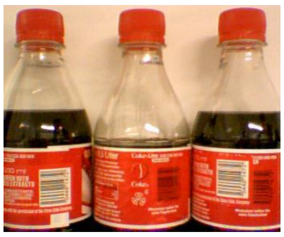
Embedded Image Processing – Assignment 1 – Factory bottles defect detection – Report

Name: Amogh Agnihotri
Student ID: 21236437
Course: 1MAI1

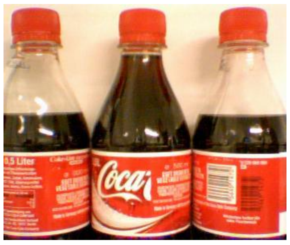
A. Problem Statement

We are given a various set of images which has 3 bottles in it. The images are of the bottles of Coca-Cola company which may are may not have some faults. There are basically seven types of faults and a case where the bottle is deformed. In picture, there are three bottles but we have to focus explicitly on the middle bottle and perform fault detection on it. Rest of the image can be ignored.
The faults in the bottles are shown below

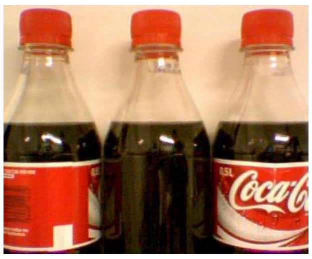
1. Underfilled Bottles



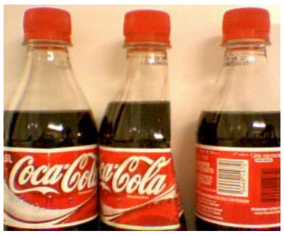
2. Overfilled Bottles



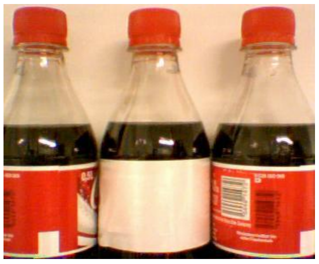
3. Bottles with missing labels



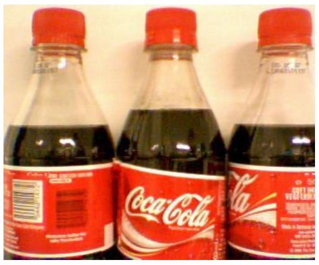
7. Deformed Bottles



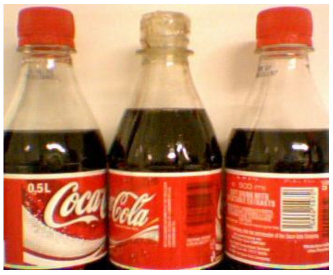
4. Bottles with no label prints



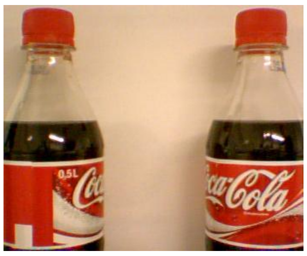
5. Label is not straight



6. Bottle with missing cap



8. Bottle not present



The normal bottle is the one where there are no faults and no output or fault is given for it. **The solution has to be as such that all the faults in the given bottles needs to be reported, for example, a bottle might be overfilled and label is missing, both the faults need reporting.**

B. System Design for the solution

The general algorithm followed to check the faults is as below –

1. **Define a variable which holds the folder name of all the images.**
2. **Check if that folder exists, otherwise, throw error with valid error message**
3. **Store the image of the normal bottle (gold standard) in a global variable to use it for comparison of each fault later on.**
4. **Crop the area of the image where we want to find the fault, for example, if need to find the label related faults, just crop the area of the label of middle bottle.**
5. **Convert it to grayscale**
6. **Calculate distance of each fault cropped from the gold standard.**
7. **Run functions in order as below:**
   a. **Bottle missing**
   b. **Bottle deformed**
   c. **Overfill or Underfill**
   d. **Label missing, if not, run label printing, else run label positioning and printing issue detection**
   e. **Test for cap missing**
8. **Compare the distance with original image and predefined threshold for the fault.**
9. **Print all the faults that are present.**

C. Design Decisions and Applied Algorithm

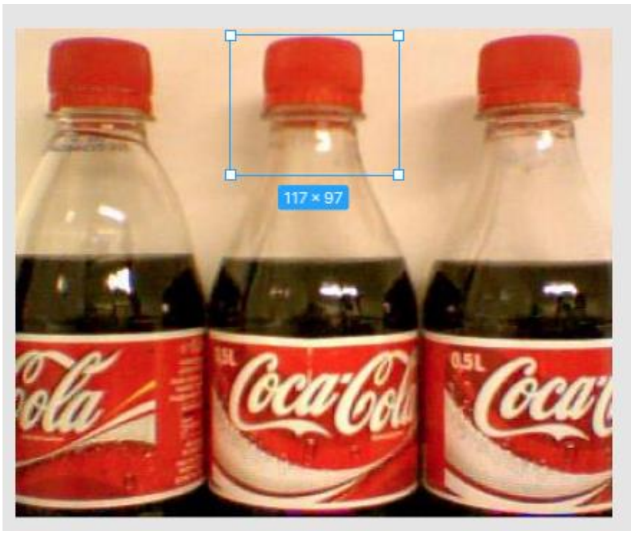
1. **Initially crop the incoming image to the 'area of interest', where we want to find the fault. This is explained case by case later.**
2. **Convert the cropped image to grayscale.**
3. **Do the same for Gold standard or image with no faults**
4. **Now, convert both images to histogram with the help of *imhist(image_name)* function and store the vectorized value of histogram in a variable. The code snippet for this is as below:**
   ```
   [c1, n] = imhist(normal_grayscale); --- 'n' is used to store dimensions, which I
   wanted to check
   ```
5. **Normalize both the vectors as per the image in order to perform calculations on them with formula as below:**
   ```
   c1 = c1/size(normal_grayscale, 1) / size(normal_grayscale, 2); -- Image name is
   normal_grayscale here as dummy variable as placeholder to show the formula.
   ```

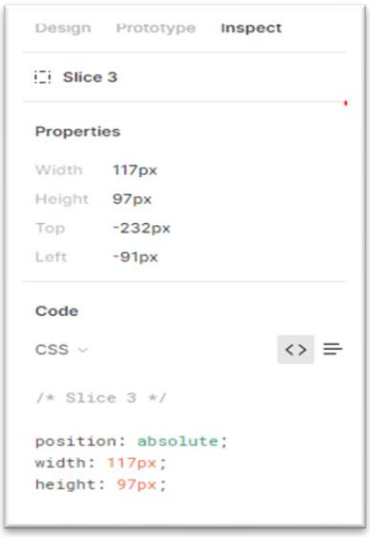
6. As not we have normalized both the histograms according to their images, find the Euclidean distance between them with the help of pdist2(c1,c2) function. Which will output the Euclidian distance for all the dimensions of both the images giving us the difference between two images.
7. Find the mean of all dimensions and we get the single numerical value between the images, which we store in a variable and consider it as a threshold for identifying the particular fault.
8. Run this on all types of faults to be recognized and find the threshold for all the categories. These thresholds are listed separately later for all the categories.
9. Run for all the new images to find faults

D. Finding the area of interest

I used Figma tool to calculate dimensions of area of interest as below –

Cap Missing –





Slice 3

Properties

Width 117px
Height 97px
Top -232px
Left -91px

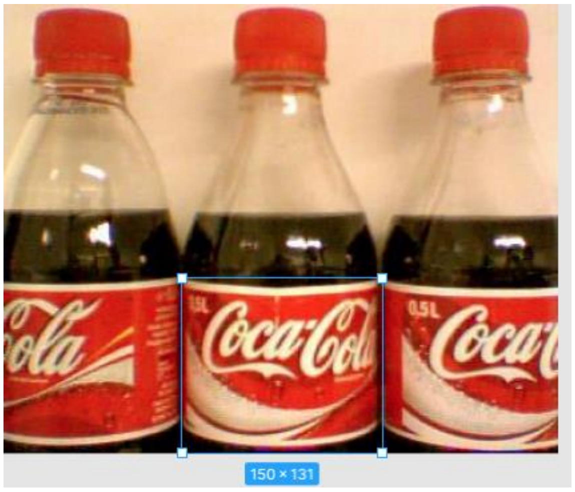
Code

CSS

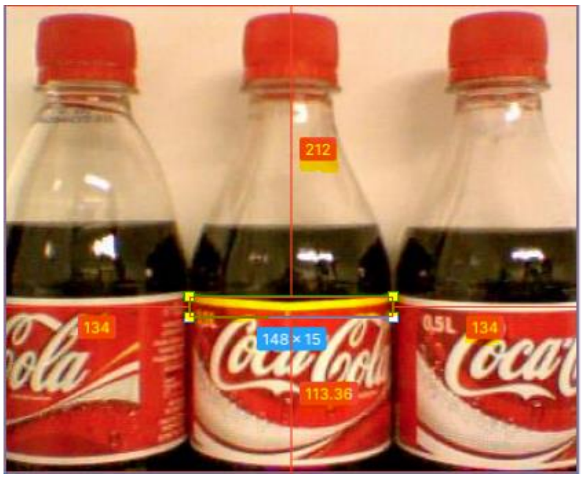
```
/* Slice 3 */

position: absolute;
width: 117px;
height: 97px;
```

This gives us the length of the area of interest and the coordinates which can be given as parameter to imcrop() function – for example, `imcrop(Normal_img,[111 177 135 255])`

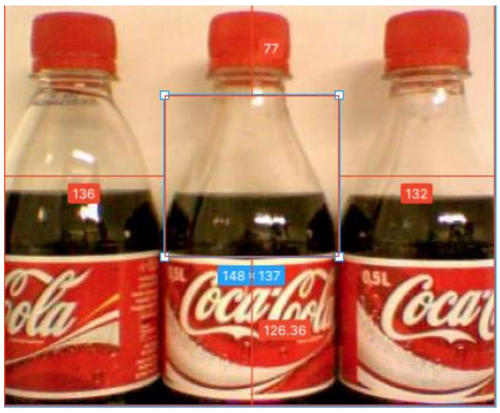
Label – Present / Not Printed



Label – Incorrect positioning



Overfill or Underfill –



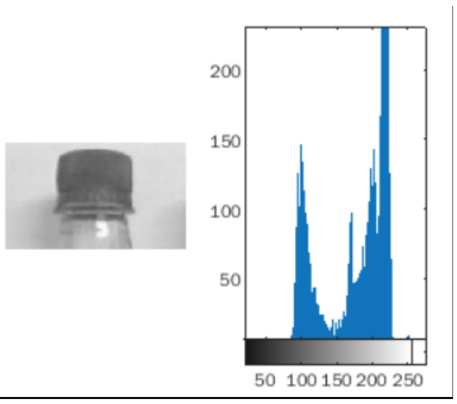
Here you can clearly see, positions from all the sides of the picture. Similar is the case for rest.

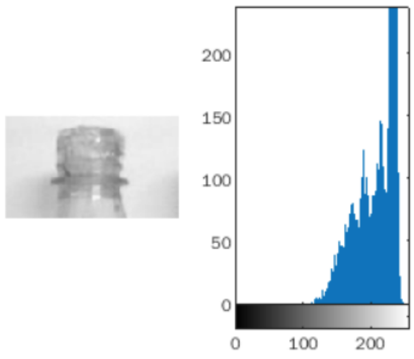
Normal Bottle

E. <u>Histograms Plotting and Comparison</u>

Here are few examples of histograms for comparisons of few scenarios, where histogram of normal bottle is compared with bottle with defects.
We can clearly see the difference in the b/w values of the images with the help of histogram
We take advantage of the difference to differentiate with the help of pixel values stores in form of vector as histogram and calculate Euclidean distance between two images.

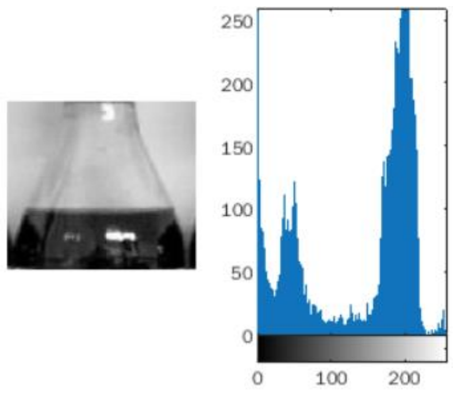
1. **No cap:**
   Normal Bottle – **Bottle with missing cap –**



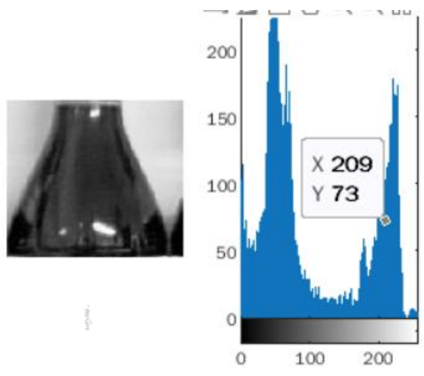


2. **Overfilling and Underfilling**

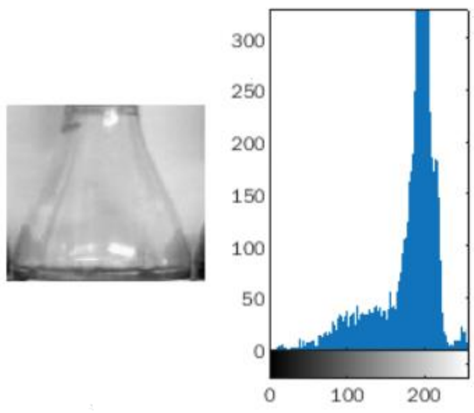
Normal Bottle



Overfilled bottle

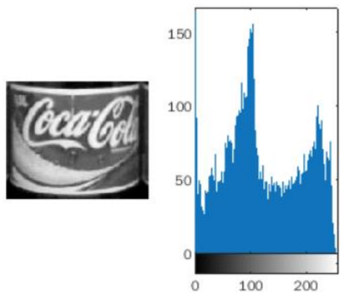


Underfilled bottle

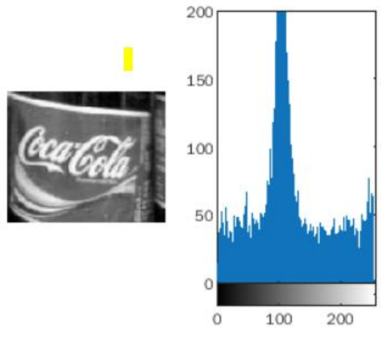


3. **Labels –**

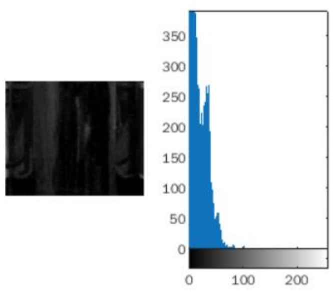
Normal bottle –



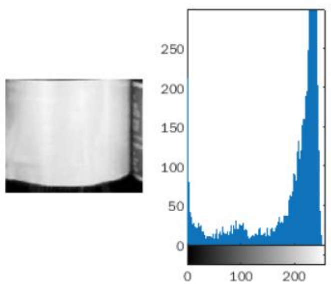
Label incorrect –



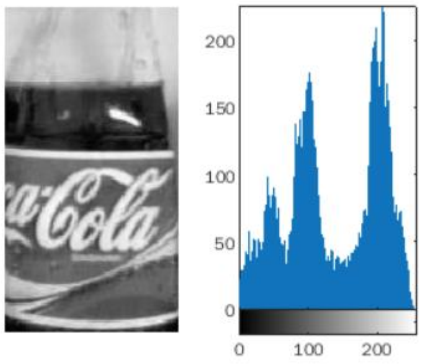
No Label –

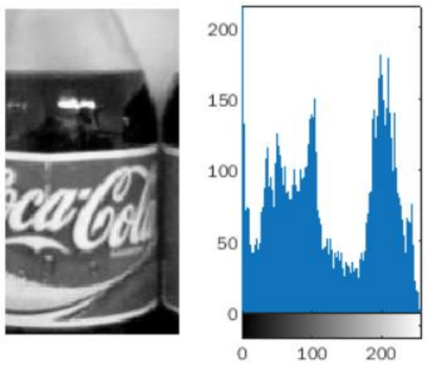


Blank Label –



4. **Deformed Bottle –** **Normal bottle –**





F. Thresholds defined for each fault

The threshold was calculated by training on provided images and then executed on new data to find faults.
The threshold is basically the Euclidean distance from Normal bottle image and the bottle we are giving input as prospective bottle with faults.

Below are category wise thresholds –
Please note, the thresholds depend on particular cropped area of the fault, hence reducing false positives

1. Missing Cap **– distance > 0.0058**
2. Overfilled **– (0.0035<distance) && (distance<0.0045)**
3. Underfilled **– (0.0048<distance) && (distance<0.0053)**
4. No Label **– (0.0055<distance) && (distance<0.0062)**
5. Print on the Label is missing **– (0.0040<distance) && (distance<0.0045)**
6. Label printed incorrectly **— (0.0020<distance) && (distance<0.0030)**
7. Deformed bottle **– (0.0023<distance) && (distance<0.0025)**
8. Missing bottle **– distance > 0.0052**

G. Accuracy tests on the data

To run the program place it in same folder which contains all the test images along with 1 normal image which has no faults. It will run tests on all the images present from given folder path and display faults if there are any. If there are no faults, **there is no explicit message (To print any message we can just add one else, but considering industry scenario, if there is no fault, there is no explicit signaling and bottle just passes from the belt).**
The program will first check if the bottle is present, if it is present, it will move to find more faults in it otherwise, if the bottle is missing, the appropriate message will be displayed and it will move to next sample.

All the tests are done manually by me and results are reported as observed. If all the faults are correctly detected then it is considered as correct detection, otherwise it is detected as either a False Positive or False Negative.

Below is the results of Fault detection by individual category –

1. Cap Missing – 100% (10/10 detection)
2. Overfilled – 100% (10/10 detection)
3. Underfilled – 100% (10/10 detections)
4. Label Missing – 100% (10/10 detections)
5. Label Print missing – 100% (10/10 detections)
6. Label pasted incorrectly – 80% (8/10 detections)
7. Deformed bottles – 70% (7/10 detections)
8. Missing bottles – 100 % -- all the missing bottles were ignored

Performance metrics –

Total Images – 141
117 images detected perfectly
12 images detected as False Positives
12 images detected as False negatives

H. Challenges Observed:

1. The biggest challenge that I faced, was to find out deformed bottles. It is difficult to narrow down on area of interest for deformed bottles. I tried with various areas but the approach I have used is not really the best for such classifications. Even though, I was able to get almost all the true positives and reduce false positives to good extent.
2. The second issue I faced was to check if label is straight or not. It was difficult to classify from the same dimensions used for other label issues. I then went on to crop a small area of white line on label and narrow down a threshold based on the images.
3. The challenge was also to figure out the way of converting multidimensional matrix of histogram vectors to a numerical threshold, which I did overcome with the code given in matlab file.

Overall accuracy: ~85%

Future Scope:

1. The deformed bottles are not detected as good, this might be because it is very hard to differentiate based on the distance of histograms. Also, the cropping of the image for histograms can be at both sides of the bottle and the distance which it gives from the original image is very less as compared to the deformed bottle. This could be done better by combining the results of overfilling and label misprints which are the most probable results of bottle deformation
2. The program can be made better with helper functions which will help it run quickly even if the data is more. This requires more functions to write in order to get more parts of the bottle. So bottle can be divided in multiple parts and then detection can be given with multiple combinations.
3. Use of a shallow CNN to classify the faults.