da imagem e a imagem original.

Filtros na prática

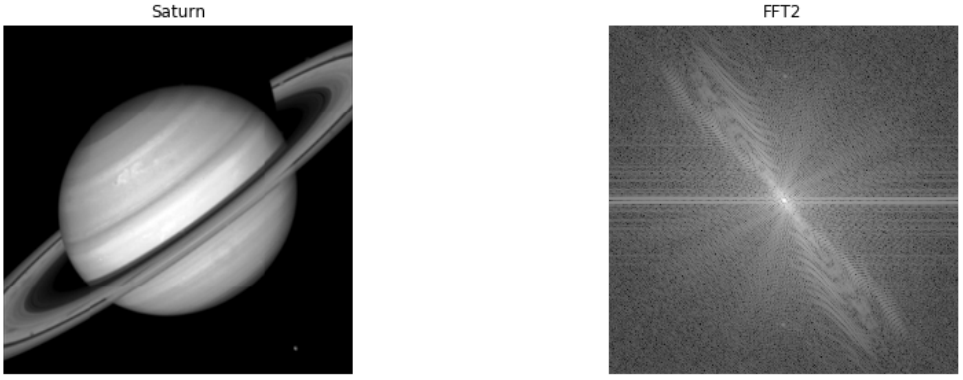
Um exemplo prático, programado em linguagem C++ utilizando a biblioteca OpenCV foi elaborado e será apresentado na aula.

Outro exemplo iterativo pode ser visto no endereço: http://setosa.io/ev/image-kernels/

Transformada de Fourier 2D

A transformada de Fourier expressa uma função em termos de funções de base sinusoidal, i.e., como soma ou integral de funções sinusoidais multiplicadas por coeficientes. Em processamento de imagens digitais, grande parte das aplicações da transformada de Fourier para remoção de ruídos semelhantes a texturas, com um padrão que se repete ao longo da imagem.

Para realizar a transformada de Fourier em uma imagem existe algumas abordagens, a mais utilizada é conhecida como linha-coluna. Primeiro, aplica-se a transformação em cada linha da imagem, gerando assim uma imagem intermediária. Posteriormente, a transformada é aplicada em todas as colunas dessa imagem intermediária. Um exemplo da transformada de Fourier 2D de uma imagem pode ser visto na Figura 9.



(a) (b)

Figura 9: Exemplo de transformada de Fourier. (a) Imagem do planeta saturno (b) transformada de Fourier 2D de (a).

Existe alguma diferença entre aplicar as transformadas em uma sequência diferente i.e., linha-coluna e coluna-linha?

Transformada de Fourier 2D na prática

Um exemplo prático, programado em linguagem Python pode ser acessado no endereço eletrônico: https://github.com/rafaelbidese/fvc-practice, no arquivo *2d-fft.ipynb*. Nesse exemplo são mostrados principalmente as transformadas de Fourier nos eixos x,y, x-y e y-x.

Para perceber melhor as aplicações dessa transformada traz-se aqui dois vídeos do YouTube para demonstrar o resultado da transformada de Fourier para diversos padrões visualizados em uma imagem e uma aplicação prática de remoção de ruído utilizando o *software Affinity Photo*.

Transformada de Fourier para diversos padrões de imagem: https://youtu.be/uD2BerBmnUs
Remoção de ruído utilizando *Affinity Photo*: https://youtu.be/6wfeMGwcF0c

Aplicações

Para demonstrar sistemas reais que utilizam os conceitos abordados anteriormente, traz-se aqui dois exemplos práticos, um da área de engenharia e outro da área biológica. Primeiro, será apresentado um sistema de inspeção óptica e posteriormente um sistema de contagem de bactérias em uma imagem de microscópio.

Sistema automático de inspeção óptica (AOI)

O setor da indústria usa complexos sistemas de visão computacional, com o objetivo de minimizar erros produzidos nos processos de produção e aumentar a qualidade dos produtos finais. Dentro desses sistemas, existem os focados na inspeção óptica automática,

que são muito usados na produção de placas de circuitos. Dependendo dos problemas a solucionar e requerimentos propostos, os algoritmos são usados para diferentes propósitos, tais como: medição de peças, de detecção de falhas, reconhecimento e classificação de objetos.

Neste exemplo apresenta-se um sistema que tem como finalidade detectar uma série de peças numa fita transportadora, cada um desses objetos terá informação importante como sua área, sua posição dentro da imagem, altura, largura, etc. Este tipo de informação é relevante para futuras etapas de classificação e manipulação por parte de robôs (Ver Figura 10).

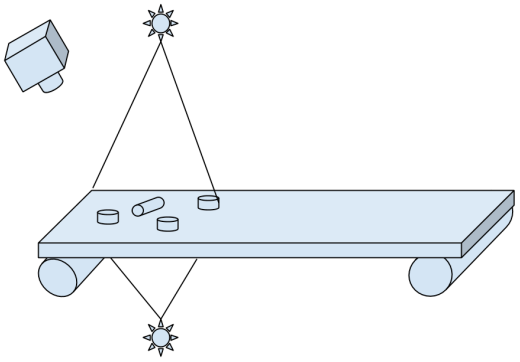


Figura 10 : Diagrama da esteira transportadora utilizada nesse exemplo.

Para levar a cabo o processo de segmentação realizam-se uma série de operações que são descritas na Figura 11.

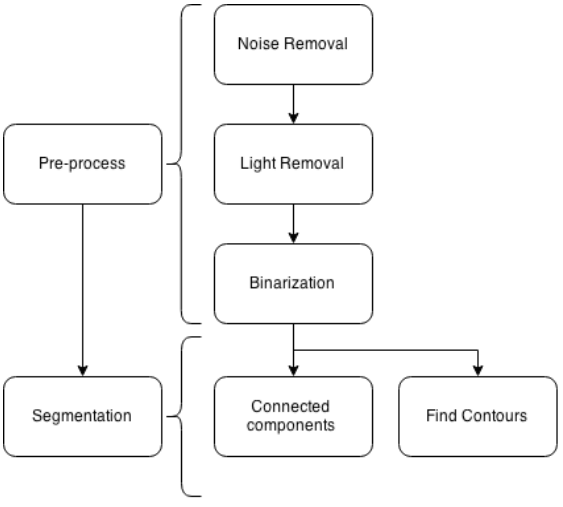


Figura 11 : Fluxograma do processo de classificação de peças.

A primeira etapa do processo, chamada de pré processamento. Tem como finalidade fazer um tratamento da imagem através da remoção de ruídos presentes e efeitos de luz

indesejados, isto, com o fim de minimizar possíveis erros na etapa de binarização da imagem.

Para remover o ruído presente na imagem é utilizado um filtro da mediana. Este é muito usado para remover o efeito conhecido como “sal e pimenta”.

Para minimizar os efeitos devido às mudanças de iluminação, é usado um “padrão de iluminação”, este consiste em uma imagem do cenário onde são capturadas as imagens dos objetos (background). Depois, podem ser usadas simples operações matemáticas como a diferença ou a divisão a fim de remover esse padrão da imagem com os objetos. Como resultado obtém-se uma imagem com um mínimo de objetos indesejados (Figura 12).

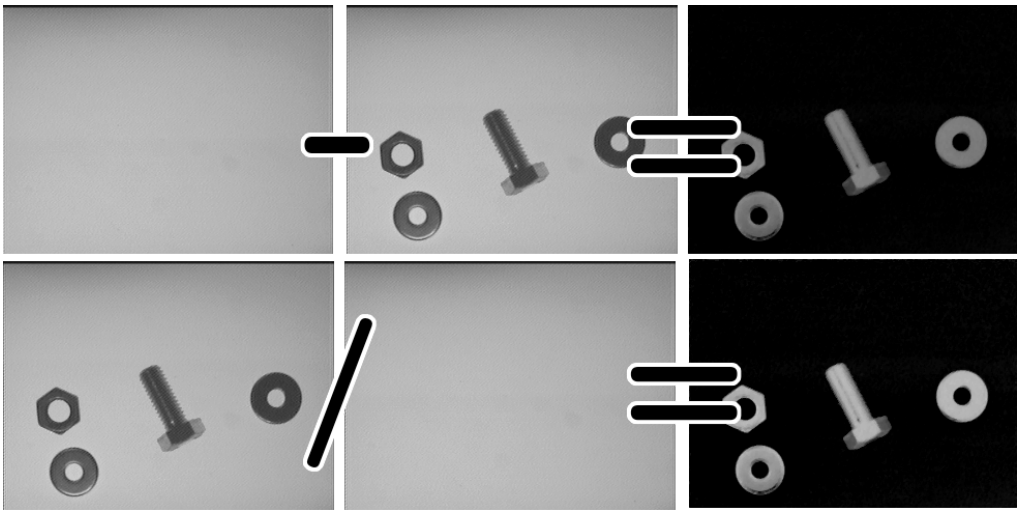


Figura 12 : Cancelamento de plano de fundo.

Após remover o background da imagem é feita uma binarização na imagem resultante, para isto é utilizado um valor de limiar igual a 30.

Neste ponto do processo, pode ser feita a segmentação das peças resultantes na imagem. Uma técnica muito usada para identificar e segmentar objetos numa imagem binarizada é a de análise de componentes conectados. Esta, tem como objetivo fundamental etiquetar os elementos presentes numa imagem usando conectividade de 4 ou 8 pixels. Na Figura 13 pode-se observar um exemplo sobre o funcionamento do algoritmo.

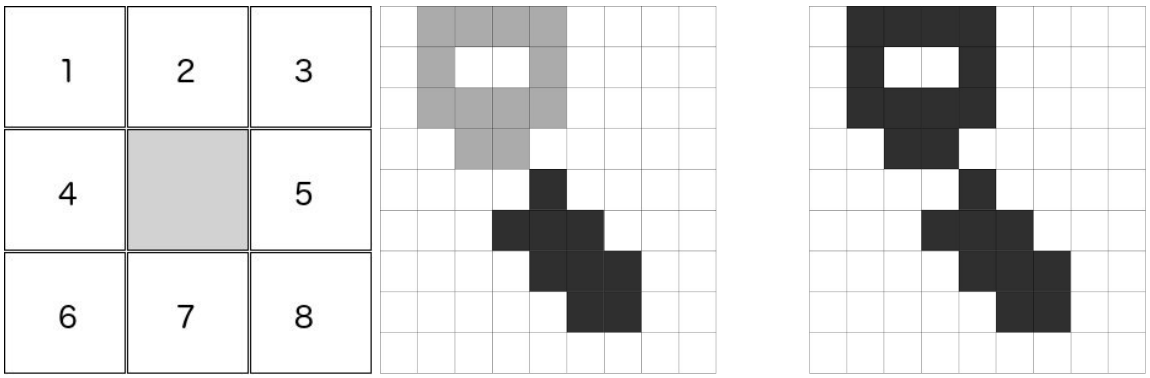
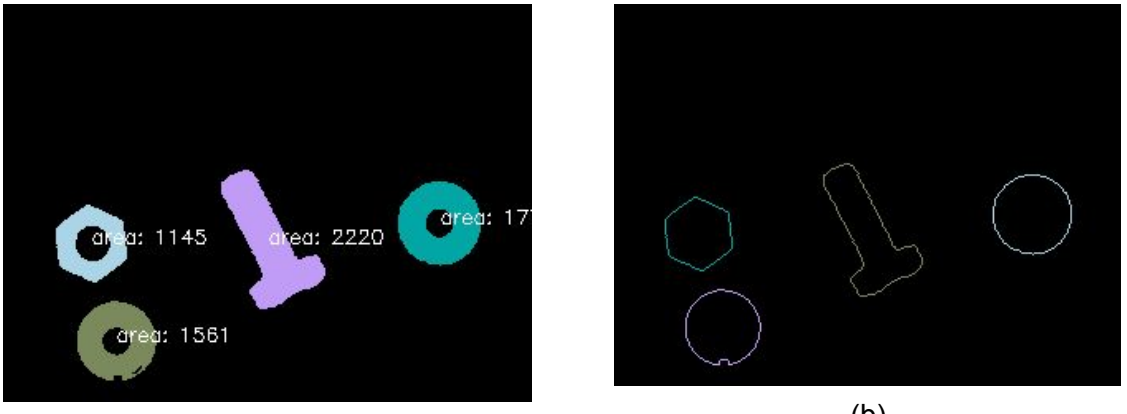


Figura 13 : Análise de componentes conectados.

Graças a função ***connectedComponentsWithStats*** presente na biblioteca de OpenCV é possível observar informação importante de cada um dos objetos etiquetados: Altura, largura, posição na imagem, área.

Outro algoritmo que é muito usado para segmentar objetos dentro de uma imagem é aquele baseado na detecção dos contornos de cada um dos elementos a segmentar. Para este fim OpenCV possui uma função chamada ***findContours.***

O resultado final ao aplicar os dois algoritmos mencionados anteriormente pode ser observado na Figura 14 .



(a) (b)

Figura 14: Resultado final. (a) medida das áreas das peças (b) contornos das peças

Contagem de bactérias em imagem de microscópio

Trabalhos manuais em laboratórios de pesquisa e análise são extremamente demorados. Muitas vezes, tem-se um grande número de amostras a analisar, ou até mesmo, a análise exige uma contagem de milhares de células em uma mesma amostra. Pela utilização de longos tempos fazendo essa contagem, dois problemas surgem: erros relacionados ao cansaço e custo de hora-trabalho envolvido.

Com isso, faz sentido que esse tipo de trabalho seja automatizado e o modo mais intuitivo de fazê-lo é por meio do processamento das imagens digitais que podem ser capturadas pelo microscópio.

Nessa aplicação, vão ser contadas são *caulobacter crescentus,* que são bactérias utilizadas como modelo de estudo da regulação do ciclo celular e o diferenciamento celular. Imagens de microscópio de exemplares dessa bactéria podem ser vistas na Figura 15 .

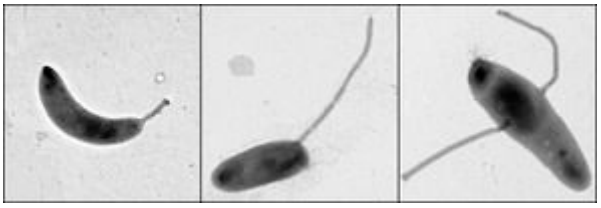


Figura 15: Posicionamento diverso das bactérias durante análise.

Essa aplicação foi programada em linguagem Python e pode ser acessado no endereço eletrônico: https://github.com/rafaelbidese/fvc-practice, no arquivo *bacteria-example.ipynb*. Um fluxograma que representa as etapas de processamento realizadas pelo código pode ser visto na Figura 16.

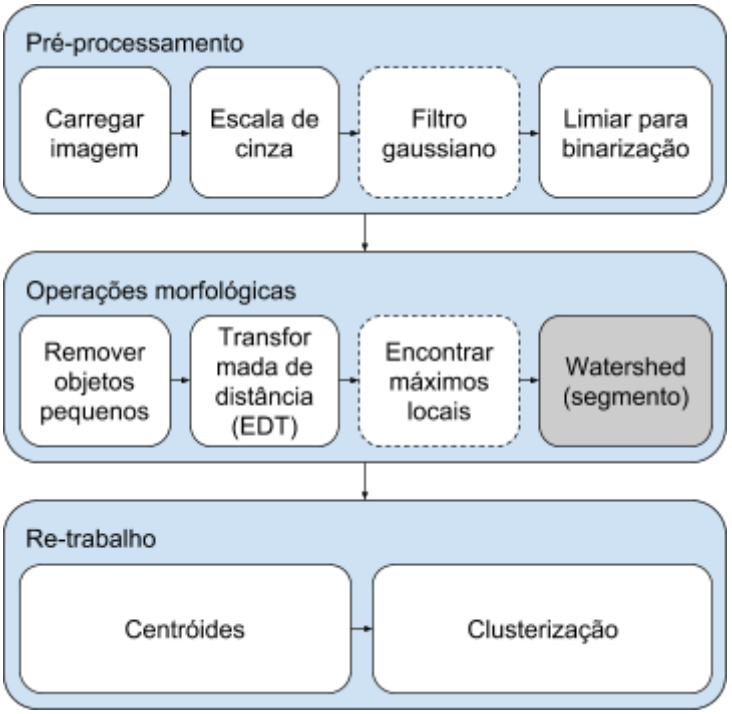


Figura 16 : Fluxograma do processo de segmentação de bactérias

Desafio: Leitura de códigos de barras

Diversas características das cenas e dos sistemas ópticos podem levar a mudança na morfologia de um código de barras. Transformações como distorção pela perspectiva e rotação pelo posicionamento do item são os efeitos mais recorrentes. Como, em um código de barras a informação está armazenada nas espessuras das barras e distâncias entre elas essas deformações podem levar a leituras equivocadas.

No seu sistema você utiliza uma câmera, para avaliar as formas e tamanhos de produtos que passam por uma esteira, como no exemplo anterior. Porém, agora também é necessário que o código de barras de cada item seja lido.

Se a sua câmera funciona de maneira fixa, que não é possível adaptá-lo para cada ângulo de inclinação que os códigos de barras se apresentam, e assim compensar as distorções, proponha em conjunto com a turma uma maneira de fazer com que seu sistema seja capaz de identificar os códigos de barras.