Escola Politécnica da Universidade de São Paulo

PMR2560 - Elementos de Robótica

Trabalho de Visão Computacional

Alvaro Henrique Chaim Correia
NUSP 7626932

Prof. Eduardo Lustosa Cabral

Jorge Luiz Moreira Silva
NUSP 7626776

1 Introdução

A proposta do trabalho é o desenvolvimento de um algoritmo de detecção de objetos baseado principalmente na coloração da imagem. Para isso escolhemos bananas como objeto alvo porque sua cor é relativamente uniforme e o seu formato alongado fornece um desafio interessante.

O código foi todo desenvolvido em MATLAB e pode ser encontrado no endereço `https://github.com/AlCorreia/Image_Processing` para reprodutibilidade.

Como referência, mostraremos as etapas de processamento na seguinte imagem que foi usada para desenvolver o algoritmo. Apesar de essa ter sido a figura usada para definir os paramêtros de detecção do modelo, outras imagens serão apresentadas ao longo do relatório para validar o algoritmo e ilustrar pontos interessantes.

Figura 1: Imagem usada para treinar o modelo a detectar bananas.

A partir da imagem na Figura 1, percebe-se dois desafios importantes que não podem ser resolvidos somente com a detecção de cores: (i) a presença de pintas frequentes em bananas maduras e (ii) a separação de bananas num mesmo cacho que se sobrepõem na imagem. Por isso, exploramos outros métodos, como processamento morfológico e detecção de bordas.

Esse relatório é organizado da seguinte forma. Nas secções de 2 à 8, cada uma das etapas de processamento do algoritmo é apresentada e discutida com referência aos resultados obtidos na imagem acima. Na secção 9, a sequência de imagens ilustrando o passo-a-passo para cada uma das imagens testadas. Finalmente, discutimos os resultados e apresentamos a listagem do código em MATLAB.

2 Detecção da Colaração Amarela em HSV

O primeiro passo é ler as imagens e convertê-las do formato RGB (Red Green Blue) para HSV (Hue, Saturation, Value). Isso pode se feito com os comandos *imread* e *reg2hsv* em MATLAB.

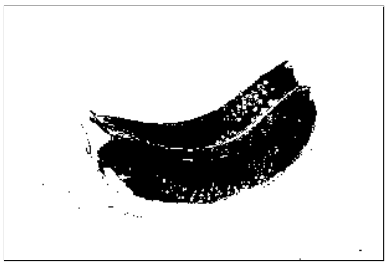
Para detectar regiões amarelas na imagem, foram criadas três máscaras, uma para cada componente HSV. Dessa forma, pode-se eliminar todas as partes da imagem que não estejam na banda de colaração amarela correspondente a bananas típicas.

$$\begin{aligned} 0.10 < Hue < 0.14 \\ 0.4 < Saturation < 1.0 \\ 0.6 < Value < 1.0 \end{aligned}$$

Com esses limites em HSV, pode-se produzir uma imagem binária que representa as regiões da

imagem que contém a cor amarela. De fato, como se pode observar na Figura 2, a máscara produzida corresponde à parte amarela das duas bananas na Figura 1.

(a) Imagem original.

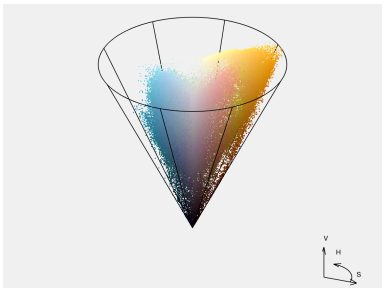


(b) Imagem binária com as regiões identificadas como sendo amarelas.

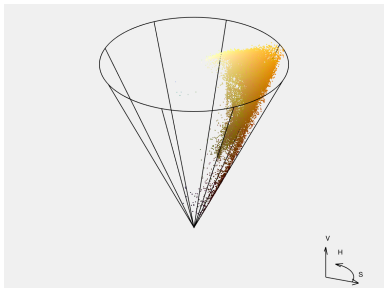
(c) Imagem filtrada por cor.

Figura 2: Passos de filtragem por cor simples.

Também é interessante visualizar a distribuição de cores na imagem original em comparação aos tons de amarelo preservados pela máscara. A Figura 3 mostra as cores presentes na Figura 2 (a) e (c) quando plotados num mapa HSV. Nota-se que, originalmente, a imagem era marcada por tons de azul e rosa que compunham o fundo, e que a máscara efetivamente preservou somente os tons de amarelo.



(a) Antes da filtragem por cor.



(b) Depois da filtragem por cor.

Figura 3: Cores presentes na imagem, antes e depois da filtragem por cor.

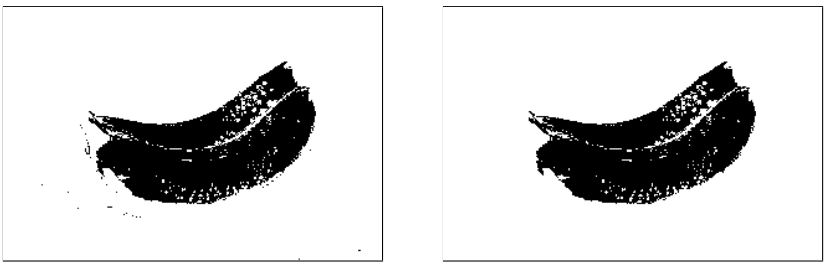
3 Processamento Morfológico

Apesar de a máscara conseguir filtrar as cores indesejadas, as regiões preservadas ainda não permitem identificar uma banana por completo. Como mostrado na Figura 2, somente a filtragem por cor não é suficiente para separar as duas bananas nem definir suas formas por conta de imperfeições causadas pela presença de outras cores no corpo da banana. Para mitigar esse problema, aplica-se

uma sequência de operações de processamento morfológico à máscara binária para eliminar essas imperfeições e recuperar regiões mais próximas do formato de uma banana.

3.1 Remoção de pequenos objetos

Como os objetos a serem detectados estão em primeiro plano, eles tendem a ocupar largas áreas da imagem. Por isso, pequenas regiões amarelas provavelmente não contém uma banana e podem ser descartadas. O limite usado para definir uma região suficientemente grande na Figura 1 foi de 1000 pixels.



(a) Antes (b) Depois

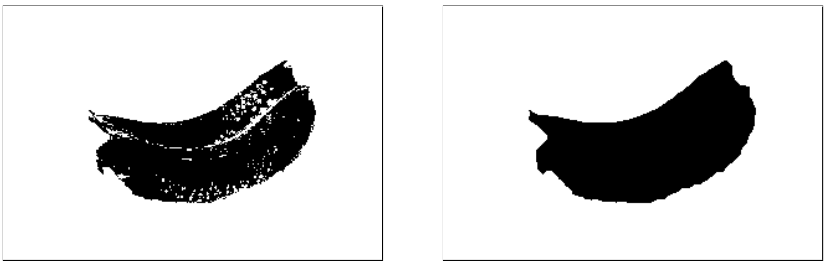
Figura 4: Remoção de pequenas regiões na máscara.

Na Figura 4, percebe-se que pequenos pontos foram removidos, preservando-se somente a parte central que corresponde de fato às bananas.

3.2 Crescimento de Região

Também é possível que o objeto de interesse apresente outros tons de cores que não o amarelo. Nesse case, a máscara produz uma região com buracos, o que dificulta a extração de informações interessantes, como a área ocupada pela banana.

Crescimento de regiões preenche partes da máscara que estejam cercadas por "1"s e assim elimina "buracos"como os causados pelas pintas da banana. Essa operação pode ser realizada com o comando *imclose* em MATLAB, que consiste na aplicação de dilação seguida de erosão.



(a) Antes (b) Depois

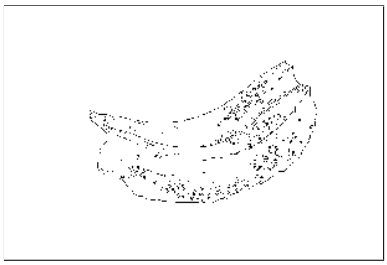
Figura 5: Crescimento de região na máscara.

Na Figura 5, observa-se o resultado da aplicação de *imclose* com um núcleo de tamanho de 11 pixels. Claramente, todos os buracos da máscara foram removidos, porém também se perde qualquer sinal de separação entre as bananas. Esse problema pode ser resolvido com métodos de detecção de bordas, como apresentado na secção seguinte.

4 Detecção de Bordas

Para separar múltiplas bananas numa mesma imagem, foi aplicado um algoritmo de deteção de bordas conhecido como *Sobel*. Essa operação pode ser usada em MATLAB através da função *edge*.

(a) Imagem após a aplicação da máscara definida na Figura 5 (b).



(b) Bordas das bananas detectadas pelo método *Sobel*.

Figura 6: Detecção de bordas.

A ideia consiste em sobrepor as bordas encontradas com o método *Sobel* e a máscara mostrada na Figura 5(b). Para tanto, o método foi aplicado sobre a imagem já filtrada, Figura 6(a), para evitar ruídos provenientes do fundo. Porém, como as bordas são sutis, a sensitividade do algoritmo deve ser relativamente baixa, da ordem de 0.01, e dessa forma as pintas são igualmente detectadas, gerando ruído. Antes de aplicar as bordas à máscara, é preciso então detectar e remover essas pintas da Figura 6(b), como explicado na secção seguinte.

Uma vez removidas as pintas, é preciso aplicar uma operação de *abertura*, que consiste de erosão seguida de dilação. Porém, nessa etapa, observou-se que o processo funciona melhor com núcleos de tamanhos diferentes e assim, a erosão aplicada teve tamanho de 11 pixels e a dilação, de 5 pixels. Esse procedimento de *abertura* é necessário para realçar as bordas e permitir a separação das bananas nos passos seguintes.

5 Detecção e Remoção de Pintas

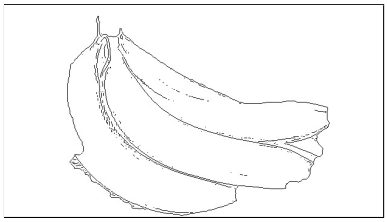
A presença de pintas escuras no corpo das bananas dificulta a detecção de bordas. Para efeito de comparação, são mostradas as bordas detectadas numa das imagens de teste na Figura 7. Com bananas sem pintas e completamente lisas, pode-se notar que as bordas detectadas são muito bem definidas e há pouco ruído no interior de cada fruta. A Figura 8 mostra que as pintas provocaram a detecção de várias bordas no interior de cada banana, por conta do alto contraste entre elas e o amarelo característico da fruta. Para remover este ruído e permitir uma melhor detecção de bananas com pintas na imagem, desenvolveu-se um algoritmo que removesse as bordas detectadas nas pintas.

O algoritmo consiste das duas etapas listadas abaixo:

1. Encontrar todos os BLOBs formados pelos pixels de borda.
2. Deletar BLOBs que não contenham linhas em seu interior através da verificação de algumas features de cada BLOB.

O primeiro passo é feito atavés da sequência de comandos *bwlabel*, para calcular os BLOBs, seguido de *regionprops* para calcular as propriedades geométricas de cada BLOB encontrado. Por fim, deleta-se os BLOBs que satisfazem pelo menos uma das condições abaixo:

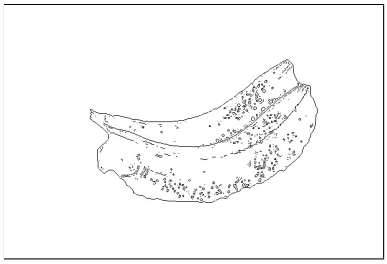
(a) Imagem Original.



(b) Imagem pré-processada com um filtro hsv, seguido de um detector de bordas sobel.

Figura 7: Detecção de bordas em bananas sem pintas.

(a) Imagem Original.



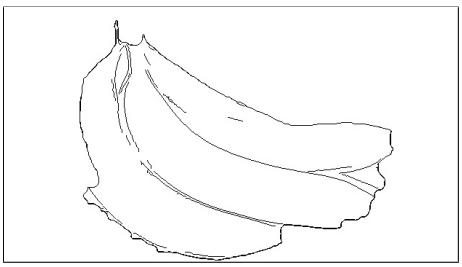
(b) Imagem pré-processada com um filtro hsv, seguido de um detector de bordas sobel.

Figura 8: Detecção de bordas em bananas com pintas.

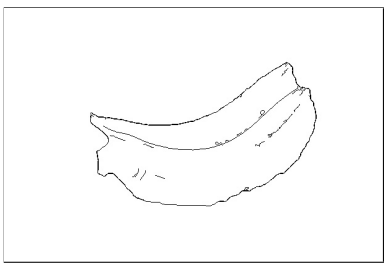
$$\frac{1 - \frac{1}{d_{min}}}{5,0} < \rho < 5.0\frac{1}{d_{min}}$$

$$d_{max} \leq 4$$

onde d_{min} e d_{max} são o comprimento da aresta menor e maior do retângulo que encobre o BLOB, respectivamente, e ρ é a densidade de pixels do BLOB em relação à área do mesmo retângulo.



(a) Banana sem pintas.



(b) Bananas com pintas.

Figura 9: Detecção de bordas após remoção de bordas geradas por eventuais pintas ou manchas nas bananas.

A inspiração destas features para detectar linhas originou-se da hipótese de que existem apenas três tipos de linhas: horizontal, vertical e diagonal. Enquanto as duas primeiras tem uma alta densidade de pixels, podendo chegar a 100%, a terceira é extremamente esparsa. Por este motivo,

adicionou-se os limites superiores e inferiores na primeira condição. O $5,0$ é um valor que foi tunado para este problema. Finalmente, a limitação da dimensão máxima foi definida, pois espera-se que BLOBs de linhas tenham pelo menos um de seus comprimentos longo, enquanto ruído gerado pelas pintas não.

Figure 9 mostra o resultado da aplicação do método apresentado nesta seção nas figuras 7 e 8. Em ambos os casos, pode-se verificar um resultado positivo na redução de ruídos no interior de cada fruta, enquanto suas bordas se mantiveram quase intocadas, como desejado.

6 Detecção de Blobs

Depois de definidas as bordas finais que serão usadas para separar as bananas, é possível detectá-las como BLOBs. Para isso, pode-se usar a função *regionprops* do MATLAB, que identifica cada um dos blobs e calcula propriedades interessantes como áreas e centróides.

(a) BLOBs identificados pelo método *regionprops*.

(b) BLOBs depois do crescimento de região.

Figura 10: Detecção de BLOBs.

Uma vez, definidos os blobs, crescimento de utilizado é aplicado novamente com *imclose*, mas dessa vez o processo é aplicado a cada BLOB individualmente. Isso permite corrigir eventuais rasgos e buracos residuais do processo de *abertura* das bordas sem unir os BLOBs já separados.

7 Ajuste de Densidade de Pixels

Para que o modelo pudesse ser aplicado a diferentes imagens é preciso corrigir alguns parâmetros com relação à densidade de pixels. Essa mudança é necessária porque as operações são todas feitas no nível de pixels e logo todas as propriedades geométricas dependem da resolução da imagem. Para adaptar os parâmetros a qualquer imagem, criou-se um coeficiente que varia com a raiz quadrada do número de pixels, como definido abaixo. A intuição,

$$coeff = \sqrt{\frac{num\_pixels}{250000}}$$

$$param = param \times coeff$$

onde num_pixels é o número de pixels na imagem a ser processada e $param$ representa um parâmetro qualquer do modelo. Os parâmetros em questão são o limite inferior para a área de cada região,

e o tamanho dos núcleos usados para dilação e erosão. O valor de $2,5.10^5$ foi escolhido porque todos os parâmetros foram definidos em relação a Figura 1, que tem $2,5.10^5$ pixels.

8 Propriedades Geométricas

As propriedades geométricas consideradas foram área, centróide e orientação. Enquanto as duas primeiras podem ser encontradas através da função *regionprops*, a orientação foi calculada com base na matriz de inércia dos pixels binários de cada BLOB através dos passos abaixo.

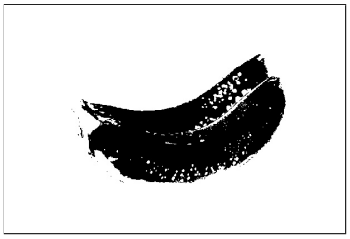
1. Cálculo das posições horizontal e vertical do centróide C_x e C_y.
2. Cálculo da diferença da posição de cada pixel contido no BLOB e da posição do centróide do respectivo BLOB: $C_x - p_{pixel,x}$ e $C_y - p_{pixel,y}$.
3. Cálculo de I_{xx}, I_{yy} e I_{xy}.
4. Cálculo dos autovalores e autovetores da matriz de inércia para achar as direções principais.
5. Plotar a direção do maior autovalor em azul e a menor em vermelho em cada banana.

Neste trabalho, essas características não foram utilizadas para auxiliar na detecção das bananas, mas apenas para caracterizar as que foram encontradas. Um possível aprimoramento do algoritmo apresentado neste trabalho seria descartar corpos nos quais as direções principais de inércia apresentam valores semelhantes, pois isso é incomoum em imagens de bananas por, em geral, apresentarem formas alongadas nas imagens.

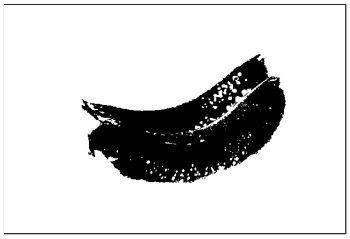
9 Processamento Passo a Passo

O processamento de cada imagem utilizada no desenvolvimento e validação do algoritmo deste trabalho é detalhado abaixo, com uma imagem para cada etapa. Em particular, são apresentados os resultados com a Figura 1 e duas outras imagens de teste.

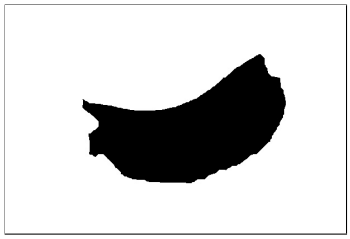
(a) Imagem original.



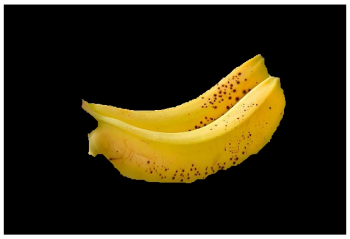
(b) Filtragem de cor.



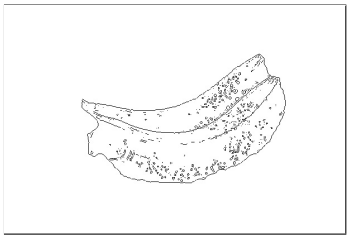
(c) Remoção de áreas pequenas.



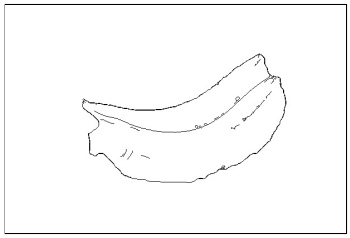
(d) Crescimento de região.



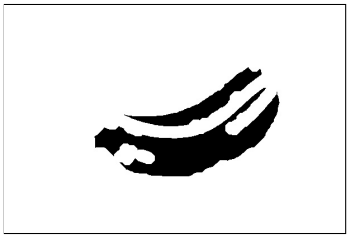
(e) Resultado da máscara.



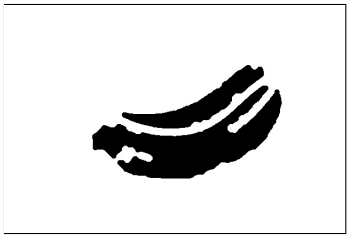
(f) Bordas de Sobel.



(g) Bordas sem pintas.



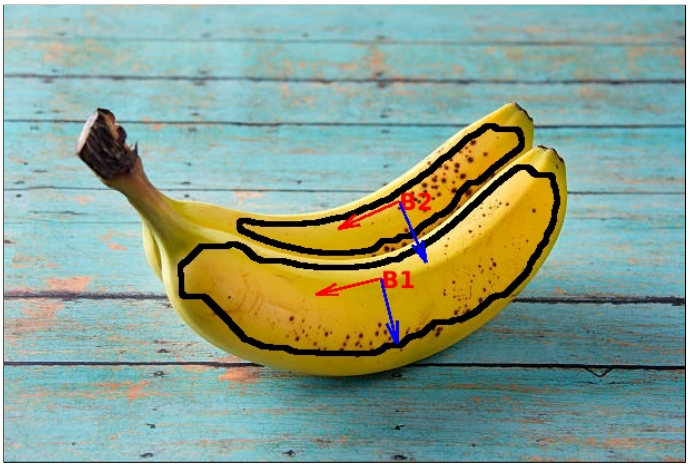
(h) Erosão.



(i) Dilação.

(j) Detecção de BLOBs

(k) Crescimento de região.

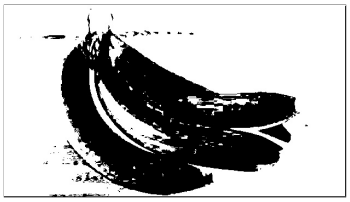


Banana (#)	Area (Pixels)
1	25840
2	8220

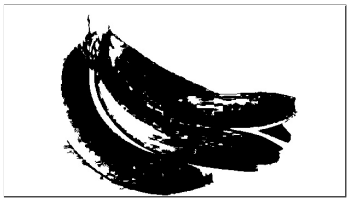
(l) Centroide e orientação.

Figura 11: Processamento passo-a-passo para a imagem de treino.

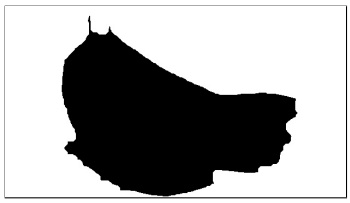
(a) Imagem original.



(b) Filtragem de cor.

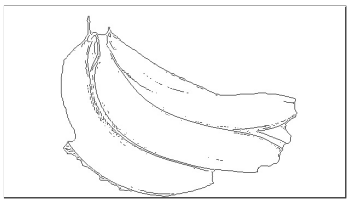


(c) Remoção de áreas pequenas.

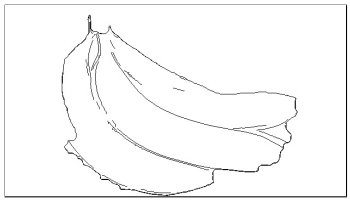


(d) Crescimento de região.

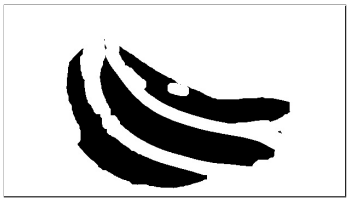
(e) Resultado da máscara.



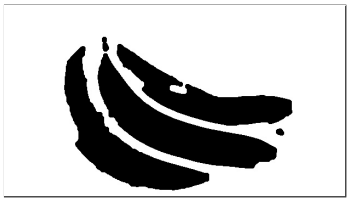
(f) Bordas de Sobel.



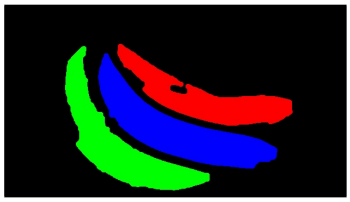
(g) Bordas sem pintas.



(h) Erosão.

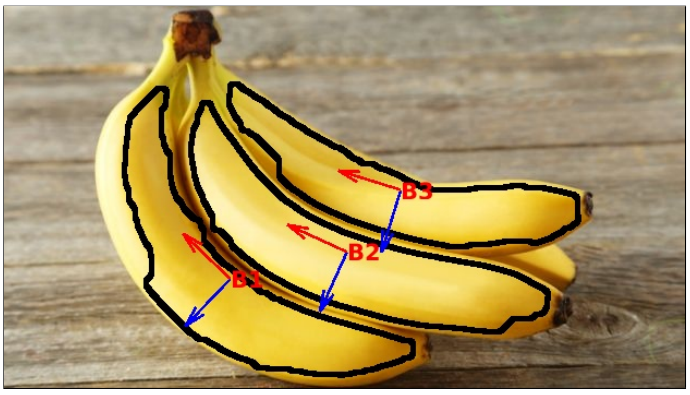


(i) Dilação.



(j) Detecção de BLOBs

(k) Crescimento de região.



(l) Centroide e orientação.

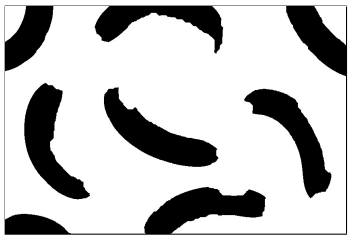
Banana (#)	Area (Pixels)
1	19327
2	22064
3	16142

Figura 12: Processamento passo-a-passo para a imagem de teste 1.

(a) Imagem original.

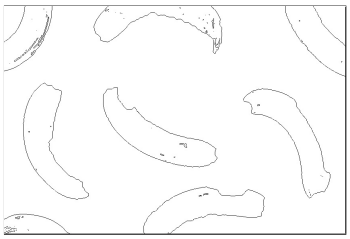
(b) Filtragem de cor.

(c) Remoção de áreas pequenas.

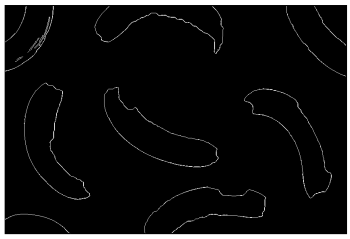


(d) Crescimento de região.

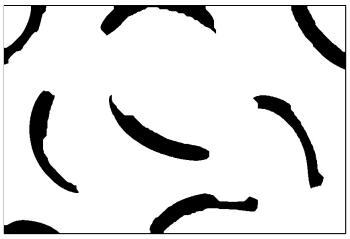
(e) Resultado da máscara.



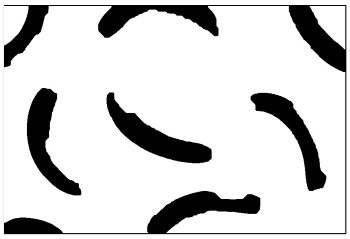
(f) Bordas de Sobel.



(g) Bordas sem pintas.



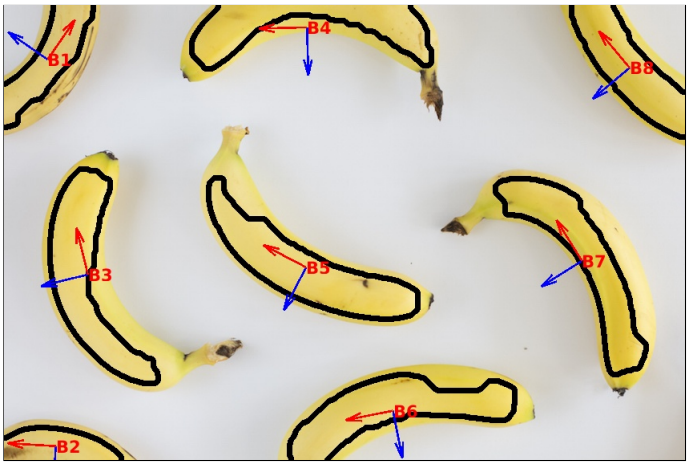
(h) Erosão.



(i) Dilação.

(j) Detecção de BLOBs

(k) Crescimento de região.



Banana (#)	Area (Pixels)
1	6596
2	4669
3	13649
4	10706
5	14968
6	13491
7	12777
8	9953

(l) Centroide e orientação.

Figura 13: Processamento passo-a-passo para a imagem de teste 2.

10 Discussão de Resultados

O método apresentado aqui mostrou-se capaz de reconhecer bananas nas várias imagens testadas, atingindo os objetivos do trabalho. As eventuais imperfeições nas formas atribuídas às bananas se dão majoritariamente por causa das limitações da detecção baseada cores em condições variadas de iluminação. Por exemplo, na Figura 12, a parte removida da banana mais à esquerda corresponde a região da fruta onde a luz incide com maior intensidade e que por isso é quase branca. Por outro lado, se valores mais baixos de saturação fossem permitidos, o algoritmo passaria a aceitar largas regiões do fundo da imagem, o que dificultaria os passos seguintes.

Outro fator relevante é o formato das bananas que não só varia de uma fruta para outra, mas também se projeta de formas diferentes dependendo do ângulo em que a foto foi tirada. Essa complicação impede que a forma da banana seja prevista de antemão em comparação à objetos mais simples como as esferas ou prismas sugeridos na proposta do trabalho. Uma possibilidade para mitigar esse problema seria utilizar o cálculo do momento de inércia também durante a fase detecção, pois ele permitiria verficar se a forma alongada é observada.

Código MATLAB

```
clear all
fontSize = 9;

% Read the image file as RGB
pic = imread('images/banana13.jpg');
% Calculate the number of pixels to correct erode and dilation kernel
    sizes
[num_pixels_1, num_pixels_2, xxx] = size(pic);
num_pixels = (num_pixels_1 * num_pixels_2);
coeff = sqrt(num_pixels/250000);

% Convert the image to hsv
pic_gray = rgb2gray(pic);
pic_hsv = rgb2hsv(pic);
h_pic = pic_hsv(:,:,1);
s_pic = pic_hsv(:,:,2);
v_pic = pic_hsv(:,:,3);

% Define the thresholds that define the color yellow
% Those are larger than what is strictily defined as yellow so as to not
% risk missing any pixel. A second mask is applied later on to remove
    any
% other colors.
hueThresholdLow = 0.10;
hueThresholdHigh = 0.14;
saturationThresholdLow = 0.4;
saturationThresholdHigh = 1;
valueThresholdLow = 0.6;
valueThresholdHigh = 1.0;

% Define the size of the smallest acceptable area.
% Smaller areas will be removed.
smallestAcceptableArea=round(1000*coeff);

% Define a mask for each hsv component with the thresholds.
hueMask = (h_pic >= hueThresholdLow) & (h_pic <= hueThresholdHigh);
saturationMask = (s_pic >= saturationThresholdLow) & (s_pic <=
    saturationThresholdHigh);
valueMask = (v_pic >= valueThresholdLow) & (v_pic <= valueThresholdHigh)
    ;
% The final mask is a the intersection of the three above
coloredObjectsMask = uint8(hueMask & saturationMask & valueMask);

% Remove small objects.
coloredObjectsMask = uint8(bwareaopen(coloredObjectsMask,
    smallestAcceptableArea));
subplot(3, 3, 1);
imshow(coloredObjectsMask, []);
caption = sprintf('Objects of %d Pixels Removed', smallestAcceptableArea
    );
title(caption, 'FontSize', fontSize);
fontSize = 13;
```

```
47
48 % Close holes with imclose() to obtain a smoother image.
49 structuringElement = strel('disk', round(11*coeff));
50 coloredObjectsMask = imclose(coloredObjectsMask, structuringElement);
51 subplot(3, 3, 2);
52 imshow(coloredObjectsMask, []);
53 title('Border smoothed', 'FontSize', fontSize);
54
55 % Fill in any holes, since they are also likely to be yellow.
56 coloredObjectsMask = imfill(logical(coloredObjectsMask), 'holes');
57 subplot(3, 3, 3);
58 imshow(coloredObjectsMask, []);
59 title('Regions Filled', 'FontSize', fontSize);
60
61 % Filter the original image in gray scale
62 filtered_image = uint8(double(pic_gray).*double(coloredObjectsMask));
63 % Apply Sobel border detection with low sensitivity
64 out_border = edge(coloredObjectsMask, 'sobel');
65 filtered_image = localcontrast(filtered_image);
66 subplot(3, 3, 4);
67 imshow(filtered_image, []);
68 title('First Filtering', 'FontSize', fontSize);
69
70 % Apply Sobel border detection with high sensitivity
71 sobel_mask = (edge(filtered_image, 'sobel', 0.04));
72 subplot(3, 3, 5);
73 imshow(sobel_mask, []);
74 title('Sobel Mask', 'FontSize', fontSize);
75
76 % Detect and Label the blobs
77 [labeledImage, numberOfBlobs] = bwlabel(sobel_mask, 8);
78 blobMeasurements = regionprops(labeledImage, filtered_image);
79 delete = labeledImage*0;
80
81 % Remove small dots
82 for i =1:numel(blobMeasurements)
83     square = blobMeasurements(i).BoundingBox;
84     ratio = blobMeasurements(i).Area/(square(3)*square(4));
85     max_dim = max(square(3:4));
86     min_dim = min(square(3:4));
87     cte=5;
88     if max_dim<=4 || (ratio<(1/min_dim)*cte&& ratio>(1-1/min_dim)/cte)
89     delete = delete+(labeledImage==i);
90     end
91 end
92
93 % Redefine the borders by summing
94 sobel_mask = max(sobel_mask-delete,0);
95 sobel_mask = imcomplement(min(sobel_mask+out_border,1));
96
97 subplot(3, 3, 6);
98 imshow(delete)
99 title('Pixels to delete')
100
```

```
% Calculate the intersection between the masks and the sobel
coloredObjectsMask = uint8(coloredObjectsMask & sobel_mask);

% Apply erosion
structuringElement = strel('disk', round(11*coeff));
coloredObjectsMask = imerode(coloredObjectsMask, structuringElement);
subplot(3, 3, 7);
imshow(coloredObjectsMask, []);
title('After Erode', 'FontSize', fontSize);

% Apply dilation
structuringElement = strel('disk', round(5*coeff));
coloredObjectsMask = imdilate(coloredObjectsMask, structuringElement);
subplot(3, 3, 8);
imshow(coloredObjectsMask, []);
title('After Dilate', 'FontSize', fontSize);

% Filter small regions
coloredObjectsMask = uint8(bwareaopen(coloredObjectsMask,
    smallestAcceptableArea));

% Apply Blobify to define the final blobs and extract properties
[meanHSV, areas, numberOfBlobs, labeledImage] = Blobify(
    coloredObjectsMask, h_pic, s_pic, v_pic, coeff);
final_figure=figure;
labeledImage2 = labeledImage>0;
edges_new = imdilate(edge(labeledImage2,'sobel'), strel('disk', round(3*
    coeff)))*255;
filt_m = pic;
filt_m(:,:,1) = uint8(max(double(pic(:,:,1))-double(edges_new),0));
filt_m(:,:,2) = uint8(max(double(pic(:,:,2))-double(edges_new),0));
filt_m(:,:,3) = uint8(max(double(pic(:,:,3))-double(edges_new),0));
imshow(filt_m)
properties_reg = regionprops(labeledImage);
sizefig = size(pic);
index_lin = repmat([1:sizefig(1)]',[1,sizefig(2)]);
index_col = repmat([1:sizefig(2)],[sizefig(1),1]);

for i =1:numberOfBlobs
    % Add the name of the centroid
    txt1 = '\leftarrow sin(\pi) = 0';
    Centroid = properties_reg(i).Centroid;
    tx = text(Centroid(1),Centroid(2),['B',int2str(i)]);
    tx.FontSize = 16;
    tx.FontWeight = 'bold';
    tx.Color = [1.0,0,0];
    hold on
    % Select blob i and calculate its moment inertia to extract the
    % principal directions
    fig_blob = labeledImage==i;
    mass = sum(sum(fig_blob));
    fig_blob_lin = sum(sum((fig_blob.*(index_lin-Centroid(2))).^2));
    fig_blob_col = sum(sum((fig_blob.*(index_col-Centroid(1))).^2));
    fig_blob_lincol = -sum(sum((fig_blob.*(index_lin-Centroid(2))).*(
```

```
        fig_blob.*(index_col-Centroid(1)))));
152     mat = [fig_blob_lin, fig_blob_lincol; fig_blob_lincol, fig_blob_col
        ];
153     [a,b] = eig(mat);
154     % Plot the principal directions on each blob
155     quiver(Centroid(1),Centroid(2),a(1,2)*60,a(2,2)*60,0,'lineWidth',2,'
        Color','b','MaxHeadSize',14)
156     quiver(Centroid(1),Centroid(2),a(1,1)*60,a(2,1)*60,0,'lineWidth',2,'
        Color','r','MaxHeadSize',14)
157 end
158
159 function [meanHSV, areas, numberOfBlobs, labeledImage] = Blobify(
    maskImage, hImage, sImage, vImage, coeff)
160 try
161     % Label each blob to allow for further measurements
162         [labeledImage, numberOfBlobs] = bwlabel(maskImage, 8);
163
164         if numberOfBlobs == 0
165                 % Didn't detect anything on this image.
166                 meanHSV = [0 0 0];
167                 areas = 0;
168                 return;
169     end
170
171     % Apply imclose to each blob individually.
172     matrizref = 0*labeledImage;
173     for i=1:numberOfBlobs
174         structuringElement = strel('disk', round(21*coeff));
175         matrizref = matrizref + imclose(labeledImage==i,
            structuringElement);
176     end
177
178     % Recalculate the blobs.
179     [labeledImage, numberOfBlobs] = bwlabel(matrizref, 8);
180
181     % Get all the blob properties.
182         blobMeasurementsHue = regionprops(labeledImage, hImage, 'area',
            'MeanIntensity');
183         blobMeasurementsSat = regionprops(labeledImage, sImage, 'area',
            'MeanIntensity');
184         blobMeasurementsValue = regionprops(labeledImage, vImage, 'area'
            , 'MeanIntensity');
185
186     % Assign the areas.
187     % One row for each blob. One column for each color.
188         areas = zeros(numberOfBlobs, 3);
189         areas(:,1) = [blobMeasurementsHue.Area]';
190         areas(:,2) = [blobMeasurementsSat.Area]';
191         areas(:,3) = [blobMeasurementsValue.Area]';
192
193     % Assign mean hsv values
194     meanHSV = zeros(numberOfBlobs, 3);
195         meanHSV(:,1) = [blobMeasurementsHue.MeanIntensity]';
196         meanHSV(:,2) = [blobMeasurementsSat.MeanIntensity]';
```

```
197         meanHSV(:,3) = [blobMeasurementsValue.MeanIntensity]';
198
199     % Assign a different color to each blob and plot them all
200     coloredLabels = label2rgb(labeledImage, 'hsv', 'k', 'shuffle');
201     subplot(3, 3, 9);
202     imshow(coloredLabels, []);
203         title('Final Blobs', 'FontSize', 13);
204
205 catch ME
206         errorMessage = sprintf('Error in function %s() at line %d.\n\
            nError Message:\n%s', ...
207                 ME.stack(1).name, ME.stack(1).line, ME.message);
208         fprintf(1, '%s\n', errorMessage);
209         uiwait(warndlg(errorMessage));
210 end
211 return; % from Blobify()
212 end
```