Andrew Kozempel

Professor Avanzato

CMPSC 497

8 September 2023

**Lab #1B: Cookie Defect Detector**

**Problem Statement:**

The objective is to design and implement a computer vision algorithm to detect cracker defects in a given image using MATLAB. The algorithm is aiming to analyze images of crackers to identify and detect whether they are “good” or “bad.”

**MATLAB Script:**

| **%{ Andrew Kozempel CMPSC 497 Fall 2023 LAB #1B Cookie Defect Detector %}  % Initialize running totals total\_crackers = 0; total\_good = 0; total\_bad = 0;  %images = {'cracker10.png'}; images = {'cracker1.png', 'cracker2.png', 'cracker3.png', 'cracker4.png', 'cracker5.png', 'cracker6.png', 'cracker7.png', 'cracker8.png', 'cracker9.png', 'cracker10.png'};   for i = 1:length(images)   % Access original RGB image  RGB = imread(images{i});  figure, imshow(RGB);    % Create grayscale image  gray = rgb2gray(RGB);  threshold = graythresh(gray);  %figure, imshow(gray);    % Create bw (binary image)  bw = im2bw(gray, threshold);  %figure, imshow(bw);    % Remove all object containing fewer than 30 pixels  bw1 = bwareaopen(bw, 30);  %figure, imshow(bw1);    % Fill a gap in the pen's cap  se = strel('disk', 2); % Structuring element; "paintbrush"  bw2 = imclose(bw1, se); % We will cover later in course  %figure, imshow(bw2);    % Fill any holes, so that regionprops can be used to estimate the area   % enclosed by each of the boundaries  bw3 = imfill(bw2, 'holes');  %figure, imshow(bw3);    % Get pixels for boundaries of each object  [B,L] = bwboundaries(bw3, 'noholes');    % Display the label matrix and draw each boundary  figure, imshow(label2rgb(L, @jet, [.5 .5 .5]));    % Allow graphics to be added to same plot  hold on;    % length(B) is number of objects  for k = 1:length(B)  % B is "cell" data type (set)  boundary = B{k};  plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2);  end    % Find area(in pixels) and centroid (x, y) for each object in label matrix  stats = regionprops(L, 'Area', 'Centroid');     % Arbitrary value (change as needed)  threshold = 0.75;       fprintf('\nCracker %d:', i);  % Loop over the boundaries (each object has a boundary)  for k = 1:length(B)    % Obtain (X,Y) boundary coordinates corresponding to label 'k'  boundary = B{k};    % Compute a simple estimate of the object's perimeter    % Find 2-col. array of (x2-x1)^2 and (y2-y1)^2  delta\_sq = diff(boundary).^2;     % Sum (row) and take sqrt to find dist.  % Then sum all distances to find perimeter  perimeter = sum(sqrt(sum(delta\_sq,2)));     area1 = perimeter^2/(4\*pi);    % Obtain the area calculation corresponding to label 'k'  area2 = stats(k).Area; % So, we calculated the area 2 different ways    % Compute the roundness metric (compare 2 methods)  metric = area2/area1; % Circular objects have metric close to 1    % Display the results  metric\_string = sprintf('%2.2f', metric);     % Mark objects above the threshold with a small black circle in the center of the object  if metric > threshold  centroid = stats(k).Centroid;  plot(centroid(1), centroid(2), 'ko');    % Increment the round object count  total\_good = total\_good + 1;  % Print test case  fprintf(' GOOD (%0.2f)', metric);   else   % Increment the round object count  total\_bad = total\_bad + 1;  % Print test case  fprintf(' BAD (%0.2f)', metric);   end    text(boundary(1,2)-35, boundary(1,1)+13, metric\_string, 'Color', 'y', 'FontSize', 14, 'FontWeight', 'bold');  end    total\_crackers = total\_crackers + 1; end  % Print total number of crackers fprintf('\n\nTotal Number of crackers: %d\n', total\_crackers);  %Print total number of good crackers fprintf('Total Number of good crackers: %d\n', total\_good);  %Print total number of bad crackers fprintf('Total Number of bad crackers: %d\n', total\_crackers-total\_good);** |
| --- |

**Results:**

| **Test Case** | **Starting Image** | **Final Image** |
| --- | --- | --- |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |
| **7** |  |  |
| **8** |  |  |
| **9** |  |  |
| **10** |  |  |

| **MATLAB OUTPUT** | |
| --- | --- |
| The output identifies:   1. the cracker number 2. states whether it is good or bad 3. the roundness metric.   The totals are then shown at the end. | |
| **Copy and Paste** | **Screenshot** |
| >> Lab\_1B  Cracker 1: GOOD (0.82)  Cracker 2: GOOD (0.81)  Cracker 3: GOOD (0.76)  Cracker 4: GOOD (0.76)  Cracker 5: GOOD (0.77)  Cracker 6: BAD (0.62)  Cracker 7: BAD (0.68)  Cracker 8: BAD (0.59)  Cracker 9: BAD (0.60)  Cracker 10: BAD (0.51) BAD (0.61) BAD (0.62)  Total Number of crackers: 10  Total Number of good crackers: 5  Total Number of bad crackers: 5 |  |

**Conclusion:**

Overall, it was a pretty straightforward lab, but I enjoyed doing it. I was successfully able to design and implement an algorithm to identify which crackers were good and which were bad. The first five crackers were good and the last five were bad, which were correctly identified (see result).

Just a few things came up during the implementation of this MATLAB script. First, I feel like the roundness metric for the good crackers are a bit low, but I lowered the threshold to make sure the crackers were correctly categorized. Of the five good crackers, the range was 0.76 to 0.82, so within 0.05 of each other. The bad ones were in the range of 0.51 to 0.62. The threshold was 0.75, so the bad ones were not very close. I could perhaps lower it to 0.7, but 0.75 felt pretty safe.

The second thing was the last cracker image. I wanted to make sure it identified a cracker that had multiple pieces. It messed up the original “bad cracker counter,” so I just subtracted the good cracker count from the total cracker count, which obviously gives the bad cracker count. The last image counted as 3 different crackers, even though it was just one split in multiple pieces, so that is why the change was made. Overall, the change corrected the issue.