

## **Dados do relatório Lista 5: segmentação**

**Matéria:** Processamento Digital de Imagens (PDI) **código:** CMP1084-A01

**Aluno:** Vitor de Almeida Silva

**Matrícula:** 20161003305497

### **ENUNCIADO**

- 1. Para cada imagem transforme-a em escala de cinza e utilize a segmentação por superpixels. Utilize 5, 10, 100 superpixels e explique os resultados.**
- 2. Refaça a questão 1 com as imagens coloridas.**
- 3. Para cada imagem utilize a segmentação por K-means clustering.**
- 4. Para cada imagem utilize a segmentação utilizando o método de otsu.**
- 5. Para cada imagem utilize a segmentação utilizando o método de médias moveis.**
- 6. Para cada imagem utilize a segmentação por region grow.**
- 7. Crie sua própria rotina de segmentação para as imagens combinando alguns métodos de sua preferência, presentes ou não na lista. Justifique o uso dos métodos escolhidos.**

**1. Para cada imagem transforme-a em escala de cinza e utilize a segmentação por superpixels. Utilize 5, 10, 100 superpixels e explique os resultados.**

**Resposta:**

O método do superpixel foi introduzido por Ren e Malik em 2003. A técnica citada, consiste em agrupar pixels com características similares tais como cores, texturas entre regiões, brilho entre regiões e outras características de baixo nível. Essa técnica, parte do pressuposto que é vantajoso diminuir a quantidade das variedades de pixels em uma imagem, para aprimorar a retirada de informações da imagem, visto que, a torna mais “resumida” em termos de pixels (STUTZ; HERMANS; LEIBE, 2018).

Deste modo, segundo Stutz, Hermans e Leibe (2018), essa técnica pode ser aplicada para solução de problemas como os seguintes:

- Detecção de objetos;
- Segmentação de imagens;
- Notações que utilizam *datasets* (conjunto de dados);
- Análise de vídeos;
- Compreensão de cenas interior;

Alguns critérios de construção de superpixels são:

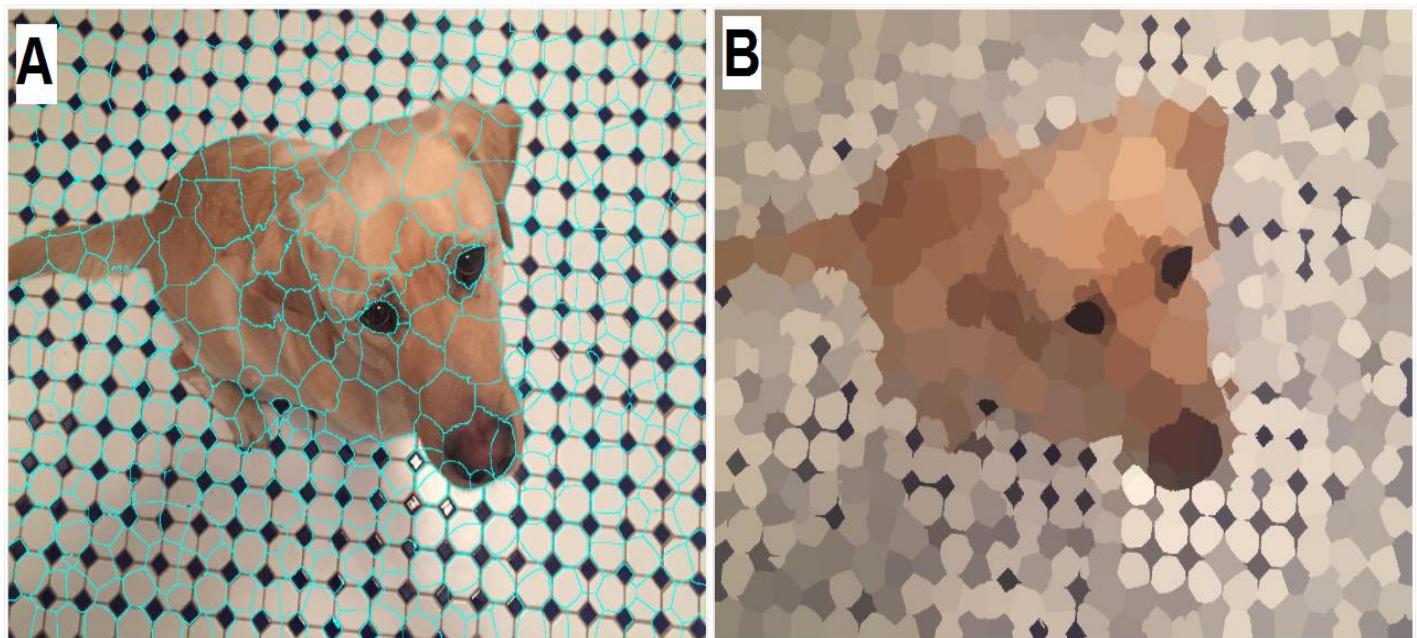
- Textura entre regiões;
- Textura intra-região;
- Brilho entre regiões;
- Brilho intra-região;
- Contorno inter-região;
- Luminosidade de contorno intra-região;
- Continuidade curvilínea.

De forma geral, a construção de um algoritmo de PDI que aplique a técnica de superpixel, traz os seguintes requisitos, segundo Stutz, Hermans e Leibe (2018):

- 1) **Particionamento:** superpixel deve definir uma partição da imagem. Superpixels devem ser separados e atribuídos a um conjunto para todos os pixels;
- 2) **Conectividade:** é esperado que os superpixels representem conjuntos conectados de pixels;
- 3) **Adesão a fronteiras/limites (Boundary Adherence):** Superpixels devem preservar os limites. Essa definição de limites da imagem pode depender da aplicação;
- 4) **Número de Superpixels:** O número de superpixels gerados deve ser controlável;
- 5) Depois de Aplicado o método e juntado os pixels em conjuntos de superpixels, normalizar o resultado para poder ser visualizado;

Tendo em vista a técnica e seus requisitos, MathWorks (2020) apresenta uma imagem, a qual foi segmentada utilizando a técnica do superpixels para  $N=500$ , sendo  $N$  o número de superpixels. A Figura 1, mostra a imagem mencionada.

Figura 1: exemplo de aplicação do superpixel,  $N=500$

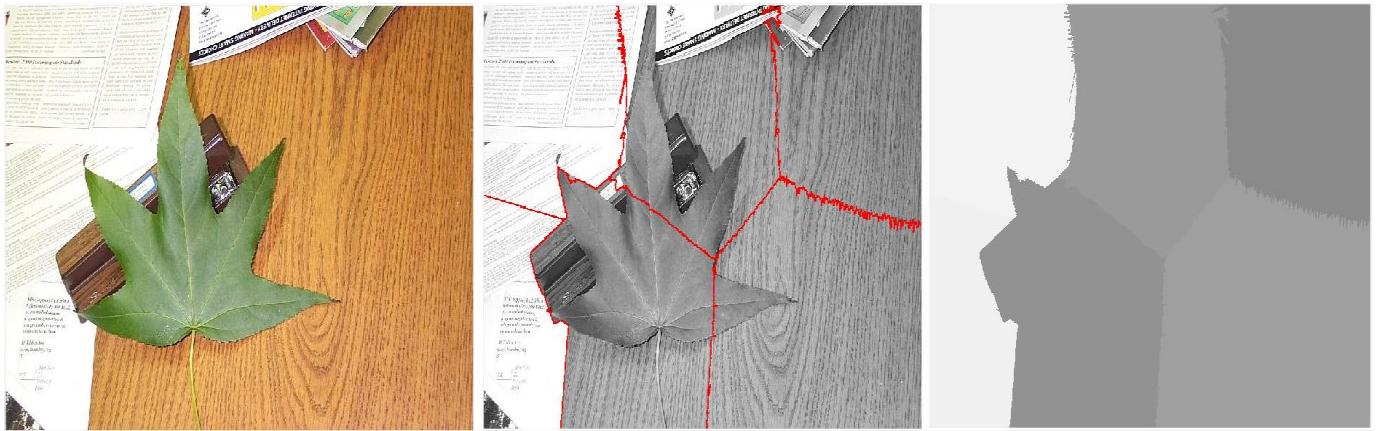


Fonte: (MATHWORKS, 2020)

Baseado nos critérios expostos sobre a segmentação por superpixel, foi desenvolvido um código em matlab baseado nas funções disponíveis em

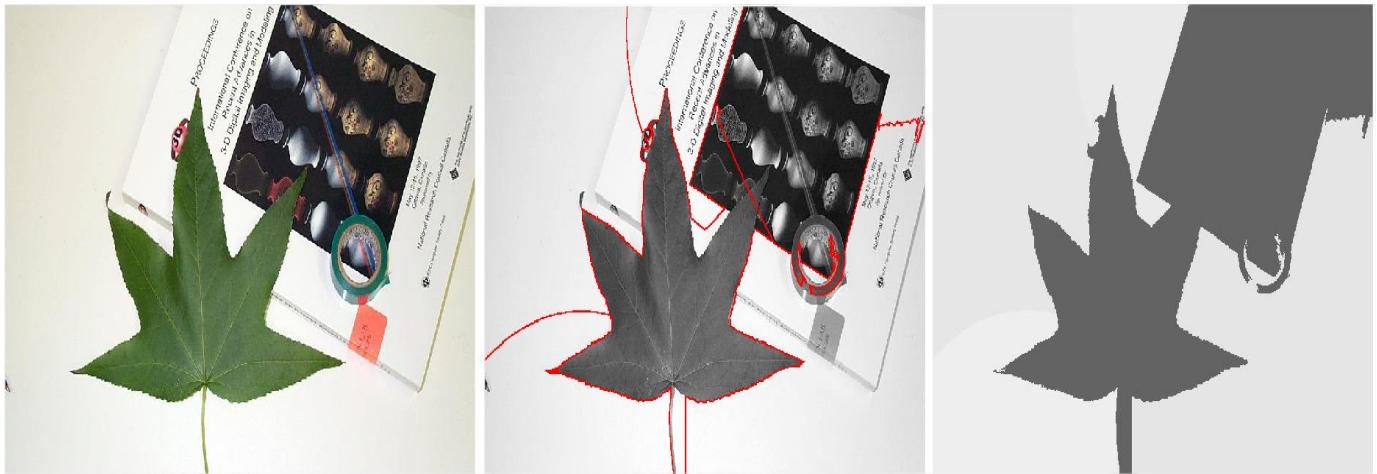
Mathworks (2020) e na teoria abordada. Serão usadas 2 imagens para mostrar o método citado nessa questão. As Figura 2 e 3 mostram o processo de segmentação de super pixel para N=5.

Figura 2: superpixel com N=5 img1



Fonte: Autoral

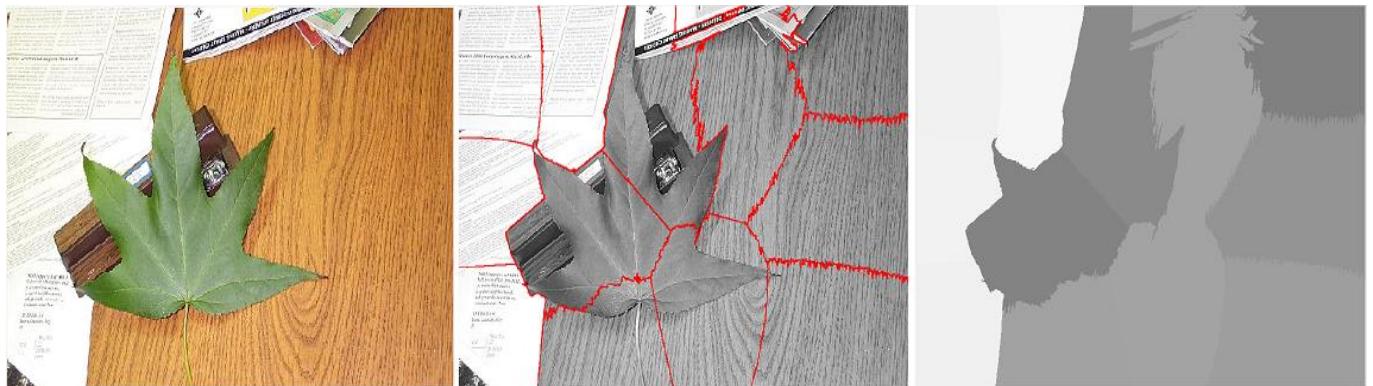
Figura 3: superpixel com N=5 img2



Fonte: Autoral

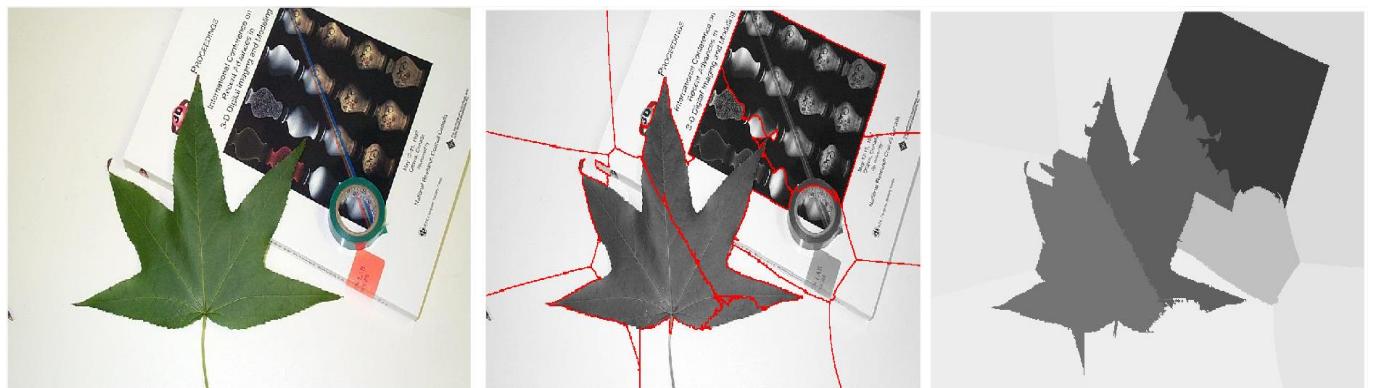
As Figura 4 e 5, mostram o mesmo procedimento para um N= 10.

Figura 4: superpixel com N=10 img1



Fonte: Autoral

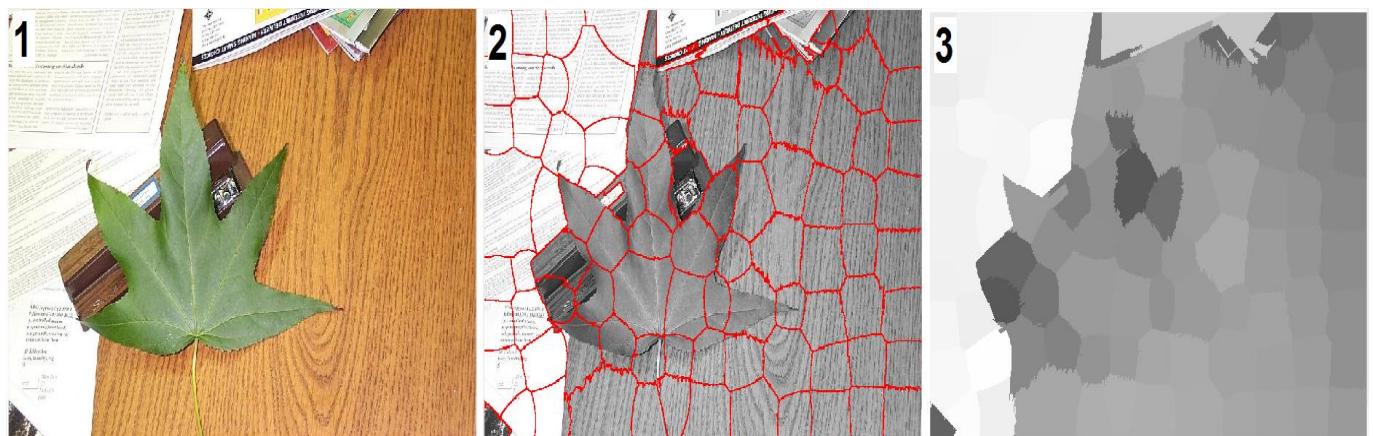
Figura 5: superpixel com N=10 img2



Fonte: Autoral

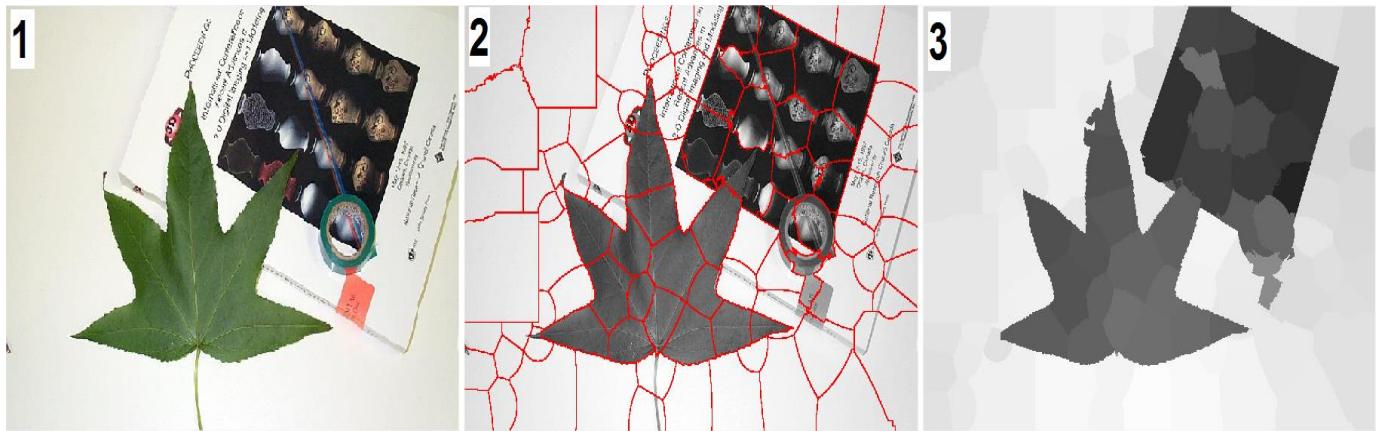
As Figura 6 e 7, mostram o mesmo procedimento para um N= 100.

Figura 6: superpixel com N=100 img1



Fonte: Autoral

Figura 7: superpixel com N=100 img2



Fonte: Autoral

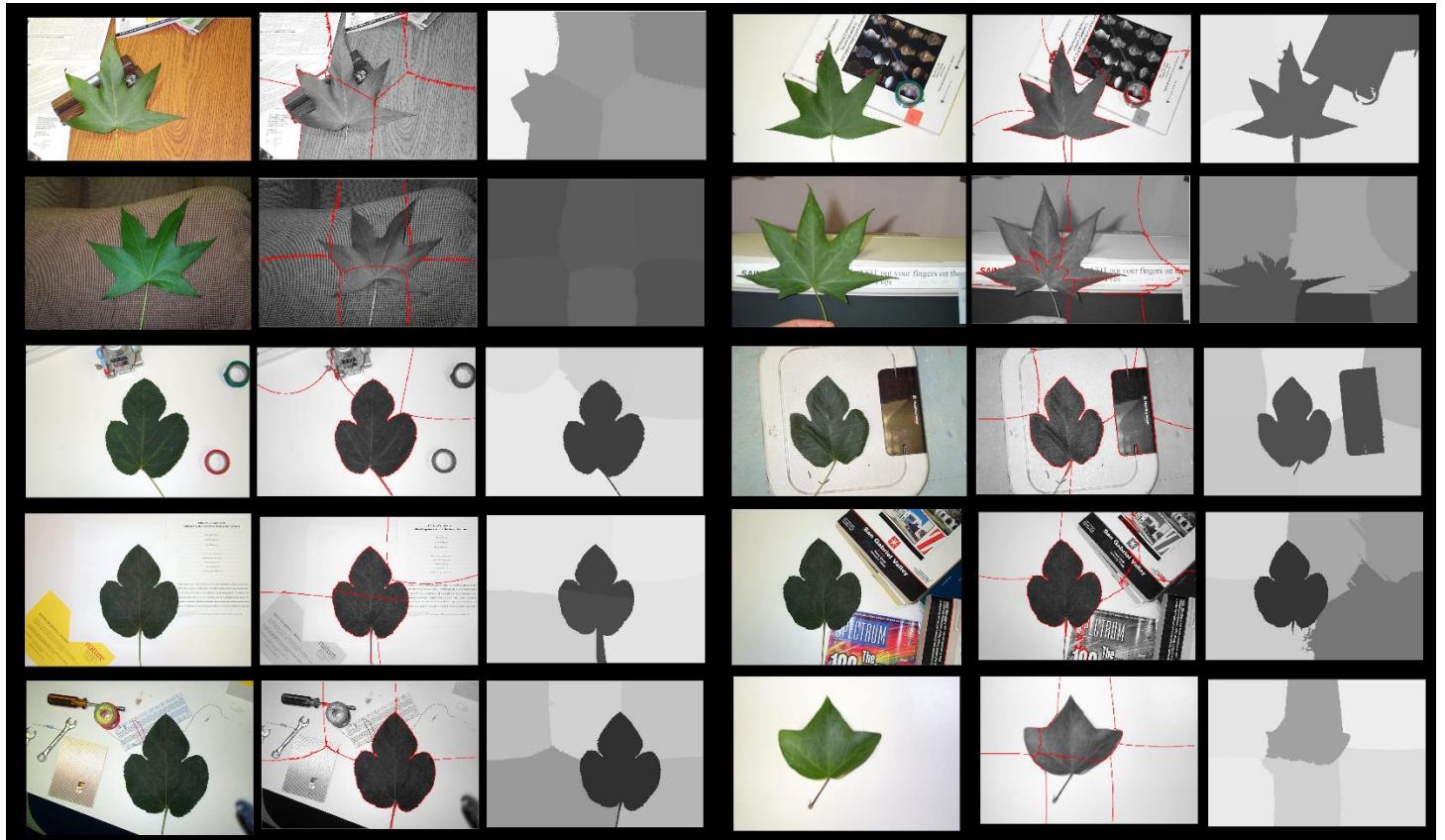
Analizando-se as Figuras geradas para N= 5, 10 e 100, é possível notar os seguintes pontos:

- A qualidade do resultado, em termos de se destacar a forma da folha, é totalmente influenciado pelo fundo da imagem. Isso pode ser visto na img2, onde o fundo é branco e contém poucas mudanças de tons, além de formar um contraste melhor com a folha. Se comparada a img2 com a img1, a img1 por ter um fundo marrom e poluído destacou bem menos a folha para os valores de N citados;
- Da mesma forma, o resultado também é influenciado pelo valor de N, que é o número de superpixels que serão gerados. Como pode ser visto nas Figura 1, 2, 3, 4, 5, 6 e 7, o formato da folha foi ficando mais bem definido conforme N aumentou, principalmente para a img2;
- Como a imagem é em escala de cinza, os superpixels são preenchidos com tons de cinza, isso pode gerar um resultado mais homogêneo de tons em algumas imagens.
- Também pode-se observar que ele retorna mais superpíxels do que definido em N, como no caso de N=5, onde foram retornados 6 superpíxels, deve ser por conta de alguma aproximação feita na função. Não entrarei neste mérito, já que, N foi definido como especificado em 5, 10 e 100.

O código utilizado para a questão está exposto a seguir, lembrando que ele está configurado para tratar todas as imagens, inclusive, mostra-las na tela. Isso

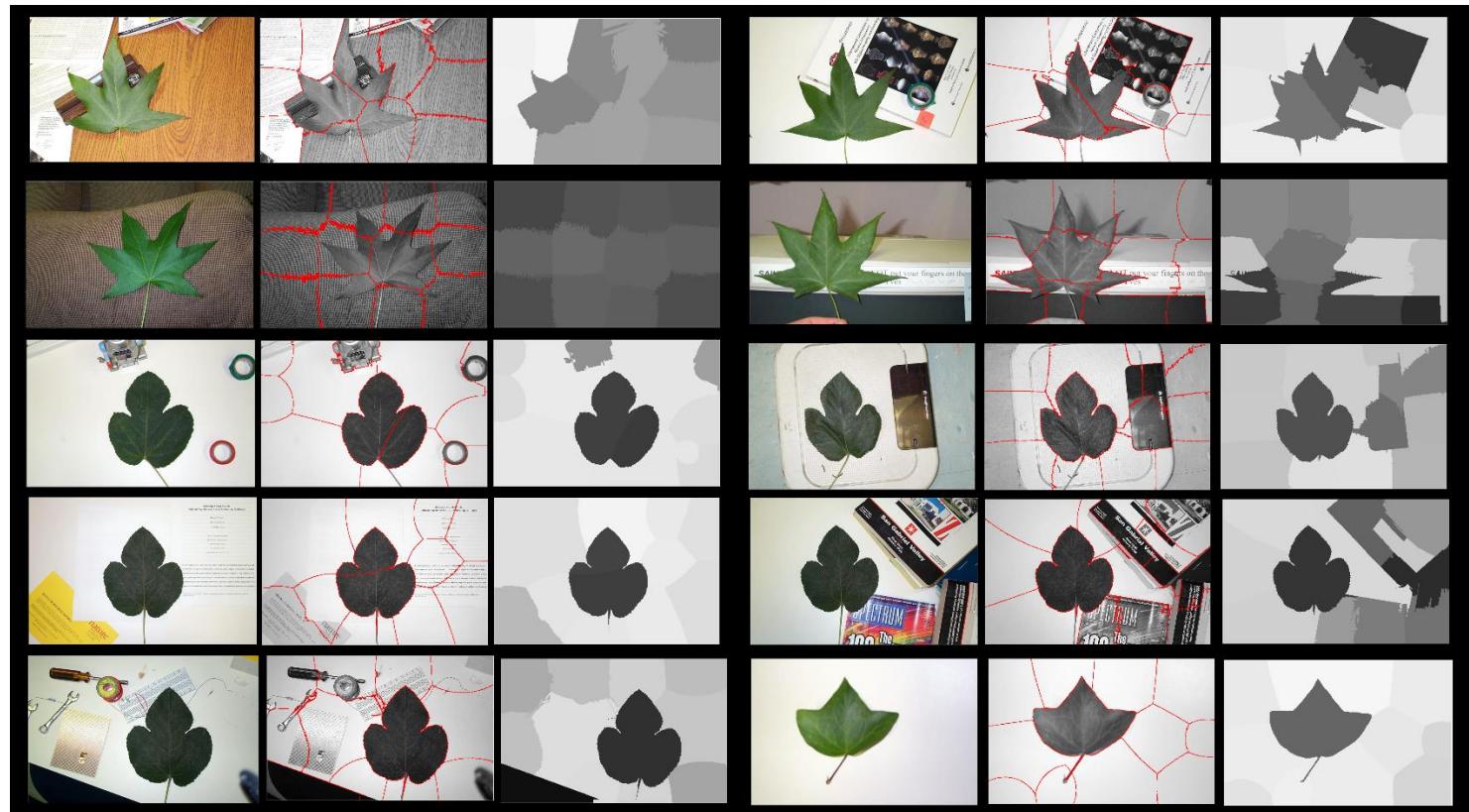
para atender a especificação do enunciado. As Figuras 8, 9 e 10, mostram a aplicação do método em todas as imagens para  $N=5$ ,  $N=10$  e  $N=100$  respectivamente.

Figura 8: superpixel para  $N=5$



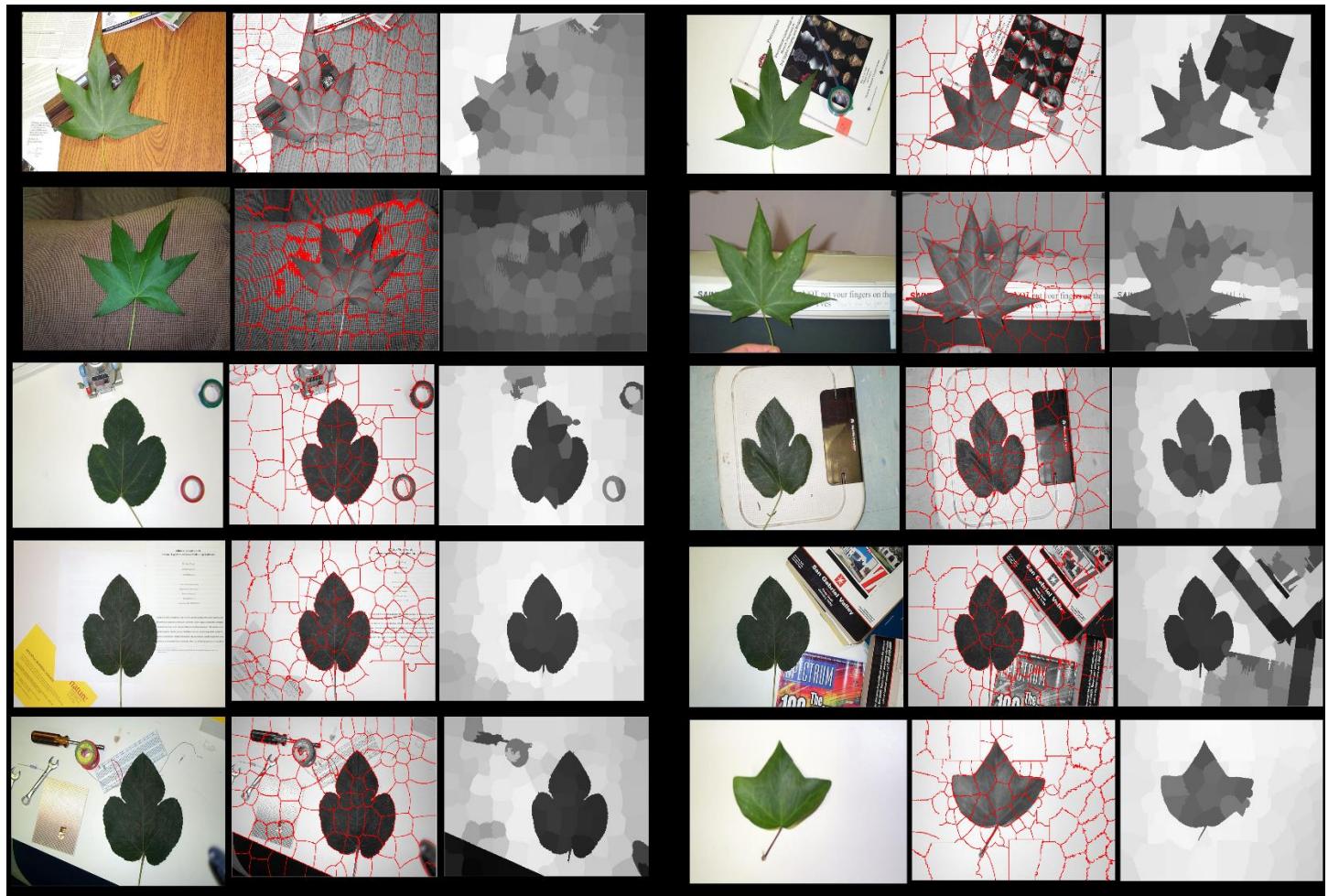
Fonte: Autoral

Figura 8: superpixel para N=10



Fonte: Autoral

Figura 8: superpixel para N=100



Fonte: Autoral

## Código questão 1:

```
%leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg*'));
n=numel(jpgfiles);
%% aplicando processamento de superpixel em todas as imagens da base
N=5; %numero de superpixels
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    imshow(img1);
    title('imagem original N=' + string(i));
    figure();

    imgGray = rgb2gray(img1);
    img2out=super_pixel_grayScale(imgGray,N,i);

    %% mostra a imagem resultante
    imshow(img2out,'InitialMagnification',67);
    title('imagem de superpixel N=' + string(i));
    figure();
end
%%
%% função de superpixel para imagem grayscale
%% essa função foi baseada nos conteudos disponíveis em (MATHWORK,2020)
function img2out = super_pixel_grayScale (img1,N, numeroImg)
%% dimensões da img1 e grayScale
[x,y,z] = size(img1);
imgGray = img1;

%% separa a imagem em superpixels controlados por N
[L,N] = superpixels(imgGray,N);

%% Essa função gera uma mascara baseada no array L, de superpixels
%consiste dos limites da imagem para o superpixel
BW = boundarymask(L);

%% imoverlay(img1,BW,'cyan') essa função "queima" uma imagem BW em cima de
%uma imagem A, é como se fosse uma colagem (MATHWORKS,2020);
imshow(imoverlay(imgGray,BW,'red'),'InitialMagnification',67);
title('conjunto de superpixels N=' + string(numeroImg));
figure();
%% retorna matriz de zeros == img1
img2out = zeros(size(imgGray),'like',imgGray);

%% converte matrizes rotulada paara celular de vetores de indice linear
idx = label2idx(L);

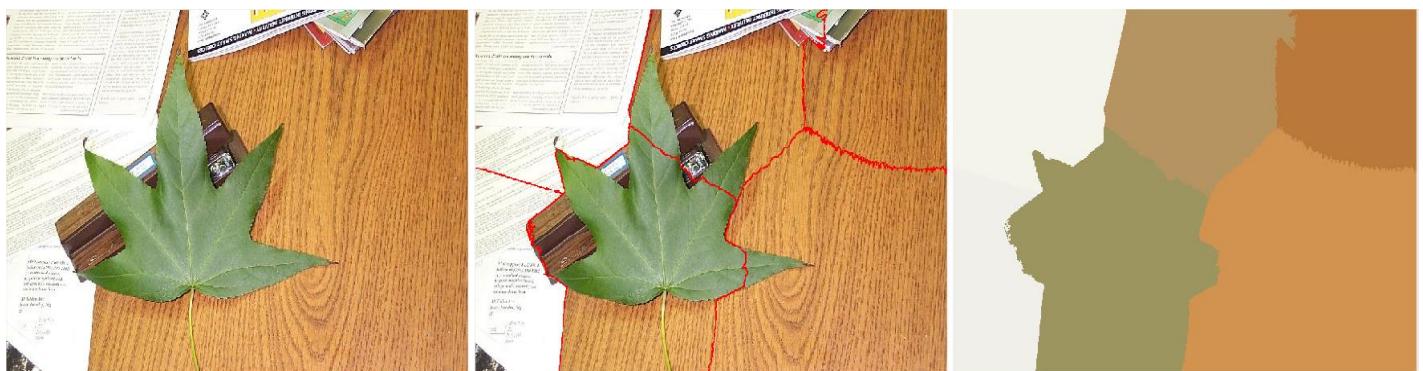
%% realiza media dos pixels para cada superpixel
for labelVal = 1:N
    IdxImg = idx{labelVal};
    img2out(IdxImg) = mean(imgGray(IdxImg));
end
end
```

## 2. Refaça a questão 1 com as imagens coloridas.

**Reposta:**

Para refazer a questão 1 utilizando imagens em RGB, é necessário adaptar o código para tratar os três tipos de conjuntos de cores. deste modo As Figuras 9, 10, 11, 12, 13 e 14 mostram os resultados da aplicação do superpixel par N=5, 10 e 100 em RGB respectivamente.

Figura 9: Superpixel RGB N=5 img1



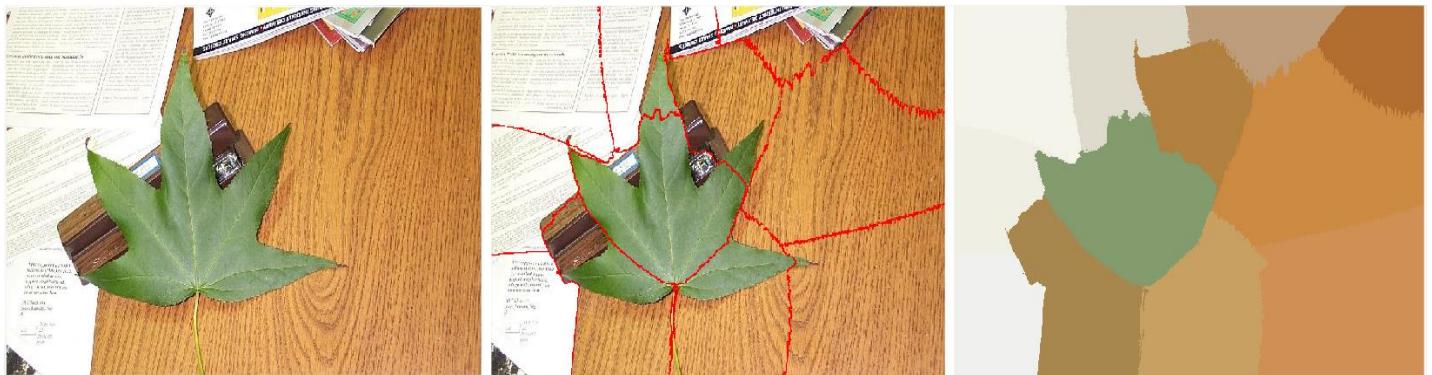
Fonte: Autoral

Figura 10: Superpixel RGB N=5 img2



Fonte: Autoral

Figura 11: Superpixel RGB N=10 img1



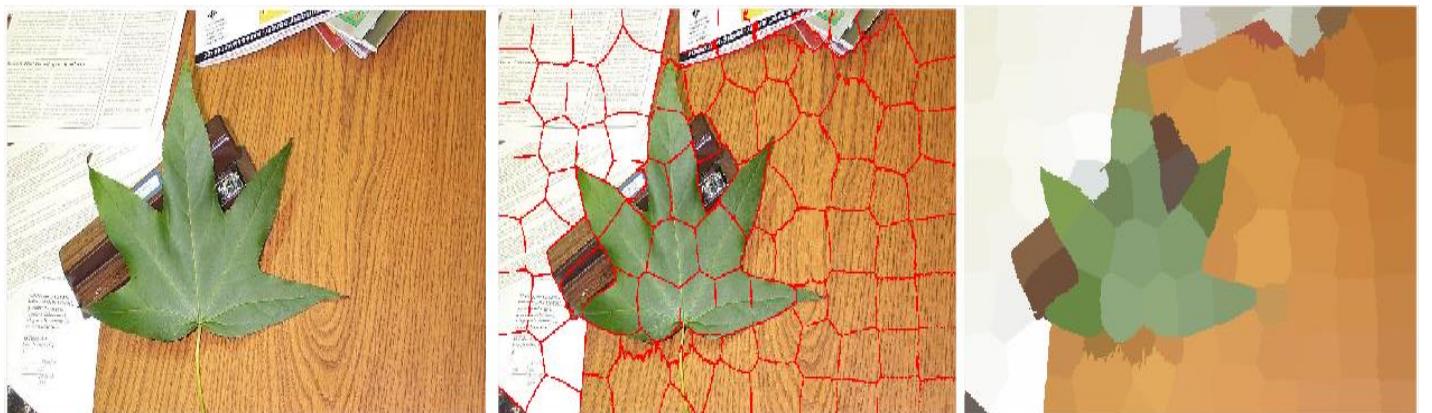
Fonte: Autoral

Figura 12: Superpixel RGB N=10 img2



Fonte: Autoral

Figura 13: Superpixel RGB N=100 img1



Fonte: Autoral

Figura 14: Superpixel RGB N=100 img2



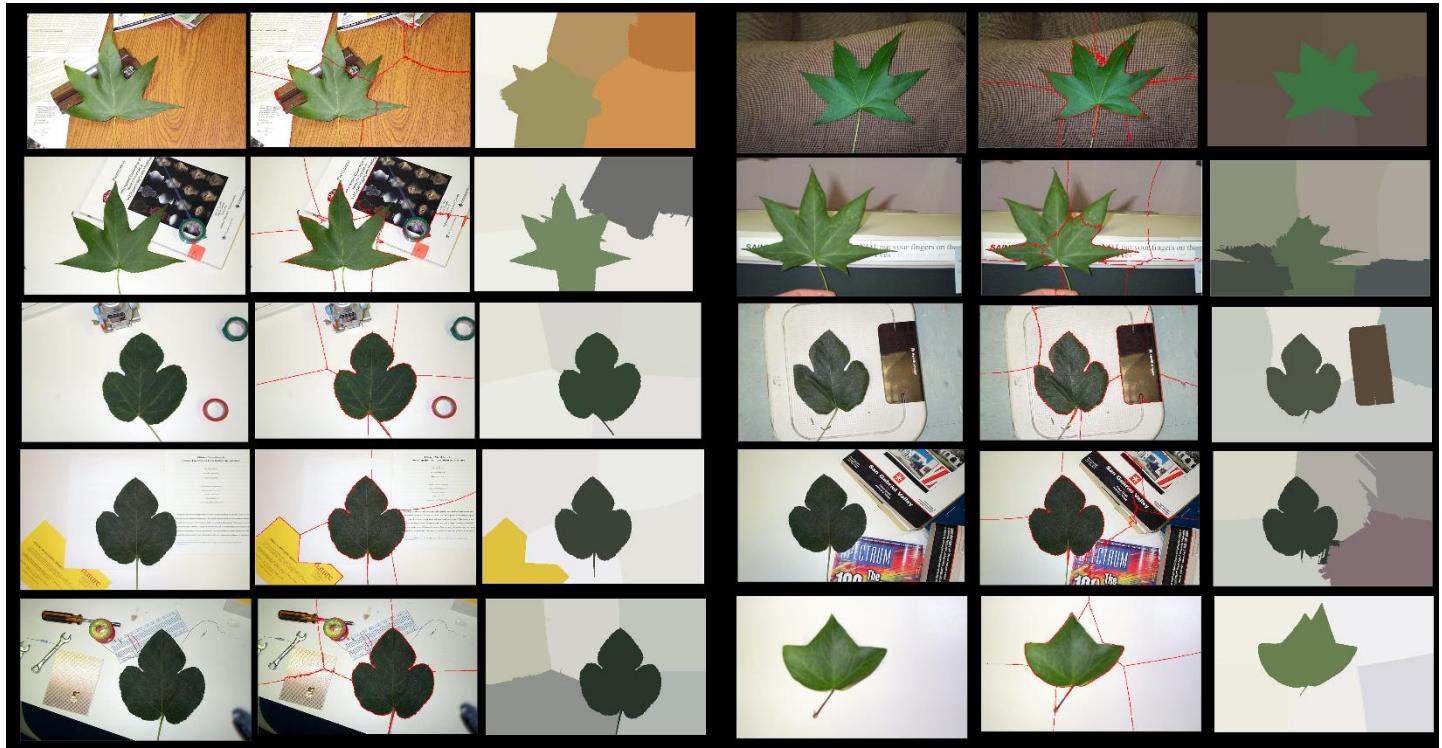
Fonte: Autoral

A partir da análise das imagens em RGB em comparação da Imagens em escala de cinza, pode se destacar os seguintes pontos:

- Similarmente ao exercício 1, conforme o valor de N aumenta, o detalhamento e a distinção dos objetos na imagem melhora, isso é visto de forma mais clara em imagens com fundos homogêneos.
- Em RGB em termos visuais é possível distinguir a posição da folha em todas as imagens. Se for comparada a Figura 11 com a Figura 4, é possível notar que na Figura 11, por estar em RGB, é possível, notar um borrão verde onde se localiza a folha;
- Pode-se pensar em uma estratégia onde se aplica a técnica do superpixels em imagens RGB para distinguir a folha do fundo, pensando até mesmo em analisar a região de cor verde;
- Por fim, as Figura 13 e 14 apresentarão os melhores resultados visuais, nestas, é possível distinguir de forma mais precisa o formato da folha na imagem;

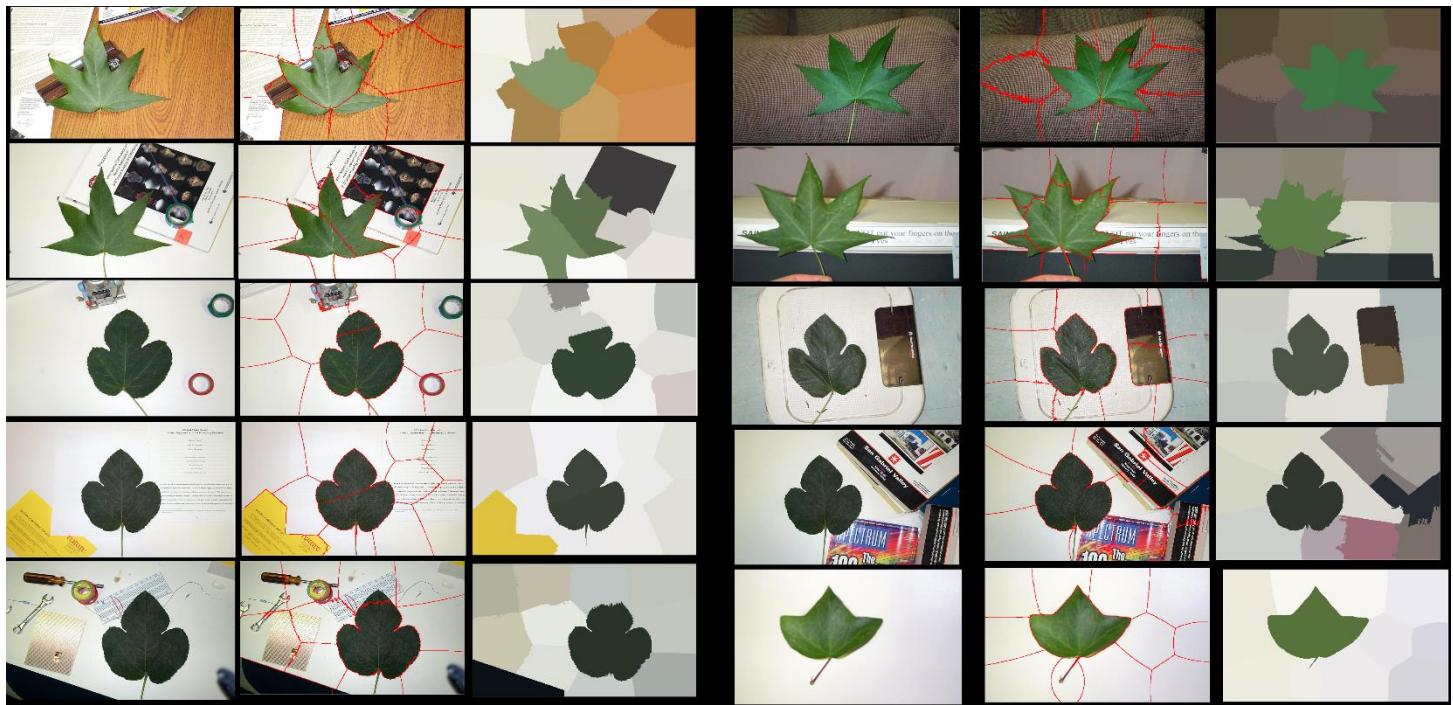
Por fim, as Figuras 15, 16 e 17, mostram a aplicação do método em todas as imagens da base para N=5, N=10 e N=100 em RGB respectivamente. O código é exposto a seguir, lembrando que, no código está definido para o processo ser feito em todas as imagens como é requisitado no exercício.

Figura 15: Superpixel RGB N=5



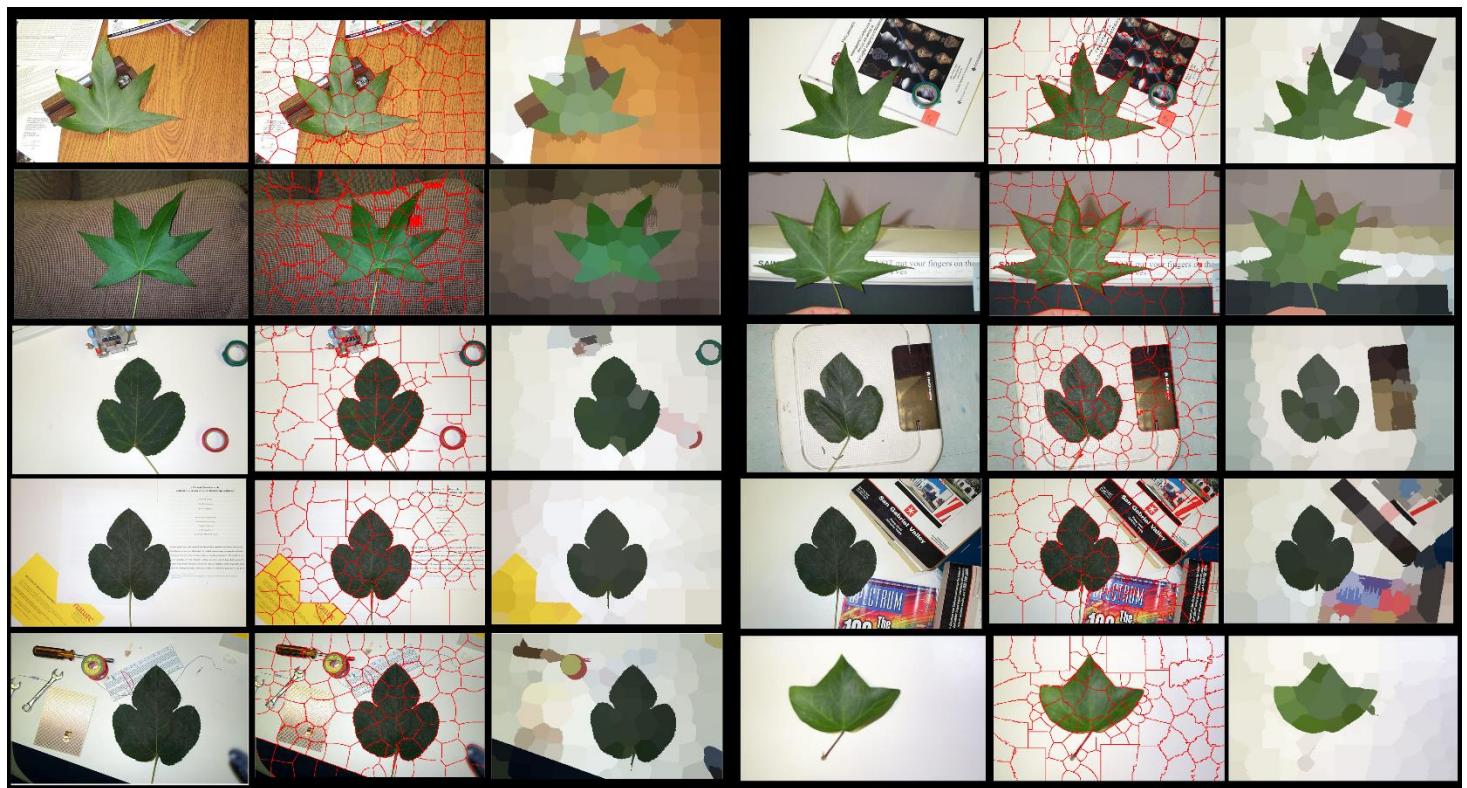
Fonte: Autoral

Figura 16: Superpixel RGB N=10



Fonte: Autoral

Figura 17: Superpixel RGB N=100



Fonte: Autoral

## Código questão 2:

```
%leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg*'));
n=numel(jpgfiles);

%% aplicando processamento de superpixel em todas as imagens da base
N=5; %numero de superpixels
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    %subplot(n,1,i);
    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();

    %subplot(n,1,i+1);
    img2out=super_pixel_RGB(img1,N,i);

    %% mostra a imagem resultante
    %subplot(n,1,i+2);
    imshow(img2out,'InitialMagnification',67);
    title('imagem de superpixel Nimg= ' + string(i) + " Nspixels= " + string(N));
    figure();
end
%%
%% função de superpixel para imagem RGB
%% essa função foi baseada nos conteudos disponíveis em (MATHWORK,2020)
function img2out = super_pixel_RGB (imgRGB,N, numeroImg)
%% dimensões da img1
[x,y,~] = size(imgRGB);

%% separa a imagem em superpixels controlados por N
[L,N] = superpixels(imgRGB,N);

%% Essa função gera uma mascara baseada no array L, de superpixels
%consiste dos limites da imagem para o superpixel
BW = boundarymask(L);

%% imoverlay(img1,BW,'cyan') essa função "queima" uma imagem BW em cima de
%uma imagem A, é como se fosse uma colagem (MATHWORKS,2020);
imshow(imoverlay(imgRGB,BW,'red'),'InitialMagnification',67);
title('conjunto de superpixels Nimg= ' + string(numeroImg) + " Nspixels= " + N);
figure();
%% retorna matriz de zeros == img1
img2out = zeros(size(imgRGB),'like',imgRGB);

%% converte matrizes rotulada para celula de vetores de indice linear
idx = label2idx(L);

%% realiza media dos pixels para cada superpixel nos conjuntos R, G e B
XporY= x*y;
for labelVal = 1:N
    redIdx = idx{labelVal};
    greenIdx = idx{labelVal}+XporY ;
    blueIdx = idx{labelVal}+2*XporY ;
    img2out(redIdx) = mean(imgRGB(redIdx));
    img2out(greenIdx) = mean(imgRGB(greenIdx));
    img2out(blueIdx) = mean(imgRGB(blueIdx));
end
end
```

### 3. Para cada imagem utilize a segmentação por K-means clustering.

**Resposta:**

*K-means clustering* é usado para segmentar a área de interesse de uma imagem do fundo da imagem. Resumidamente, ele “Clusteriza” ou seja, ele forma grupos de dados da imagem os quais tem algum tipo de informação similar entre si, obedecendo o numero de k clusters estabelecidos (CHAUHAN, 2020). O Processo feito em k-means clustering pode ser resumido no seguinte ponto:

- Clusterizar n amostras de dados contendo p características em k grupos;

Uma metodologia simplificada de se produzir um passo a passo para o método de k-means consiste dos seguintes pontos (CHAUHAN, 2020) (MATHWORKS, 2020):

- 1) Escolher o número de clusters k;
- 2) Selecionar de k pontos aleatórios, o centroide;
- 3) Agrupar cada ponto de dados ao centroide mais próximo de cada cluster;
- 4) Calcular e marcar cada novo centroide de cada cluster;
- 5) Reatribua cada ponto de dado a o novo centroide mais próximo.
  - Se tiver alguma nova retribuição, marque e volte ao passo 4;
  - caso não, o modelo está pronto;

Em termos de programação, para se realizar o método de segmentação por k-means clustering, foram examinadas duas funções diferentes, disponíveis ao público em Mathworks (2020). Tais, funções consistem das seguintes:

- **L = imsegkmeans(I,k):** Essa função seguimenta uma imagem I em k clusters por meio de realização de uma clusterização de k-means, ela retorna uma segmentação rotulada na saída, que é uma matriz 2D. essa função pode ser aplicada direto na imagem em RGB:
  - a. **labeloverlay(RGB,L):** é uma função utilizada em conjunto com a supracitada, ela normaliza os valores da matriz L de volta ao formato que pode ser visualizado;
- **idx = kmeans(X,k):** realiza uma clusterização de k-means para particionar uma matriz de dados X(nxp) em k clusters. Esta função retorna um vetor coluna contendo os índices de cada cluster estabelecido. Essa função deve ser utilizada separadamente em R, G

e B ou em uma imagem em escala de cinza. Essa característica, acaba exigindo um tratamento de código adicional, primeiro para R, G e B e depois para normalizar os valores de forma que a imagem possa ser examinada e visualizada;

Depois de textar as duas funções, a que apresentou melhor resultado foi a **imsegkmeans(l,k)**, a segunda apresentou resultados que variavam a cada execução e uma capacidade menor de distinguir os objetos usando os clusters. Como o objetivo é aplicar a função na imagem, não entrarei em detalhes sobre comparações de funções. Talvez seja interessante destacar que, como a versão do matlab da minha máquina é 2017a, não foi possível executar a função, logo, precisei executar o programa no matlab trial online.

Dessa forma, as Figuras 18, 19, 20, 21, 22 e 23 mostram os resultados para as operações de k-means clustering para k=2, k=3 e k=6, como sugerido em Mathworks (2020).

Figura 18: k-means clustering k=2 img1



Fonte: Autoral

Figura 19: k-means clustering k=2 img2



Fonte: Autoral

Figura 20: k-means clustering k=3 img1



Fonte: Autoral

Figura 21: k-means clustering k=3 img2



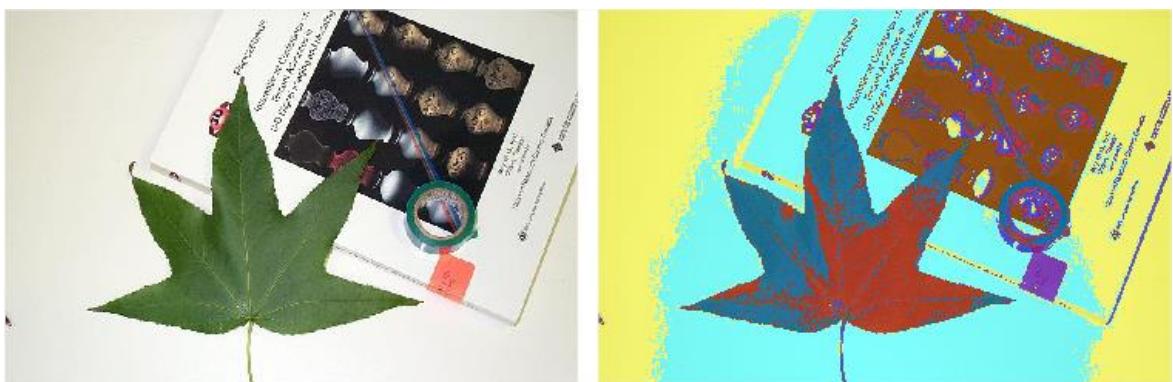
Fonte: Autoral

Figura 22: k-means clustering k=6 img1



Fonte: Autoral

Figura 23: k-means clustering k=6 img2



Fonte: Autoral

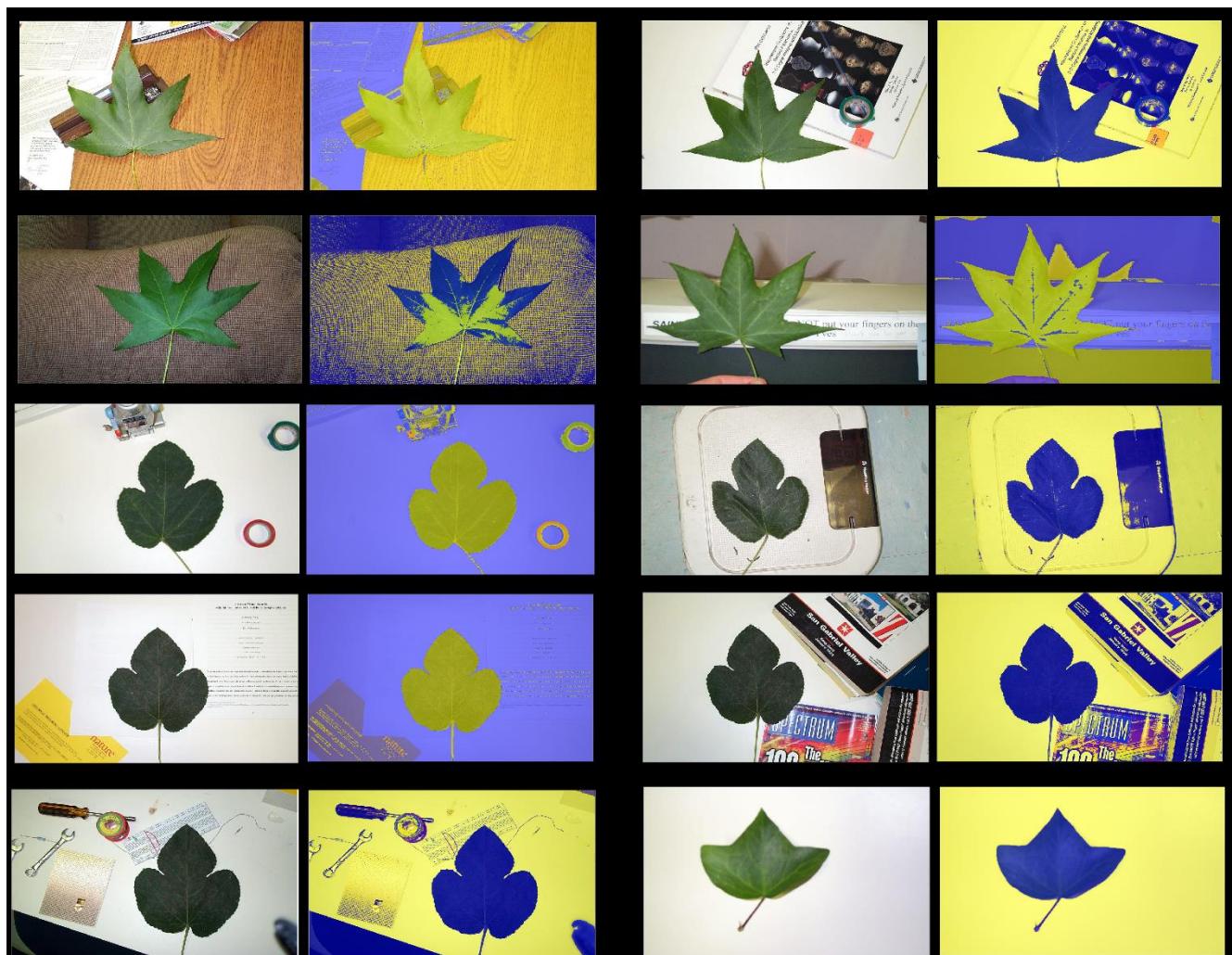
A partir da análise das Figuras 18, 19, 20, 21, 22, 23 é possível discutir os seguintes pontos:

- Se comparado todas as figuras, é possível notar que, para um k=2, as divisões da folha do fundo ficaram mais clara, a função dividiu a imagem em duas cores. Mesmo assim, o resultado foi melhor para a img2, isso pode ser devido a textura do fundo, que na img2 é constante e branca;
- Ao ritmo que k foi incrementado, pode-se notar uma melhora significante na passagem de k=2 para k=3, principalmente para a img2. Por outro lado, para um k=6, como os clusters de cores foram em maior quantidade, percepção visual do objeto não ficou muito boa;

- A escolha do k é muito importante nessa técnica, dado que os resultados podem variar de imagem para imagem de acordo com o k.
- Quanto melhor o contraste entre a folha e o fundo, melhor a separação entre a folha e o fundo. Por exemplo, dado que a folha é verde escura, caso o fundo seja branco, a divisão por k-means gera resultados bons para k=2.

Por tanto, para um teste o método de k-means clustering em todo a base de dados, foi selecionado um K=2. Deste modo, a Figura 24 mostra o método de k-means clustering realizado em cada uma das imagens da base para um k=2.

Figura 24: k-means clustering k=2



Fonte: Autoral

### Código questão 3:

```
% leitura do caminho onde as imagens estão (obs mude a chave de / para \, caso ocorra erro na leitura)
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg*'));
n=numel(jpgfiles);

% aplicando processamento de k-means clustering em todas as imagens da base
K=2; %numero de clusters
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();

    img2out=k_means_clustering (img1,K);

    %% mostra a imagem resultante
    imshow(img2out,'InitialMagnification',67);
    title('imagem de k-means Nimg= ' + string(i) + " K= " + K);
    figure();
end
%%
%% função de k-means clustering para imagem RGB
% essa função foi baseada nos conteudos disponíveis em (MATHWORK,2020)
function img2out = k_means_clustering (imgRGB,K)

%% seguimenta a imagem em k clusters fazendo k-means clustering e retornando
% o seguimento rotulado para L
L = imsegkmeans(imgRGB,K);
%% normaliza as labels da imagem para valores entre 0 e 255 novamente
img2out = labeloverlay(imgRGB,L);
end
```

#### 4. Para cada imagem utilize a segmentação utilizando o método de otsu.

**Resposta:**

A limiarização pode ser tratada como um problema estatístico, onde se tem como objetivo, minimizar o erro médio ocorrido na atribuição de pixels para dois ou mais grupos (classes). Este tipo de entendimento pode ser realizado pelo método de Otsu.

O método de Otsu, por sua vez, é ótimo por conta de *maximizar a variância entre classes*. Essa técnica assume, que as classes com limiares bem estabelecidos devem ser distintas em relação aos valores de intensidade de seus pixels, deste modo, um limiar  $k$  que oferece a melhor separação de intensidade entre as classes, é um limiar ótimo. Além de um  $k$  ótimo, o método de otsu tem a vantagem de se basear em cálculos realizados no histograma da imagem.

O  $k$  que é estabelecido no método de Otsu é um valor entre 0 e  $L-1$ , sendo este, restrinido pelas seguintes regras:

- $0 < k < L-1$ ;

A princípio a técnica de Otsu busca separar a imagem em duas classes distintas,  $C_1$  e  $C_2$ , essa separação é feita por meio de uma série de cálculos estatísticos e probabilísticos. Estes cálculos buscam encontrar o  $k$ -ótimo, por meio dos cálculos de variância e moda valores e probabilidades das intensidades da imagem, com também, pela avaliação da função mostrada na Figura 25.

Figura 25: Equação da variância entre classes

$$\sigma_B^2(k*) = \max_{0 \leq k \leq L-1} \sigma_B^2(k)$$

Fonte: (GONZALEZ, WOODS, 2010)

Para encontrar o  $k$ -ótimo, basta avaliar a equação da Figura 20 para todos os valores de  $k$  (obedecendo  $0 < P_1(k) < 1$ ) e selecionando o valor de  $k$  que produz o **máximo da variância ao quadrado de  $k$** . Caso o máximo exista para mais de um valor, normalmente é feita uma média entre esses valores e é definido o  $K$ -ótimo.

Uma vez que se tenha estabelecido o K, é aplicado uma limiarização da imagem utilizando k como limiar, a Figura 26 mostra a regra dessa limiarização.

Figura 26: limiarização por K

$$g(x,y) = \begin{cases} 1 & \text{se } f(x,y) > k * \\ 0 & \text{se } f(x,y) \leq k * \end{cases}$$

Fonte: (GONZALEZ, WOODS, 2010)

Cabe Lembrar que, todos cálculos probabilísticos e de variância foram realizados utilizando o histograma da imagem.

O que foi explicado até o momento consiste da aplicação do método de Otsu em uma segmentação global. Para aprimorar o método, pode-se, ainda, aplica-lo em partes da imagem de forma separada, para tanto é preciso particionar a imagem em N partes retangulares e aplicar nestes, o método de ostu. Essa abordagem é utilizada para compensar a não uniformidade de iluminação e/ou a refletância.

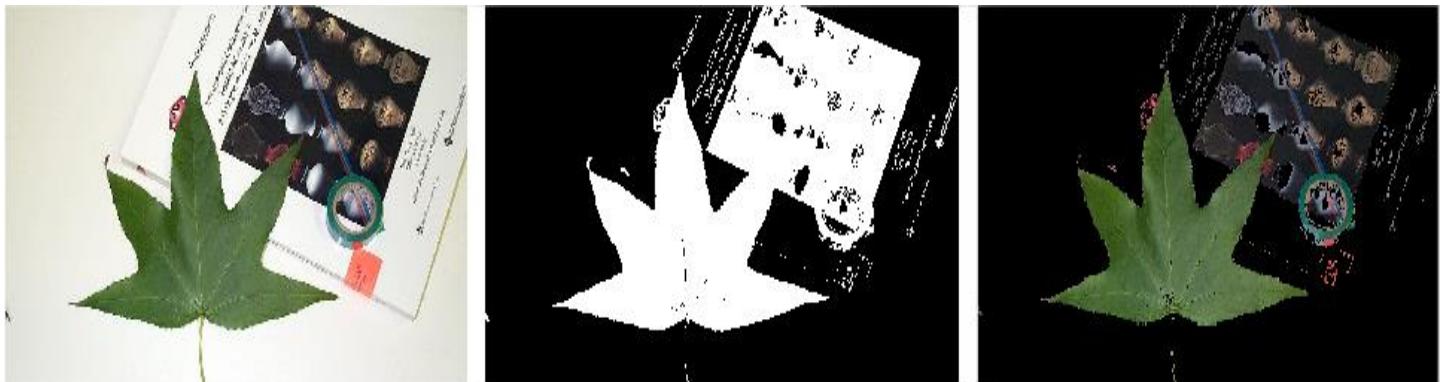
Baseado nos conceitos supracitados, foi desenvolvido uma função de otsu, para dividir as imagens em dois campos, fundo e objeto. Para tanto, foi utilizado a função **graythresh(imgGray)**, a qual faz uso do método de otsu para retornar um valor k entre 1 e 0, que foi utilizado para aplicar uma limiarização global com k na imagem. As Figuras 27, 28 e 29 mostram as imagens originais, seguidas da máscara gerada com otsu e, pôr fim, a imagem final com a aplicação da máscara.

Figura 27: Segmentação por otsu img1



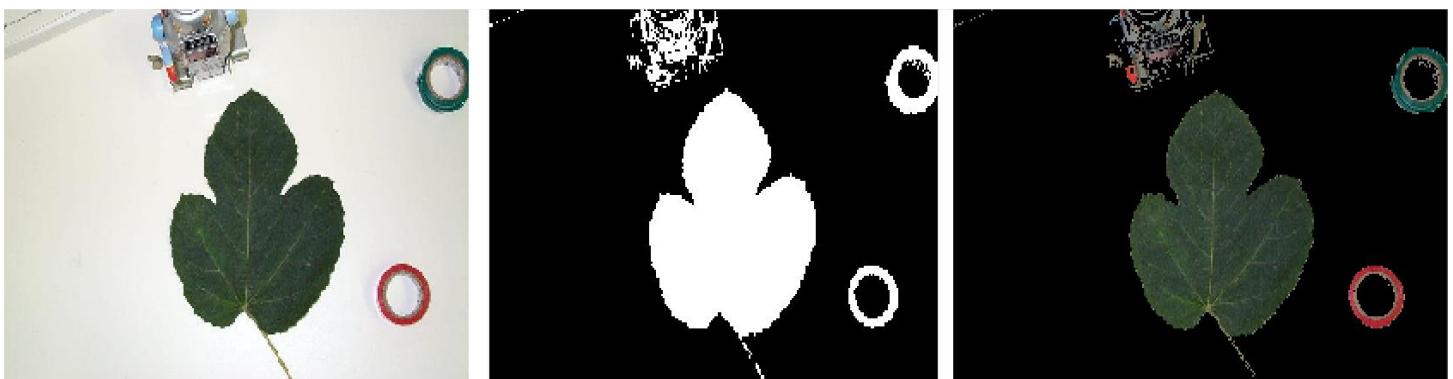
Fonte: Autoral

Figura 28: Segmentação por otsu img2



Fonte: Autoral

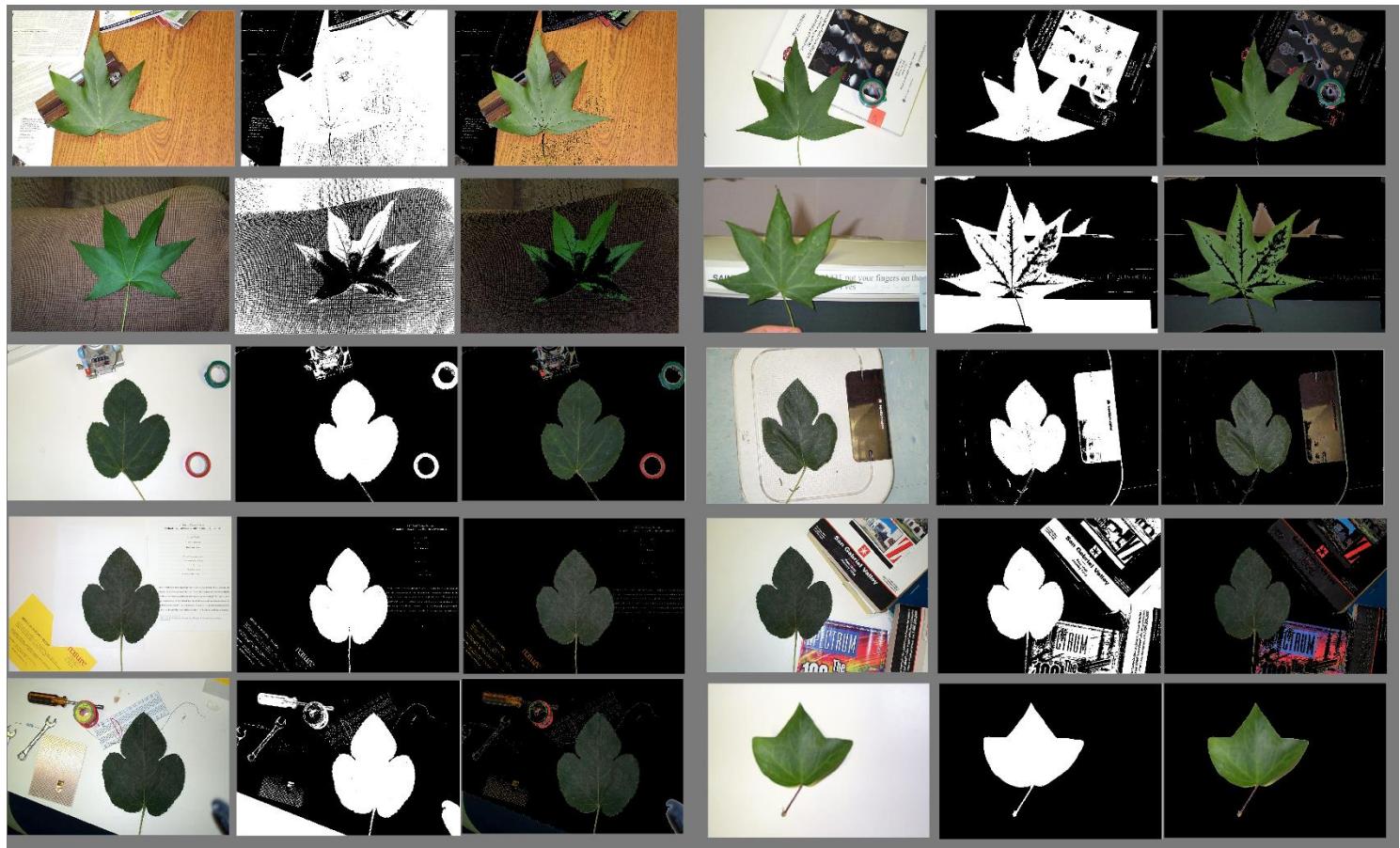
Figura 29: Segmentação por otsu img3



Fonte: Autoral

Por meio da análise das Figuras 27, 28 e 29, é possível notar que, o método de otsu aplicado de forma global para  $n=2$  segmentos, conseguiu separar a folha do fundo de forma mais clara, em imagens onde o fundo tem uma cor constante e branco. A Figura 30, mostra o resultado da aplicação do método de otsu em todas as imagens do conjunto de dados.

Figura 30: método de Otsu



Fonte: Autoral

## Código questão 4:

```
%% leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg*'));
n=numel(jpgfiles);

%% aplicando o metodo de otsu em todas as imagens da base
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    %subplot(n,1,i);
    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();

    %subplot(n,1,i+1);
    img2out=otsu(img1,i);

    %% mostra a imagem resultante
    %subplot(n,1,i+2);
    imshow(img2out,'InitialMagnification',67);
    title('imagem resultante, Nimg= ' + string(i));
    figure();

end
%%

%% função de otsu para imagem RGB
% essa função foi baseada nos conteúdos disponíveis em (MATHWORK, 2020)
% e (GONZALEZ, WOODS, 2010)
% a função aplica um otsu de forma global, buscando dividir a imagem em 2
% seguimentos, fundo e objeto

function img2out = otsu(img1,i)
% converte para escala de cinza
    imgGray = rgb2gray(img1);
% Encontrar a mascara utilizando otsu
% essa função retorna um k para servir de limiar para a img1
% usando o método de otsu
k = graythresh(imgGray);
%normaliza o valor de k
k = k * 255;
% aplica a limiarização usando k obtido com o método de otsu
mascara = uint8( (imgGray < k) * 255);
imshow(mascara);
title('masck, Nimg= ' + string(i));
figure();
%% normalizando a mascara
mascara01 = mascara/255;
%% aplicando a mascara na img1
img2out = mascara01 .* img1;
end
%%
```

**5. Para cada imagem utilize a segmentação utilizando o método de médias moveis.**

**Resposta:**

Um caso especial de limiarização local, é a utilização de média móvel ao longo das linhas de digitalização de uma imagem. Esse tipo de método é muito útil para processamento de documentos, já que é um método rápido. A digitalização normalmente obedece a um padrão linha por linha em zigue-zageu para reduzir o viés de iluminação (GONZALEZ, WOODS, 2010).

Tendo em vista esse movimento, média móvel consiste de se utilizar uma equação de media que calcula a média conforme a digitalização vai ocorrendo. Desta forma, a equação da média toma a forma mostrada na Figura 31 (GONZALEZ, WOODS, 2010). Essa equação leva em conta que,  $Z(k+1)$  denota a intensidade do ponto encontrado na sequência de digitalização na etapa  $k+1$ , então seria uma intensidade média.

Figura 31: equação de médias moveis

$$\begin{aligned} m(k+1) &= \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i \\ &= m(k) + \frac{1}{n} (z_{k+1} - z_{k-n}) \end{aligned}$$

Fonte: (GONZALEZ, WOODS, 2010)

Sendo:

- N: número de pontos utilizado no cálculo da média;
- $M(1)= Z1/n$ ;

Já que a média móvel é calculada para cada ponto da imagem, a segmentação é implementada utilizando a equação da Figura 35 com  $Tx = BMxy$ , para os qual:

- B é a constante;
- $Mxy$  é a média móvel da equação da Figura 31 no ponto  $(x,y)$  na imagem de entrada;

Figura 32: limiarização com Txy

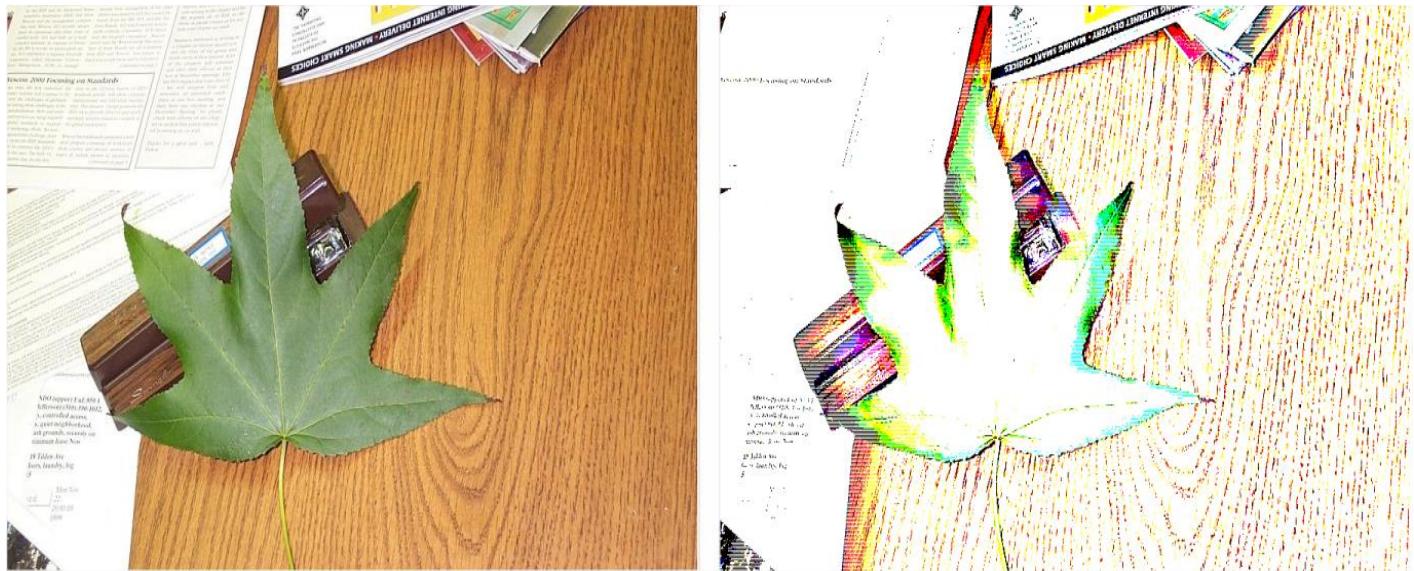
$$g(x, y) = \begin{cases} 1 & \text{se } f(x, y) > T_{xy} \\ 0 & \text{se } f(x, y) \leq T_{xy} \end{cases} \quad (10.3-35)$$

Fonte: (GONZALEZ, WOODS, 2010)

Desta forma, baseando-se nos conceitos supracitados, foi desenvolvido um script em Matlab que aplica a função **movingthresh( f1 , n , K )**, que é exposta em (GONZALEZ; WOODS; EDDINS, 2009, p. 575). Foi feito uma adaptação na forma de utilizar a função, onde a mesma foi aplicada nos elementos R, G e B da imagem para os resultados serem mostrados nesse modo de cores.

Sendo assim, as Figuras 33, 34 e 35 mostram os resultados da aplicação da função de médias moveis em duas imagens diferentes. Os autores indicaram valores padrões para n e B, sendo n=20 e B=0.5, porém, como estes parâmetros estavam sendo utilizados para segmentar fotos de documentos, eles foram alterados por meio de testes sucessivos para n=50 e B=0.734.

Figura 33: Aplicação de moving averages, n=50, B=0.734 img1



Fonte: Autoral

Figura 35: Aplicação de moving averages, n=50, B=0.734 img2



Fonte: Autoral

Figura 35: Aplicação de moving averages, n=50, B=0.734 img3



Fonte: Autoral

A partir da análise das Figura 33, 34 e 35 é possível destacar os seguintes pontos:

- É possível notar a marcação das cores em forma de riscos horizontais, isso pode ser devido a teoria onde se utiliza a compensação do movimento de zig-zag para se realizar a aplicação das médias móveis;
- De certa forma, o método conseguiu destacar de forma razoável a folha para fundos com muitas texturas. Por outro lado, em comparação com métodos como superpixel ou otsu, para fundos brancos, ele não

foi tão efetivo, mesmo que tenha destacado bem a silhueta da folha, como pode ser visto na Figura 35;

- O método também foi executado relativamente mais rápido que os outros, o que confirma o que foi dito na teoria como justificativa para ele ser utilizado na digitalização de documentos;

A seguir, a Figura 36 expõe a aplicação do método das médias móveis em todas as 10 imagens da base.

Figura 36: médias moveis nas imagens da base n=50, B=0.734



Fonte: autoral

## Código questão 5:

```
%% leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg*'));
n=numel(jpgfiles);

%% aplicando o metodo de otsu em todas as imagens da base
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    %subplot(n,1,i);
    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();

    %subplot(n,1,i+1);
    imgGray= rgb2gray(img1);
    img2out=zeros(size(img1,1),size(img1,2),3);
    img2out(:,:,1)= movingthresh( img1(:,:,1), 50,0.734);
    img2out(:,:,2)= movingthresh( img1(:,:,2), 50,0.734);
    img2out(:,:,3)= movingthresh( img1(:,:,3), 50,0.734);

    %% mostra a imagem resultante
    %subplot(n,1,i+2);
    imshow(img2out);
    title('imagem resultante, Nimg= ' + string(i));
    figure();
end
%%

%% essa função foi baseada nos conteudos disponíveis em (MATHWORK,2020)
%,(GONZALEZ, WOODS, 2010)
%%
%% função demostrada no livro
%n: é a quantidade de pontos utilizado nas médias moveis
%B: é uma constante que tem seu valor no intervalo de [0,1]
function g = movingthresh( f1,n , B )
%% segmentação movingthresh usando metodo das médias móveis.
%   g = movingthresh( f1,n , B ) seguimenta uma imagem F por meio de
%   thresholding de intensidades baseado na media moveis das intensidades
%   ao longe das linhas individuais de uma imagem. A média no pixel B é
%   formada pela media de intensidades do pixel e dos n-1 processamento
%   dos vizinhos. para reduzir efeito bias, o escaneamento é feito em
%   zig-zag, tratando o pixel como se eles fossem um vetor contínuo de
%   dados 1-D. se o valor de uma imagem no ponto exeder k% de um valor da
%   média em execução naquele ponto, o valor 1 é colocado nessa posição na
%   matrix g. Caso contrário 0 é colocado nessa posição. No final do
%   processo, g é a imagem segmentada. k deve ser um escalar entre 0 e 1;

%% Preliminares.
f = im2double(f1) ;
[M, N] = size(f) ;
if ( n < 1 ) || (rem(n, 1) ~= 0 )
    error( ' n must be an integer > = 1 . ' );
end
if (B < 0 || B > 1)
    error ( ' K must be a fraction in the range [ 0 , 1 ] . ' )
end
%% Gira todas as linhas de um imagem f para processar o equivalente ao
%% zig-zag
f(2:2:end,:)=fliplr( f( 2 : 2 : end , : ) ) ;
f = f'; % ainda é uma matriz
f = f (:)' ; % Converte para um vetor de linha para usar na função filter
%% calcula a media movel
maf = ones ( 1 , n ) / n ; % filtro média movel 1-D
ma = filter( maf , 1 , f ) ; % calculo da medeia movel
%% realiza o thesholding
g = f > B * ma ;
%% volta para o formato original da imagem
g = reshape ( g , N , M )';
% gira as linhas de volta a posição normal
g (2:2:end , : ) = fliplr (g(2:2:end,:));
end
```

**6. Para cada imagem utilize a segmentação por region grow.**

**Resposta:**

É um procedimento que agrupa os pixels ou as sub-regiões em regiões maiores com base em critérios predefinidos para o crescimento. Resumidamente, o processo consiste de se selecionar um conjunto de pontos iniciais chamado de “semente” e, a partir deles, fazer as regiões crescerem anexando a cada semente aqueles pixels vizinhos que tem propriedades predefinidas semelhantes à das sementes (GONZALEZ, WOODS, 2010).

Quando uma informação *a priori* não estiver disponível, o procedimento é calcular em cada pixel o mesmo conjunto de propriedades que, em última análise, serão utilizadas para designar os pixels das regiões durante o processo de crescimento (GONZALEZ, WOODS, 2010).

Um problema do método poderia ser quando se usa imagens monocromáticas, já que o método leva em conta intensidades de cores. Nesse caso, a análise de região deve ser realizada com um conjunto de indicadores baseados nos níveis de intensidade nas propriedades espaciais. Outro problema desse método é a formulação de um método de parada. Este, por sua vez, pode seguir critérios como tamanho, semelhança entre um pixel candidato e os pixels selecionados até o momento e o formato da região que está sofrendo o crescimento (GONZALEZ, WOODS, 2010).

Com base na descrição exposta até o momento, um algoritmo básico de crescimento da região, baseia-se em conectividade-8 pode ser estabelecido segundo Gonzalez, Woods (2010) como mostrado na Figura 37.

Figura 37: algoritmo de crescimento de regiões

1. Encontrar todos os componentes conectados em  $S(x, y)$  e erodir cada componente conectado a um pixel; rotular todos os pixels encontrados com o número 1. Todos os outros pixels em  $S$  recebem 0.\*
2. Formar uma imagem  $f_Q$  que, em um par de coordenadas  $(x, y)$ , deixe  $f_Q(x, y) = 1$  se a imagem de entrada satisfaz a propriedade determinada,  $Q$ , nessas coordenadas; caso contrário, deixe  $f_Q(x, y) = 0$ .
3. Digamos que  $g$  é uma imagem formada anexando a cada semente em  $S$  todos os pontos rotulados com o número 1 em  $f_Q$  que estão 8-conectados a essa semente.
4. Rotular cada componente conectado em  $g$  com uma diferente etiqueta de região (por exemplo, 1, 2, 3, ...). Esta é a imagem segmentada obtida pelo crescimento de região.

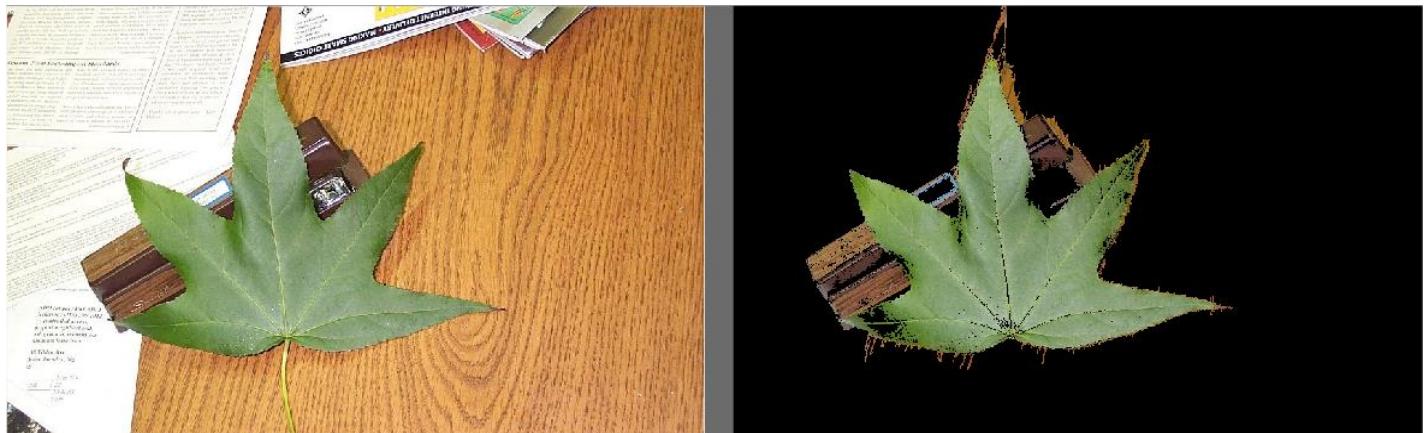
Fonte: (GONZALEZ, WOODS, 2010).

Deste modo, foi desenvolvido um algoritmo em Matlab que faz uso de uma função disponível em MathWorks(2020), que possibilita a aplicação de crescimento de regiões em uma imagem. As condições de entrada para a função são:

- **I:** imagem de entrada;
- **x e y:** as coordenadas do ponto onde se inicia a semente (`seedpoint(x,y)`). Com as definições anteriores, pode-se dizer que, o resultado da segmentação depende muito de onde esse ponto seed vai ser colocado, já que o objetivo é separar a folha do fundo, o ideal é que este ponto esteja sempre em cima de alguma parte da folha;
- **reg\_maxdist:** intensidade máxima da distância, 0.2 é o padrão;

As Figuras 38, 39 e 40 mostram a aplicação de segmentação por crescimento de região em 3 imagens do banco de dados disponibilizados para a lista. É importante notar que as duas primeiras imagens estão com o mesmo ponto de *seed*, a última está com um ponto diferente, o objetivo é mostrar o efeito dessa mudança no resultado final.

Figura 38: *region growing img1*, seedpoint( $\text{size(img1,1)}/2$ ,  $\text{size(img1,2)}/2$ )



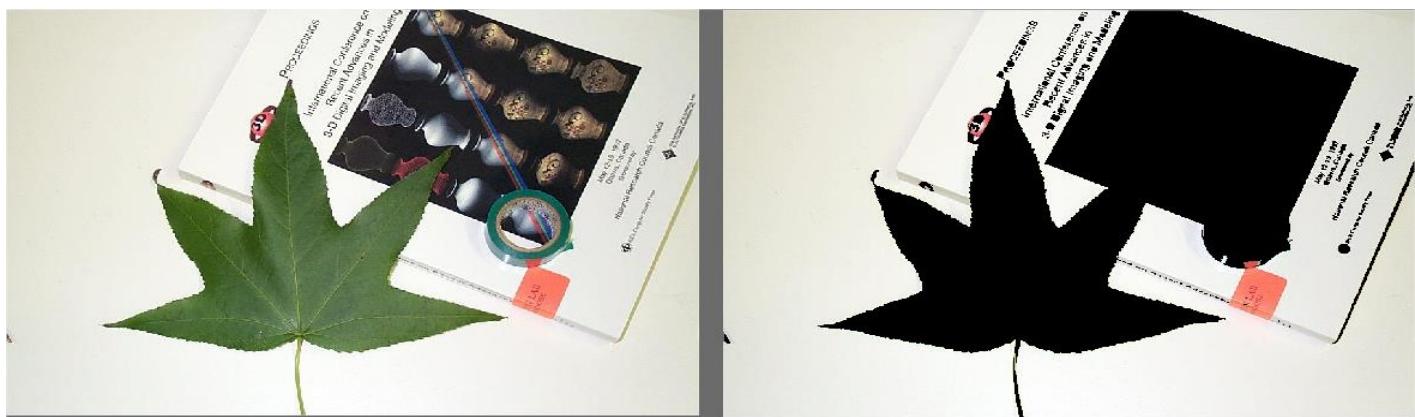
Fonte: Autoral

Figura 39: *region growing img2*, seedpoint( $\text{size(img1,1)}/2$ ,  $\text{size(img1,2)}/2$ )



Fonte: Autoral

Figura 40: *region growing img2*, seedpoint(100,100)



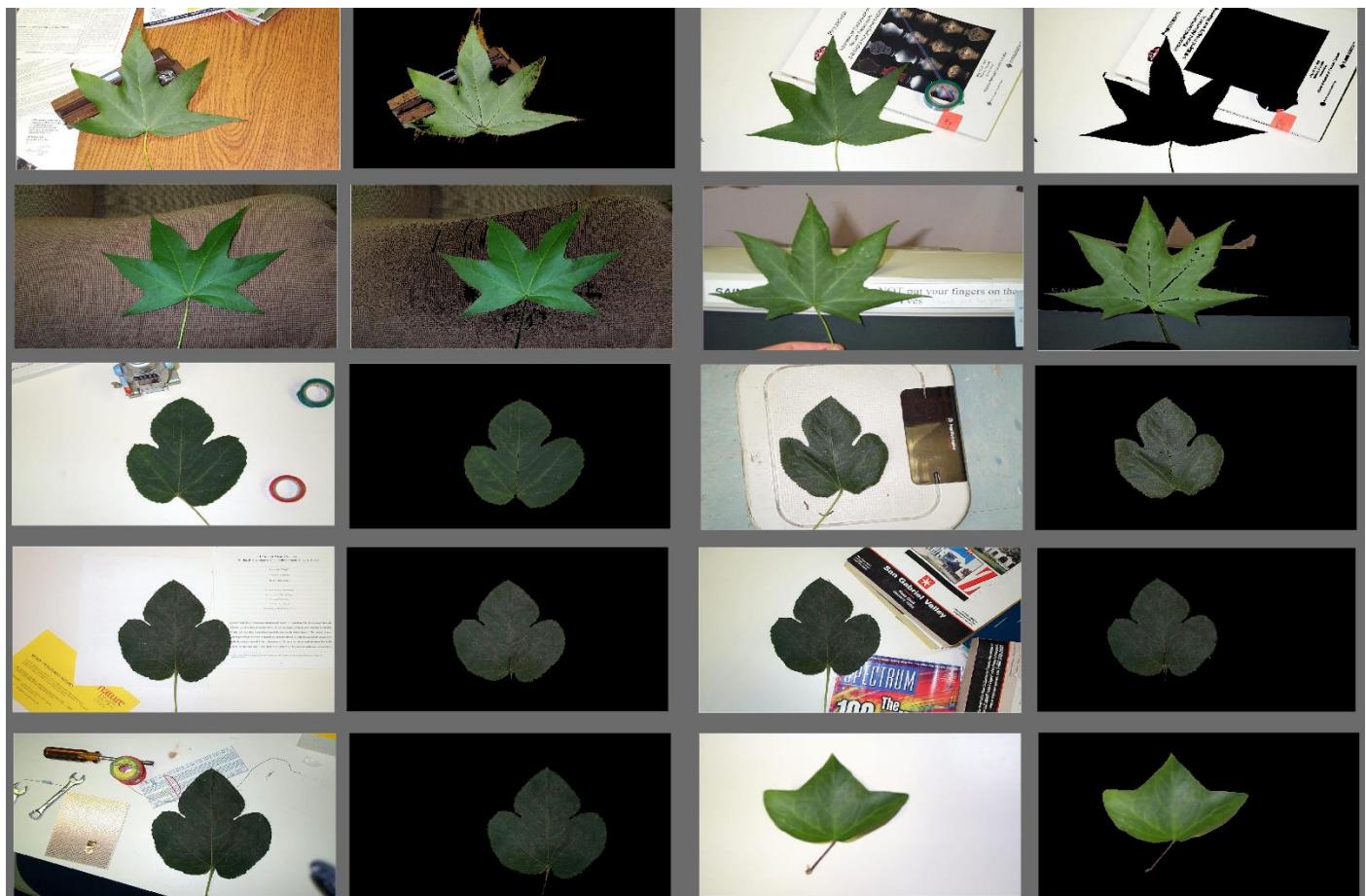
Fonte: Autoral

Analizado as Figuras 38, 39 e 40 é possível destacar os seguintes pontos:

- O método region growing conseguiu separar a folha do fundo com boa qualidade nos dois primeiros casos;
- Comparando a Figura 40 com a Figura 39, é possível notar que, a mudança do seedpoint gerou um resultado completamente diferente para as duas imagens. Isso se deve a propriedade de crescimento em direção dos conjuntos de sementes que sejam semelhantes, ou seja, o ponto de semente deve ter caído sobre a parte branca, logo, toda a parte branca foi selecionada na Figura 39. Deste modo, o ideal é que o seedpoint(x,y), sempre se inicie em cima da folha.

A Figura 41 mostra o resultado do método region grow aplicado sobre todas as imagens da base.

Figura 41: region grown, seedpoint(size(img1,1)/2, size(img1,2)/2)



Fonte: Autoral

## Código questão 6:

```
%% leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg'));
n=numel(jpgfiles);

%% aplicando o metodo de crescimento de região em todas as imagens da base
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();

    %% configurando os parâmetros da função regiongrowing()
    %pondo de seed definido de forma aleatória, o ideal é que esse ponto
    % seja sempre em cima da folha
    %a função region growing irá retornar um amascara 0/1 que pode ser
    %utilizada para extrair o Roi, da imagem por meio da aplicação dela na
    %imagem.
    x=size(img1,1)/2;
    y=size(img1,2)/2;
    img2out = regiongrowing(img1,x,y,0.2);

    %% mostra a imagem resultante
    imshow(img2out);
    title('imagem resultante, Nimg= ' + string(i));
    figure();
end
%%

%% função de region growing coletada de MathWorks(2020)
% Aplica o processo de crescimento de região iniciando o agrupamento de
% sementes do ponto x,y
function img2out=regiongrowing(img1,x,y,reg_maxdist)

I = im2double(img1);

if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
if(exist('y','var')==0), figure, imshow(I,[ ]); [y,x]=getpts; y=round(y(1)); x=round(x(1)); end
J = zeros(size(I,1),size(I,2)); % Output
Isizes = size(I); % Dimensions of input image
reg_mean = I(x,y); % The mean of the segmented region
reg_size = 1; % Number of pixels in region
% Free memory to store neighbours of the (segmented) region
neg_free = 10000; neg_pos=0;
neg_list = zeros(neg_free,3);
pixdist=0; % Distance of the region newest pixel to the region mean
% Neighbor locations (footprint)
neigb=[-1 0; 1 0; 0 -1;0 1];
% Start regiongrowing until distance between region and possible new pixels become
% higher than a certain threshold
while(pixdist<reg_maxdist&&reg_size<numel(I))
    % Add new neighbors pixels
    for j=1:4
        % Calculate the neighbour coordinate
        xn = x +neigb(j,1); yn = y +neigb(j,2);

        % Check if neighbour is inside or outside the image
        ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));

        % Add neighbor if inside and not already part of the segmented area
        if(ins&&(J(xn,yn)==0))
            neg_pos = neg_pos+1;
            neg_list(neg_pos,:)= [xn yn I(xn,yn)]; J(xn,yn)=1;
        end
    end
    % Add a new block of free memory
    if(neg_pos+10>neg_free), neg_free=neg_free+10000; neg_list((neg_pos+1):neg_free,:)=0; end

    % Add pixel with intensity nearest to the mean of the region, to the region
    dist = abs(neg_list(1:neg_pos,3)-reg_mean);
    [pixdist, index] = min(dist);
    J(x,y)=2; reg_size=reg_size+1;

    % Calculate the new mean of the region
    reg_mean= (reg_mean*reg_size + neg_list(index,3))/(reg_size+1);

    % Save the x and y coordinates of the pixel (for the neighbour add process)
    x = neg_list(index,1); y = neg_list(index,2);

    % Remove the pixel from the neighbour (check) list
    neg_list(index,:)=neg_list(neg_pos,:); neg_pos=neg_pos-1;
end
% Return the segmented area as logical matrix
J=J>1;

img2out = uint8(J) .*img1;
end
```

**7. Crie sua própria rotina de segmentação para as imagens combinando alguns métodos de sua preferência, presentes ou não na lista. Justifique o uso dos métodos escolhidos.**

**Resposta:**

Por meio de sucessivos testes utilizando combinações entre os métodos de segmentação expostos nesta lista, foi possível notar que, a combinação entre o método de **otsu** e o de **region growing** gerou um resultado melhor que combinações como **otsu** e **k-means clustering** ou **otsu** e **superpixel**. Com o propósito de mostrar esses testes, as Figura 42, 43 e 44, mostram as combinações de otsu/k-means clustering, otsu/superpixel e otsu/region growing respectivamente.

Figura 42: otsu/k-means clustering k=2



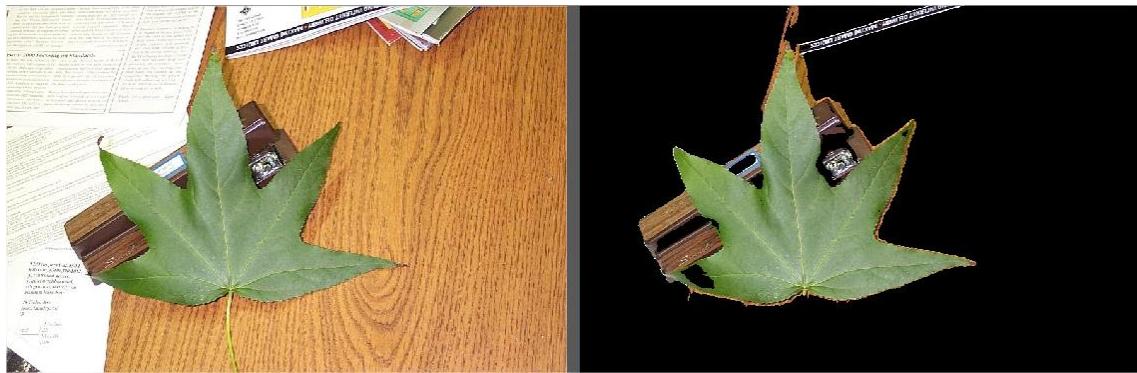
Fonte: Autoral

Figura 43: otsu/superpixel



Fonte: Autoral

Figura 44: otsu/region growing



Fonte: Autoral

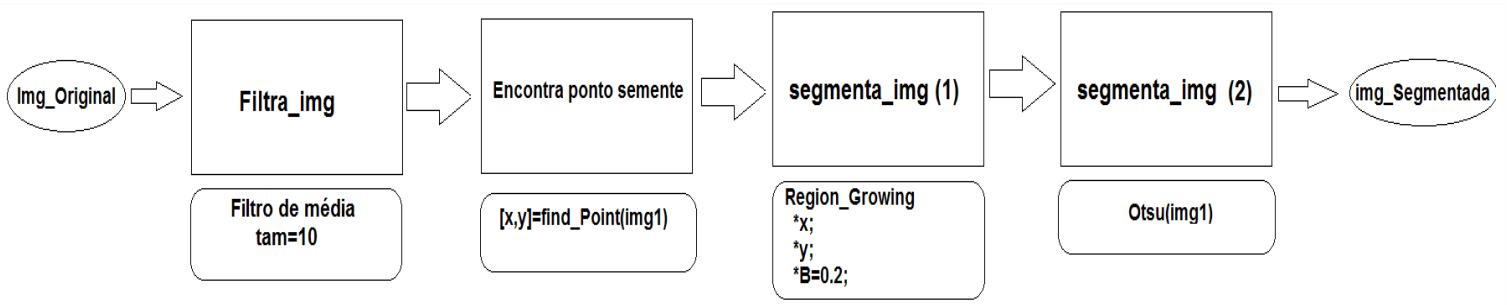
De forma geral, o objetivo dos testes acima, foi verificar uma maneira de obter a área de interesse, no caso a folha, em imagens com fundo desfavoráveis, como é o caso da imagem 1 da base de dados. Então, como pode ser vista nas figuras supracitadas, a combinação feita para a Figura 44 se mostrou melhor.

Como o método de **region growing** depende muito de se iniciar com um bom ponto de semente, foi desenvolvido um método simples chamado de **find\_point(img1)**. Tal método consiste dos seguintes passos:

- 1) Zerar as regiões de cores G e B da imagem;
- 2) Realizar um rastreio diagonal na imagem (i,i), para encontrar um ponto que esteja em cima da folha. A ideia é que os pontos em cima da folha estejam com valor de intensidade abaixo de 100;
- 3) Fazer uma pequena verificação dos pontos vizinhos para assegurar se o ponto está em cima da folha;
- 4) Retornar o ponto [x,y];

Para uma melhora sutil na imagem final, foi combinado também um filtro de média com tam=10. A ideia é que ele suavize a borda das folhas e o fundo, de modo que seja mais simples encontrar a folha com método utilizados. Deste modo, o fluxograma do método de segmentação desenvolvido é Mostrado na Figura 45.

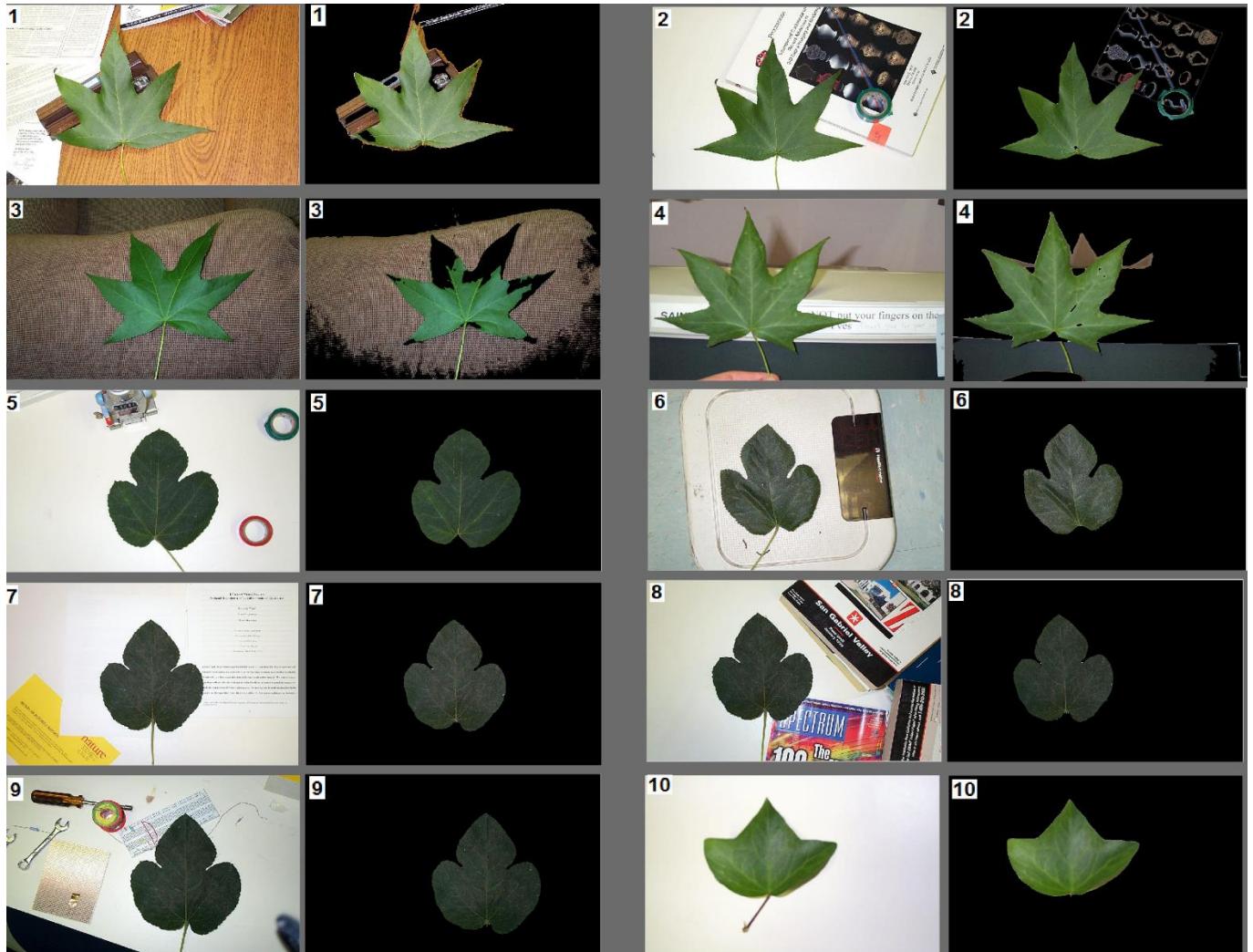
Figura 45: Fluxograma segmentação própria



Fonte: Autoral

Deste modo, o resultado para a filtragem de todas as imagens usando o método criado é mostrado na Figura 46.

Figura 46: Filtragem total método próprio



Fonte: Autoral

Como conclusão, pode se dizer que, o método desenvolvido para a questão 7 conseguiu separar a folha do fundo de forma efetiva em pelo menos 7 das 10 imagens da base. O pior resultado foi da imagem 3, que tem um fundo com muita textura e mais escuro. A seguir é apresentado o código da questão 7.

## Código questão 7 Parte1:

```
close all;
%% leitura do caminho onde as imagens estão
images ='folhas';
jpgfiles=dir(fullfile(images,'*.jpg'));
n=numel(jpgfiles);

%% aplicando a segmentação em todas as imagens da base
for i=1:n
    nomeImg=jpgfiles(i).name;
    img1=imread(fullfile(images,nomeImg));

    %% mostra imagem original
    imshow(img1);
    title('imagem original Nimg= ' + string(i));
    figure();
    %% filtra imagem
    img2out= Filtro_Q3_Lista3(img1, 10);
    %% encontra ponto semente
    [x,y] = find_Point(img2out);
    %% aplica regionGrowing
    img2out=RegionGrowingMine(img2out,x,y,0.2);
    %% aplica otsu
    img3out=otsu(img2out,img1,i);

    %% mostra a imagem resultante
    imshow(img3out,'InitialMagnification',67);
    title('imagem final Nimg= ' + string(i));
    figure();
end
%%
function img2out = otsu(img1,img2,i)
% converte para escala de cinza
    imgGray = rgb2gray(img1);
% Encontrar a mascara utilizando otsu
% esse função retorna um k para servir de limiar para a img1
% essa função usa o método de otsu
k = graythresh(imgGray);
%normaliza o valor de k
k = k * 255;
%% aplica a limiarização usando k obtido com o método de otsu
% mascara = uint8( (imgGray < k) * 255);
mascara=zeros(size(img1,1),size(img1,2),'uint8');
for i=1: size(img1,1)
    for j=1: size(img1,2)
        if(imgGray(i,j,:)>k)
            mascara(i,j)=1;
        end
    end
end
%% aplicando a mascara na
img2out = mascara .* img2;
end
%%
%% filtro
function [ imgFiltred ] = Filtro_Q3_Lista3( img1, Tam)
Mascara = 1/(Tam*Tam) * ones(Tam,Tam);
imgFiltred= imfilter(img1,Mascara,'replicate');
end
%%
```

## Código questão 7 Parte 2:

```
%% essa função rastreia a imagem para encontrar um bom ponto de semente
function [x,y] = find_Point(img1)

    img12=img1;
    img12(:,:,2)=0;
    img12(:,:,3)=0;
    [x1,y1,~] = size(img12);

    cont=0;
    for i=1 : x1
        if(img12(i,i)<100)
            cont=cont+1;
        end
        if(cont==22)
            y=i;
            x=i;
            break;
        end
    end

    end
%%%
%%%
function [ img2out ] = RegionGrowingMine(img1,x,y,reg_maxdist)

I = im2double(img1);

if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
if(exist('y','var')==0), figure, imshow(I,[]); [y,x]=getpts; y=round(y(1)); x=round(x(1)); end
J = zeros(size(I,1),size(I,2)); % Output
Isizes = size(I); % Dimensions of input image
reg_mean = I(x,y); % The mean of the segmented region
reg_size = 1; % Number of pixels in region
% Free memory to store neighbours of the (segmented) region
neg_free = 10000; neg_pos=0;
neg_list = zeros(neg_free,3);
pixdist=0; % Distance of the region newest pixel to the region mean
% Neighbor locations (footprint)
neigb=[-1 0; 1 0; 0 -1; 0 1];
% Start regiongrowing until distance between region and possible new pixels become
% higher than a certain treshold
while(pixdist<reg_maxdist&&reg_size<numel(I))
    % Add new neighbors pixels
    for j=1:4
        % Calculate the neighbour coordinate
        xn = x +neigb(j,1); yn = y +neigb(j,2);

        % Check if neighbour is inside or outside the image
        ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));

        % Add neighbor if inside and not already part of the segmented area
        if(ins&&(J(xn,yn)==0))
            neg_pos = neg_pos+1;
            neg_list(neg_pos,:) = [xn yn I(xn,yn)]; J(xn,yn)=1;
        end
    end
    % Add a new block of free memory
    if(neg_pos+10>neg_free), neg_free=neg_free+10000; neg_list((neg_pos+1):neg_free,:)=0; end

    % Add pixel with intensity nearest to the mean of the region, to the region
    dist = abs(neg_list(1:neg_pos,3)-reg_mean);
    [pixdist, index] = min(dist);
    J(x,y)=2; reg_size=reg_size+1;

    % Calculate the new mean of the region
    reg_mean= (reg_mean*reg_size + neg_list(index,3))/(reg_size+1);

    % Save the x and y coordinates of the pixel (for the neighbour add process)
    x = neg_list(index,1); y = neg_list(index,2);

    % Remove the pixel from the neighbour (check) list
    neg_list(index,:)=neg_list(neg_pos,:); neg_pos=neg_pos-1;
end
% Return the segmented area as logical matrix
J=J>1;
img2out = uint8(J) .*img1;
end
%%
```

## REFERÊNCIAS

CHAUHAN, Nagesh Singh. **Introduction to Image Segmentation with K-Means clustering.** Disponível em: <https://towardsdatascience.com/introduction-to-image-segmentation-with-k-means-clustering-83fd0a9e2fc3>. Acesso em: 02 maio 2020.

GONZALEZ, Rafael; WOODS, Richard; EDDINS, Steven. Image segmentation. In: GONZALEZ, Rafael; WOODS, Richard; EDDINS, Steven. **Digital image processing using MATLAB.** 2. ed. USA: Gastesmark, 2009. cap. 11, p. 575-578.

GONZALEZ, Rafael C., WOODS, Richard E. **Processamento Digital de Imagem.** São Paulo: Sv, 2010.

MATHWORKS. **Superpixels.** 2020. Disponível em: <https://www.mathworks.com/help/images/ref/superpixels.html>. Acesso em: 01 maio 2020.

MATHWORKS. **Imsegkmeans.** 2020. Disponível em: <https://www.mathworks.com/help/images/ref/imsegkmeans.html>. Acesso em: 02 maio 2020.

MATHWORKS. **kmeans.** 2020. Disponível em: <https://www.mathworks.com/help/stats/kmeans.html>. Acesso em: 02 maio 2020.

STUTZ, David; HERMANS, Alexander; LEIBE, Bastian. Superpixels: an evaluation of the state-of-the-art.: An evaluation of the state-of-the-art. **Computer Vision And Image Understanding**, [s.l.], v. 166, p. 1-27, jan. 2018. Elsevier BV. <http://dx.doi.org/10.1016/j.cviu.2017.03.007>.