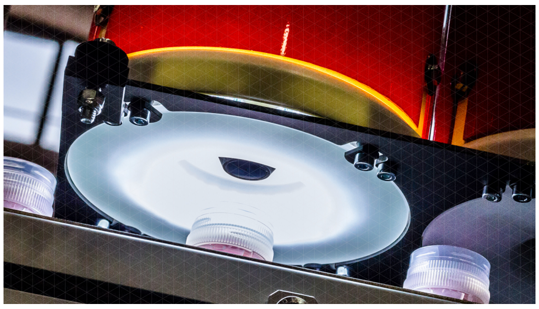
Alma Mater Studiorum · University of Bologna

Image Processing and Computer Vision - Artificial Intelligence

SACMI Projects 1+2+3 - Caps inspection



Lorenzo Cellini (lorenzo.cellini3@studio.unibo.it)

October 2, 2022

Contents

1 Summary

In the last decades, the use of digital image processing and computer vision techniques in industrial applications has gained increasing attention and importance due to the development of increasingly high-performance and better quality hardware.

In this work the three industrial problems proposed by SACMI, an Imola based machinery manufacturing company, have been addressed. All the three projects consist in the application of image processing and computer vision techniques in the field of plastic cap inspection. The followed approach is fully model based.

The software developed can be imagined as employed on a production line in order to detect defective products and prevents them to be sold as good ones.

The given set of groud thruth images contains both good and defective samples, along with defect description (only for the first project).

- The first problem consists in *plastic cap liner inspection*: an image processing & computer vision pipeline analyzes each image, classifies it based on some computed feature and outline detected defects.
- The second problem is about *cavity number preprocessing*: under given assumptions (eg. the position of cavity number with respect to the tab of cap) an image processing & computer vision pipeline takes as input the given images, finds the cavity number, rotates it in vertical position and corrects distortion due to the curvature of the cap. The output image contains the crop of the rectified cavity number, ready to be processed by an OCR tool.
- The third problem consists in detecting the *off-center of a decoration* in an aluminium cap: the image processing & computer vision pipeline reads an image, segments it based on some user defined colors and compute the Euclidean distance between the cap and its decoration centers.

Being this an industrial application I can safely assume to be in a *fully controlled environment*: this means that differences in lights, perspective, noise and colors are small between samples, and so algorithms can be more fine-tuned and specialized on the given image set.

2 Background theory

In this section theory behind the applied techniques is briefly presented.

2.1 Hough Transform

As stated before, each of the three problems involves at least one step where a circumference must be detected. To this end, *Circle Hough Transform* (CHT), a specialization of Hough transform, is employed. The Hough transform is a feature extraction technique that aims to find imperfect instances of objects, within a certain class of shapes, by a voting procedure. This voting procedure is carried out in a parameter space, called "Hough Space", from which object candidates are obtained as local maxima in a so-called "Accumulator Space" (a.k.a. "Accumulator Array" or "Quantized Hough Space").

The Hough space for a 2D circumference expressed in the form $(x - a)^2 + (y - b)^2 = r^2$ is the 3D space spanned by (a, b, r), representing a cone with apex in (a, b). The CHT algorithm counts the intersections among all the possible cones in the parameter space and outputs the set of parameter corresponding to the intersection point that accumulates more votes or the ones that surpass some user defined threshold.

Besides CHT, I have also exploited *Linear Hough Transform* (LHT) in solving problem 1, in order to highlight straight defects. By writing a line in the parametric form $r = x\cos(\theta) + y\sin(\theta)$ (for optimization reasons), LHT searches for intersections of lines in the parameter space (r, θ) and returns the parameters corresponding to points where more lines intersect.

In general HT is applied to an image after it went through denoising, binarization and a Canny edge detector. All of these steps are handled by OpenCV CHT and LHT implementations.

2.2 Filtering

Sometimes images are not ready to be processed by algorithms, due to the presence of noise that would lower algorithm performance in terms of quality of results. This problem is addressed by convolving the image with a filter that changes image pixels value depending on the chosen filter kernel. In this work have been employed Gaussian, median and bilateral filtering. In particular I used the latter when I needed to denoise but preserving edges, while the former two were used to smooth uniform regions before applying Canny edge detector (eg. in liner defect detection).

2.3 Canny edge detection

Canny Edge Detection is a popular multi stage edge detection algorithm, developed by John F. Canny in 1986. As shown in 2.1, Canny edge detector applies first a Gaussian denoising, then applies two Sobel operators, an horizontal and a vertical one, in order to compute image gradient and from these it computes gradient magnitude and direction. At this point, a full scan of the image is performed, in order to remove any unwanted pixel which may not constitute an edge (NMS step). The result is a binary image with thin edges. At the end, two thresholds are applied: the lower one discards candidates, while the higher one distinguish between what is for sure an edge and what is an edge because of being connected to a sure edge pixel. The final output is a binary image representing strong edges in the input image. Both the two thresholds must be tuned by experimental trials.

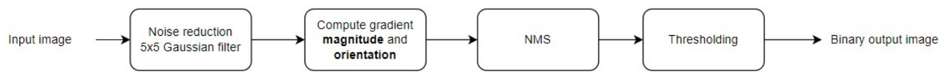

Input image
Noise reduction 5x5 Gaussian filter
Compute gradient **magnitude** and **orientation**
NMS
Thresholding
Binary output image

Figure 2.1: Canny edge detection algorithm (OpenCV implementation).

2.4 Image segmentation

In image processing and computer vision, image segmentation is the process of partitioning an image into multiple image segments, also known as image regions or image objects (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. It is typically used to locate objects and boundaries in images.

Image segmentation can be applied to both grayscale and color images, and in the latter, color information can be exploited.

In the literature there is a wide variety of segmentation techniques, relying on both classical digital image processing and deep learning.

In this work both grayscale segmentation and color segmentation have been employed: the former before applying morphology transformation in solving problem 2, while the latter in segmenting different objects based on their color in project 3.

2.4.1 Binarization

Sometimes certain tasks, e.g. object segmentation, requires a preliminary binarization step where the input grayscale image is converted to black-and-white, essentially reducing the information contained within the image from 256 shades of gray to 2. It is a form or segmentation, whereby an image is divided into its constituent objects. This technique consists in finding a threshold value in the greylevel/color histogram that effectively divides the histogram into two parts: one representing background objects while the other one representing foreground objects. For this reason this process is also called "thresholding".

There are several way to define the more appropriate binarization threshold, but the one used in this work is always the so called Otsu's algorithm: the algorithm works by exhaustively searching for the threshold that minimizes the weighted within-class variance, or, from another point of view, maximizes the between-class variance.

Binarization is often a pre-processing technique applied before morphology transformation and/or connected components analysis.

2.4.2 Color segmentation

Color segmentation consists in finding a range of pixel values that well separate image objects. This can be achieved through automatic algorithms, e.g. k-means clustering, or using a user defined range, e.g. when the user already knows which colors are in the image and which one he has to look for. In addressing problem 3, color segmentation has been employed, using two user defined ranges. I felt safe in assuming user defined ranges because the problem was to be solved in a fully controlled environment: under this assumption I have preferred the fastest and easiest approach.

2.5 Morphology transformation

Morphology transformation consists in manipulating either foreground or background pixels by means of a set of pixels called "structuring element". It is usually applied to binary images, altough not limited to. The basic morphological operators are erosion, dilation, opening, erosion followed by dilation, and closing, dilation followed by erosion.

Erosion and dilation are the two basic operators that, when combined, gives opening and closing operators. Erosion operation uses a structuring element for probing and **reducing** the shapes contained in the input image, while dilation is used for probing and **expanding** the shapes contained in the input image. Opening and closing, instead, are usually exploited for morphological noise removal. Opening removes small objects from the foreground of an image, placing them in the background, while closing removes small holes in the foreground, changing small islands of background into foreground. These techniques can also be used to find specific shapes in an image. Opening can be used to find things into which a specific structuring element can fit (edges, corners, ...).

In problem 2 I have exploited morphological transformation to isolate the tab of the cap before searching for it as a connected components.

2.6 Connected components labeling

Once foreground/background segmentation has been accomplished, in most applications the next task deals with analysis of the individual foreground objects to achieve some kind of high level knowledge on the observed scene, e.g. objects orientation, area, perimeter, position and so on. A connected component of a binary image is a **maximal** connected foreground region. A connected component detection algorithm (e.g. 2-scans) though relies on the notion of connectivity that in turn is related to the notion of distance in the discrete 2D space. The two most common distances in E^2 are the *city-block distance* D_4 and the *cheesboard distance* D_8. In problem 2 connected component based on D_4 have been found.

2.7 Blob Analysis

Usually, after connected components have been detected, the next step is to compute some metrics from them, relevant for solving the task at hand. As it will be shown, in problem 2, the barycentre, area and perimeter of the connected component representing the tab of the cap have been computed: the former has been used to compute the rotation angle with respect the cap center, while the other twos are necessary for computing the form factor, needed to recognize the tab of the cap from the other detected connected components.

The form factor is the ratio between the squared perimeter and the area, and in a continuous space takes its minimum value, 4π, for a circle. It is also scale invariant. Obviously in the discrete domain, the form factor takes its smallest value not for a circle but for an octagon or a diamond (depending on whether the 8-connectivity or 4-connectivity is employed to calculate the perimeter).

In my case, I have used the form factor to distinguish between a compact and convex object against some traces of borders of the cap that survived the morphological transformation: they appeared as fragments of circumference with a high form factor.

2.8 Polar transformation

Polar coordinate transformation is used to correct circular objects in images or to transform them into circular objects. A polar transformation maps any point (x, y) on the Cartesian plane to the polar coordinates (θ, ρ) in the polar coordinate system by the following calculation formula:

$$\begin{aligned} \rho &= k_\rho \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \theta &= k_\theta atan2((y - y_c), (x - x_c)) \end{aligned} \tag{2.1}$$

where k_ρ and k_θ are two constants relative to image size, while x_c and y_c are the origin of the coordinate system. The definition of the system origin is an important aspect that influences the transformation: e.g. it will be shown in problem 2 an example of polar transformation with the origin outside the image borders.

2.2 is an example of polar transformation applied to one of the provided image of project 2.

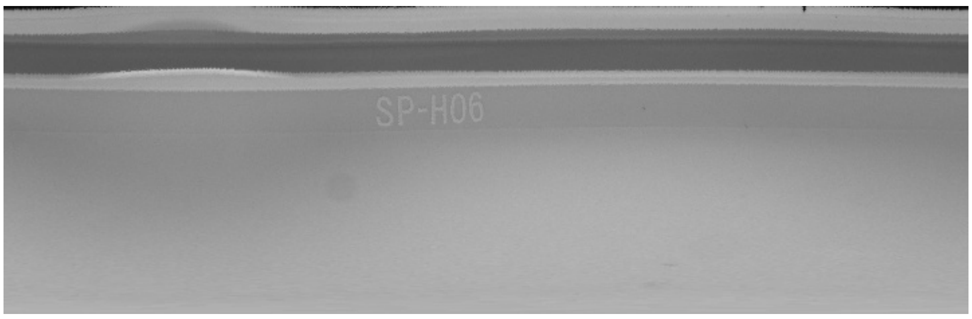


Figure 2.2: Polar transformation applied to a sample image.

3 Project 1 - Plastic cap liner inspection

3.1 Problem outline

The goal of project 1 is to develop a program to locate defects in the liner of a plastic cap. The provided ground thruth image set contains six good samples, five defective samples and other five samples declared as defective but without a real defect (I will ignore them in the discussion but I will use them as input to the program and see what happens). All the provided images are in grayscale.

An example of a good and a defective cap is shown in 3.1.

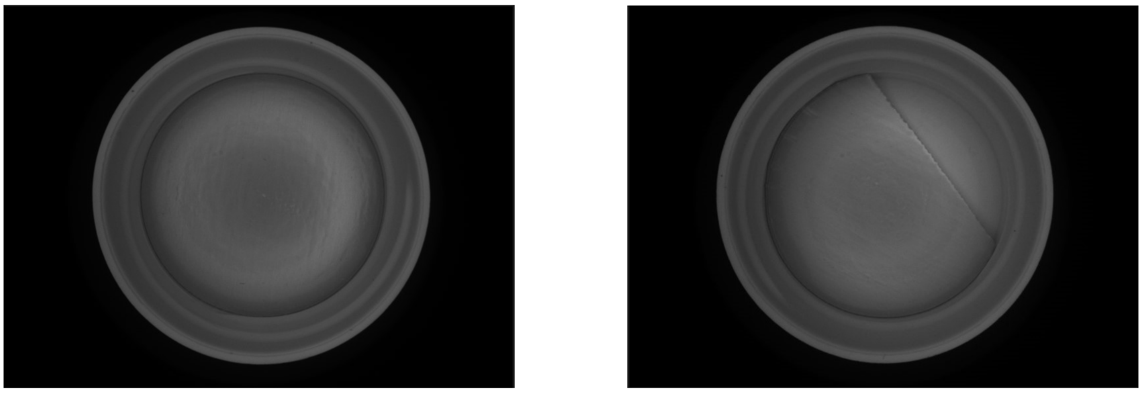


Figure 3.1: Instances of good (left) and defective (right) plastic cap.

Among the defective caps, one completely misses the liner, while the other four have an incomplete liner: the defect, that has to be detected and localized, appears like a straight cut on the liner, like in the right hand side of 3.1.

The defect detection problem is here addressed by a pipeline of image processing and computer vision techniques that reads the input image, classifies it and localize the defect, in case there are (3.2).

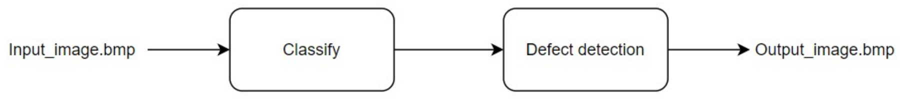


Figure 3.2: Defect detection high-level pipeline.

In particular the classification step is a multi-class problem where the classes are "good", "missing_liner" and "incomplete_liner". The classification step is needed in order to prevent good and missing-liner images to reach the defect localization step and waste computation time.

3.2 System description

The implemented procedure, as shown in 3.3 is:

1. Read input image
2. Search for the cap through Circle Hough Transform
3. In the original image, use the previous result to compute a masked histogram for only the pixels inside the cap and compute the average brightness
4. Based on a configurable threshold, classify "missing_liner" images
5. On the only images with liner ("good" or "incomplete_liner" samples) run another CHT to detect liner boundaries
6. Smooth the image using first a median and then a Gaussian filter, then run a Canny edge detector on the masked smoothed image. In the Canny's output, ignore liner contours and check the number of pixels with value greater then a predefined threshold
7. Segregate "incomplete_liner" from "good" samples and, on "incomplete_liner" ones, localize the straight defect, through a Linear Hough Transform, and draw its oriented MER (Minimum Enclosing Rectangle)

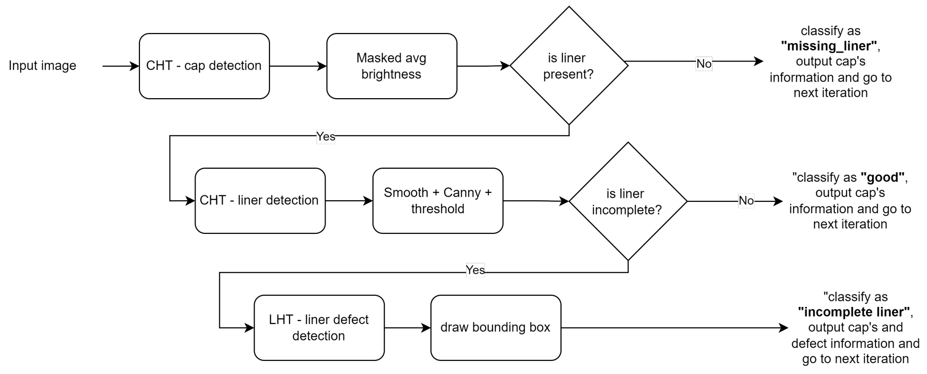

Input image
CHT - cap detection
Masked avg brightness
is liner present?
No
classify as "missing_liner", output cap's information and go to next iteration
Yes
CHT - liner detection
Smooth + Canny + threshold
is liner incomplete?
No
"classify as "good", output cap's information and go to next iteration
Yes
LHT - liner defect detection
draw bounding box
"classify as "incomplete liner", output cap's and defect information and go to next iteration

Figure 3.3: Defect detection detailed pipeline.

In the next section, a detailed description of each step is given.

3.3 Experimental setup and processing details

Following step by step the procedure described in the previous section, here I will describe how an input image is processed and what information is extracted.

3.3.1 Step 1 and 2 - Read image and Cap detection through Circle Hough Transform

Cap detection is performed through Circle Hough Transform on the input image. For the sake of efficiency, OpenCV implements a detection method slightly trickier than the standard Hough Transform: the Hough gradient method. It consists in two main stages: the first one involves edge detection and finding the possible circle centers, then the second one finds the best radius for each candidate center.

Usually it is a good practice to smooth the input image before searching for shapes. However, the OpenCV implementation of CHT already implements smoothing before performing the shape search and experimental results showed that adding another smoothing step wasn't needed.

Results of CHT on input image is shown in 3.4.

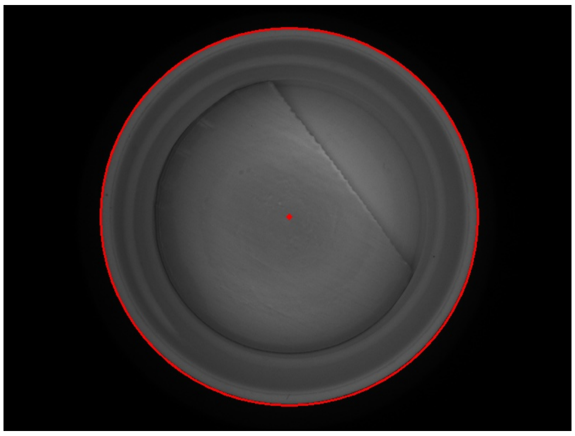


Figure 3.4: Cap detection through CHT.

CHT parameters are user-configurable and were determined through a semi-automatic iterative search: semi-automatic because, without an already labeled ground thruth, the only automatizable step was to discard parameters combinations that did not find any circles. However, this approach is safe because processing takes place under the assumption of a fully controlled environment.

CHT parameters for the cap detection are:

- cap_minDist=20
- cap_param1=200
- cap_param2=60

- cap_minRadius=0
- cap_maxRadius=0

Where *minDist* is the minimum distance between any two circumferences, *minRadius* and *maxRadius* are the lower and upper limit for the radius of the circumference to search for, *param1* is the upper threshold for the internal Canny edge detector and *param2* is the accumulator threshold for the circle centers at the detection stage.

The results of this step are cap center (a_{cap}, b_{cap}) and cap radius r_{cap}. They are both requested measures and useful information that will be used in the next steps.

3.3.2 Step 3 and 4 - Average brightness and classification of missing liner images

In this step the average brightness of the cap is computed in order to determine wheter the liner is there or not. Average brightness is a valid metric to distinguish between caps with and without liner because it has a very different reflectance with respect to the bottom of the cap, which is already visible to the naked eye 3.5.

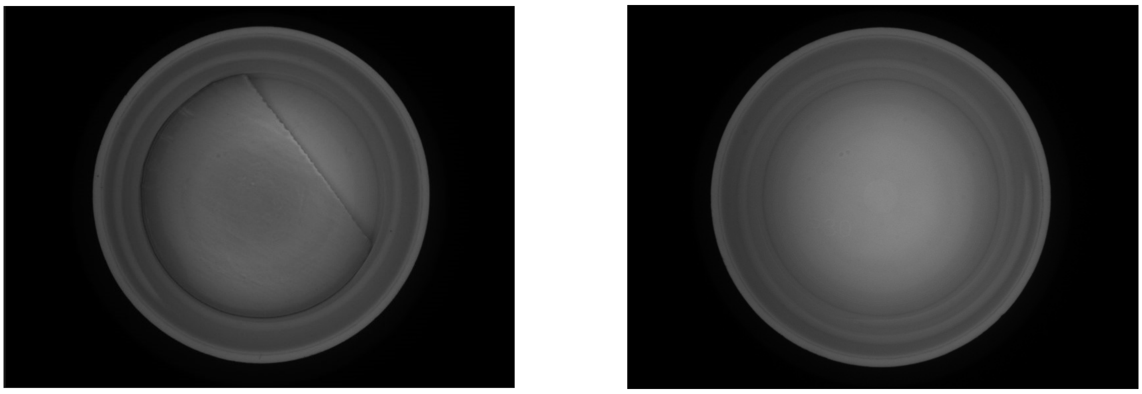


Figure 3.5: Missing liner cap vs cap with liner.

The average brightness threshold has been determined experimentally looking at the distribution of average brightness of only caps with liner and taking $\mu + 3\sigma$, as shown in 3.6.

From here on, the processing proceeds only on images with liner.

3.3.3 Step 5 - Liner detection through Circle Hough Transform

Keeping in mind the assumptions made in 3.3.1, CHT is employed again in order to detect cap liner. Once the parameters have been determined, the results are like the one shown in 3.7

CHT parameters for the liner detection are:

- liner_minDist=100
- liner_param1=50
- liner_param2=30

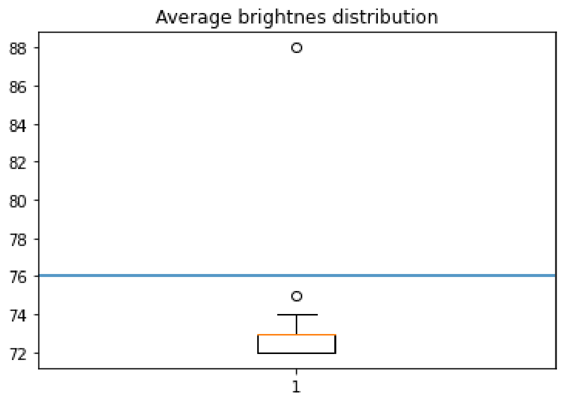


Figure 3.6: Average brightness distribution and chosen threshold.

- liner_minRadius=180
- liner_maxRadius=200

Notice that the maxRadius is limited by the cap radius.

The outputs of this stage are the center (a_{liner}, b_{liner}) and radius r_{liner} of cap liner.

3.3.4 Step 6 - Smoothing and apply Canny edge detector

Once the liner has been localized, the image is first smoothed through the convolution of two filters: first a $5x5$ median filter and then a $11x11$ Gaussian filter with $\sigma = 1.5$. Gaussian filter size has been determined following the rule of thumb of $k = \lceil 3\sigma \rceil$, being the filter size $2k + 1 \times 2k + 1$. Filters dimensions and σ have been determined by experimental trials: the parameters combination that is able to detect only the straight edge after the Canny edge detector was taken as the best one.

The smoothed image is then masked, removing everything is outside liner circumference, and passed to a Canny edge detector. The result is masked again (with a circular mask with radius $r = r_liner - 10px$) in order to isolate defect edge pixels (3.8).

3.3.5 Step 7 - Classification of incomplete liner images and defect detection

The output of Canny's edge detector is a binary image where foreground pixels represent strong edges, so reasoning on the Canny's output is equivalent to reasoning on gradient magnitude and orientation (as required by the problem hints).

In this stage a threshold on the number of white pixels is applied in order to classify an image as "good", if below the threshold, or "incomplete_liner" when above. As already said, this is equivalent to count pixels which gradient magnitude exceeds a certain threshold.

Images classified as "good" stop here, while "incomplete_liner" images go through the defect detecion phase. This task could in principle be solved in different ways, e.g. taking the masked Canny's output and draw a bounding box around the white pixels (not robust due to spurious

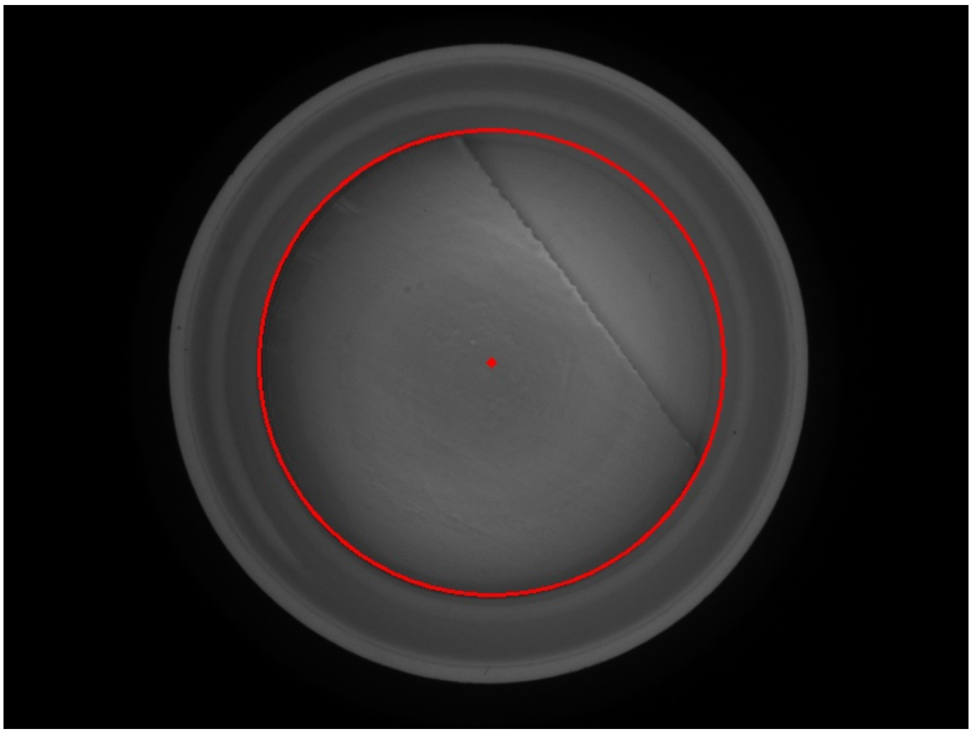


Figure 3.7: Liner detection through CHT.

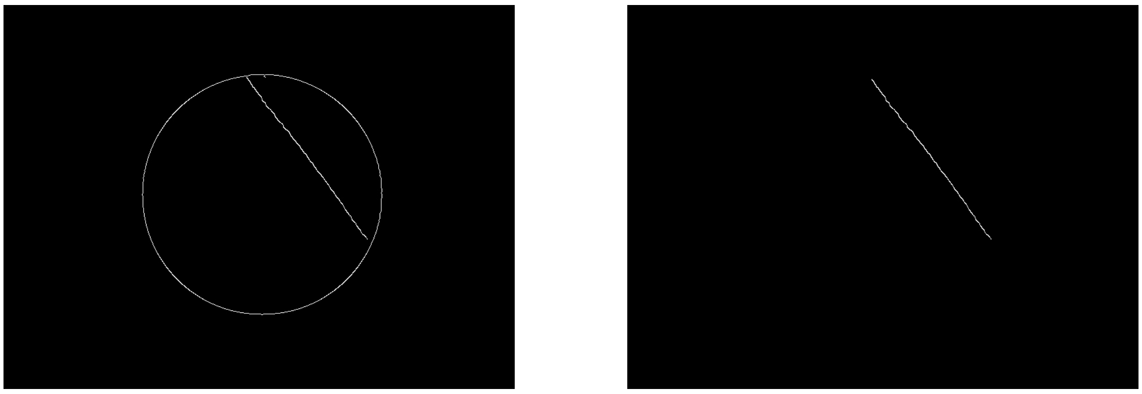


Figure 3.8: Canny output, original (left) and masked (right).

edges that make the bounding box grow), or searching for connected components in the Canny's output and use some metric to discard spurious edges.

However, I decided to solve this task by using Line Hough Transform (LHT), under the assumption (given in the project description) of **straight edge** (due to the manufacturing process). The LHT is more robust with respect to drawing the bounding box around Canny's edges, and simpler than connected components, in terms of hand-crafted rules. Results of LHT as defect detector are shown in 3.9

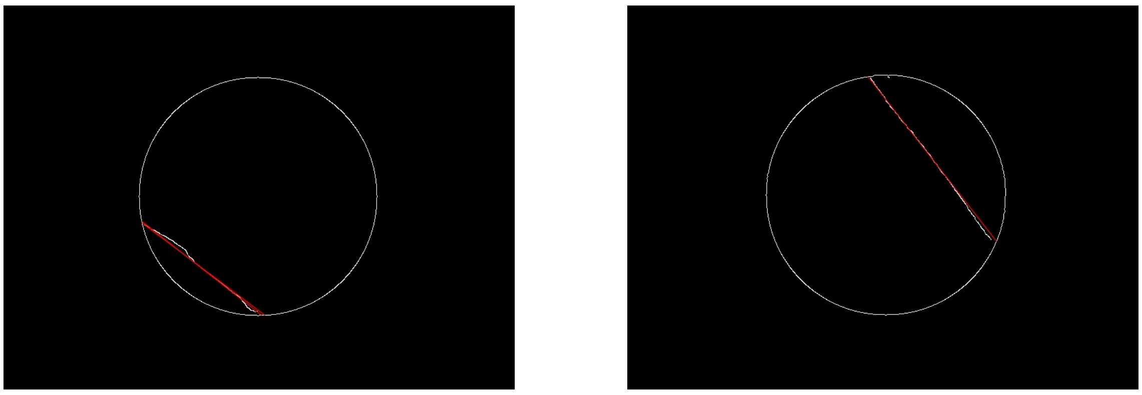


Figure 3.9: LHT defect detector. Multiple lines (right), single line (left).

In particular, I have developed two variants of LHT: one called "Single Line HT", where the output of the Hough Transform is constrained to a single line; the other called "Multiple Lines HT", with no constraints on the number of lines found in order to handle also non straight edges through segments approximation.

Once one or more lines have been found, for each line I compute the four vertices of a rectangle that would enclose each single line and collect them in a single array. The final rectangle is drawn as the minimum area rectangle that enclose all the just computed rectangle vertices 3.10.

3.4 Results and conclusion

The developed pipeline is able to correctly classify and detect the 100% of images and defect.

Although no optimization was adopted, the processing time per image is around 20ms for "missing_liner" image, around 65ms for "good" ones and around 80ms for "incomplete_liner" ones, on a commodity computer with an Intel Core i5 and 16Gb of RAM.

As already said, the parameters were determined through experimental trials in a semi-automatic fashion, because no groud thruth was provided (meaning no ground thruth for defect coordinates or for the cap and liner centers and radii). However, being the image set from a fully controlled manufacturing environment, I'm pretty sure that this program would perform well on new samples.

For what concerns the defect localization, I think, from a qualitative point of view, that the best method is what I called "Multiple Line Houg Transform": an Hough transform with no constraint on the number of lines to be found.

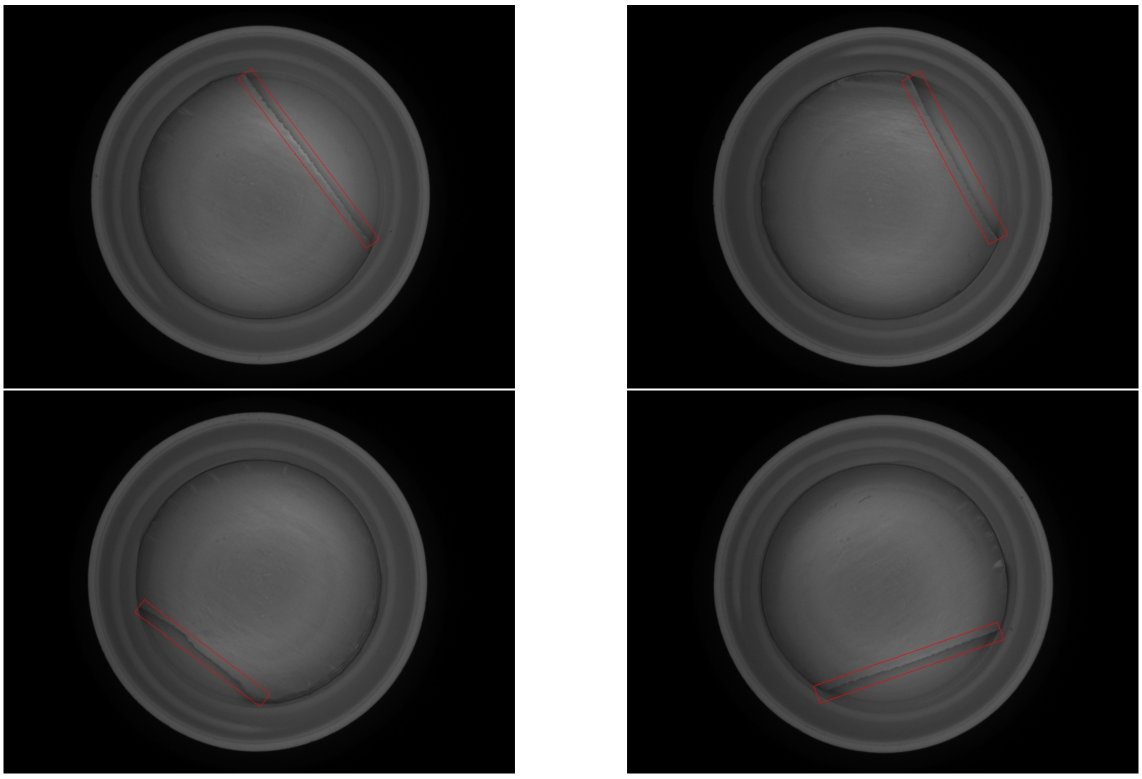


Figure 3.10: LHT defect detector. Multiple lines (right), single line (left).

4 Project 2 - Cavity number preprocessing

4.1 Problem outline

The goal of project 2 is to develop a program to preprocess an image and get it ready to perform the OCR of the cavity number of a plastic cap. The cap has an external tab at a fixed position in relation to the cavity number (4.1). The to be developed program should give as output the rectified crop containing the cavity number. The provided ground thruth image set contains 29 grayscale images, all with a cavity number and all presented with different rotation angle.

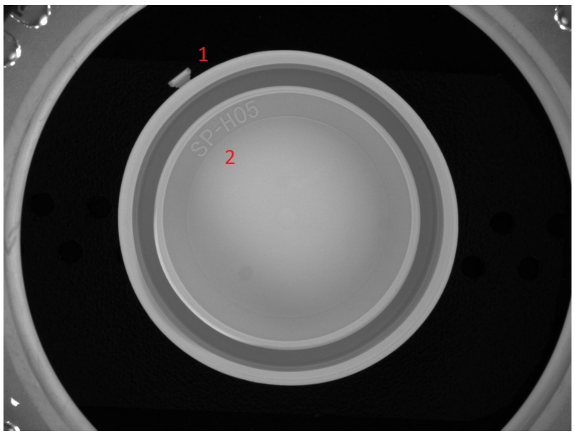


Figure 4.1: Sample image with tab (1) and cavity number (2).

4.2 System description

The implemented procedure, as shown in 4.2 is:

1. Read input image
2. Search for the cap and the apparatus outside the cap through Circle Hough Transform
3. Image binarization and morphological transformation
4. Connected components detection and selection
5. Rotation and cropping
6. Polar rectification and cropping again

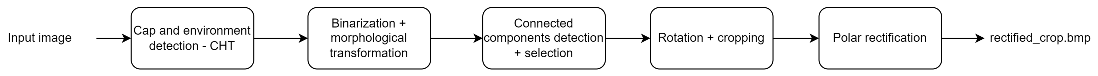

Input image
Cap and environment detection - CHT
Binarization + morphological transformation
Connected components detection + selection
Rotation + cropping
Polar rectification
rectified_crop.bmp

Figure 4.2: cavity number preprocessing detailed pipeline.

In the next section, a detailed description of each step is given.

4.3 Experimental setup and processing details

Following step by step the procedure described in the previous section, here I will describe how an input image is processed and what information is extracted. In solving this task, as already said for the previous problem, I assume to work under a fully controlled environment: this allow me to make very specific assumptions.

4.3.1 Step 1 and 2 - Read image and Cap detection through Circle Hough Transform

Cap detection is performed through Circle Hough Transform on the input image. Because in the next steps a connected components search will be applied, it would be better to remove also external apparatus visible in the image corners. Being this a fully controlled environment, I could safely assume that stuff outside the cap will always appear with a round shape and so I could exploit again CHT to detect that structure.

Results of CHT on input image is shown in 4.3.

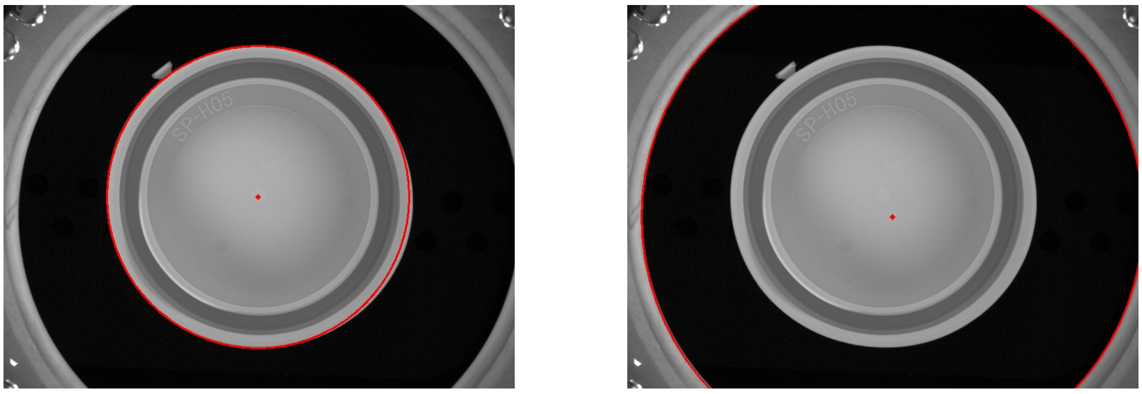


Figure 4.3: Cap and outer apparatus detected by CHT.

CHT parameters for the cap detection are:

- cap_minDist=120
- cap_param1=200
- cap_param2=60
- cap_minRadius=220
- cap_maxRadius=230

while for detection of the outer apparatus are:

- bg_minDist=120

- bg_param1=200
- bg_param2=60
- bg_minRadius=250
- bg_maxRadius=0

The result of this step are the center and radii of both cap and, let's say, background.

4.3.2 Step 3 - Morphological transformation

The results from the two previous CHT are used to mask the original image in order to get rid of outside object and the inside of the cap. The reason is that I am aiming to isolate the tab of the cap in order to get its position. So the input image is first binarized by Otsu's algorithm and then masked with the circumferences resulting from the previous step. Result of this steps are depicted in 4.4

As it can be seen, the resulting image (the middle one in 4.4) is not yet ready to be passed to a connected component detection algorithm, or in other words, it can be better cleaned applying morphological transformation. Because I wanted to get rid of small and/or elongated foreground objects, I have applied an *opening morphological operator* with a 7×11 rectangular structuring element. The result is shown in the third picture of 4.4.

Sometimes it may happen that some foreground fragments survives the morphological opening: these cases are handled by exploiting other morphological features, like the *from factor*, as described in the next session.

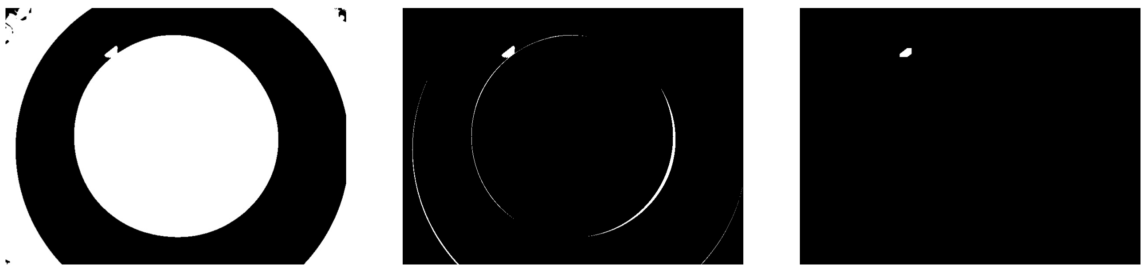


Figure 4.4: Original image after applying binarization, masking and morphologial transformation.

The result of this phase is a binary image ready to be processed by a connected component detection algorithm.

4.3.3 Step 4 - Connected components detection and selection

The result obtained so far is passed as input to a connected components detection algorithm in order to detect tab position. Detecting tab position is equivalent to detecting the centroid of the connected component representing the tab. The chosen connectivity is 4, but also 8-connectivity works well.

As already said, sometimes more than one foreground object survives the morphological opening and this generates an ambiguity that must be solved in order to identify the real tab. To this end, I have chosen to compute the form factor of the detected components. As mentioned in 2.7, the form factor is computed as $ff = \frac{p^2}{4\pi A}$ and the more compact and "round" the object, the close the form factor to 1.

So I looked for the connected component with the minimum form factor.

4.3.4 Step 5 - Rotation and cropping

Once the tab is detected, its centroid is used to compute the offset angle, with respect to a vertical line, taking into account that the y axis goes from top to bottom. The following function computes θ independently on the tab position.

$$\theta = \begin{cases} atan(\Delta y, \Delta x) + \frac{\pi}{2} & if \Delta y > 0 \wedge \Delta x > 0 \\ atan(\Delta y, \Delta x) + \frac{3\pi}{2} & if \Delta y < 0 \wedge \Delta x < 0 \\ -atan(\Delta y, \Delta x) + \frac{3\pi}{2} & if \Delta y > 0 \wedge \Delta x < 0 \\ -atan(\Delta y, \Delta x) + \frac{\pi}{2} & if \Delta y < 0 \wedge \Delta x > 0 \end{cases} \tag{4.1}$$

Where $\Delta y = (y_c - b_{cap})$ and $\Delta x = (x_c - a_{cap})$.

Based on the computed θ the matrix M corresponding to the affine transformation is computed and used to rotate the image. At this point, what is left is to crop the cavity numer. To this end I compute the rotated coordinates of the centroid of the tab (x_{rc}, y_{rc}), using the same M, and then I use y_{rc} to dinamically compute y_1 and y_2 of the crop. In particular:

$$\begin{aligned} y_1 &= y_{rc} + (b - y_{rc}) * \Delta y_1 \\ y_2 &= y_{rc} + (b - y_{rc}) * \Delta y_2 \\ x_1 &= a_{cap} - (y2 - y1) * \Delta x_1 \\ x_2 &= a_{cap} + (y2 - y1) * \Delta x_2 \end{aligned} \tag{4.2}$$

where Δs are user defined parameters that controls crop dimension.

The result of this step is the crop of the region containing the cavity number to be processed by the OCR tool after rectification, and it is shown in 4.5

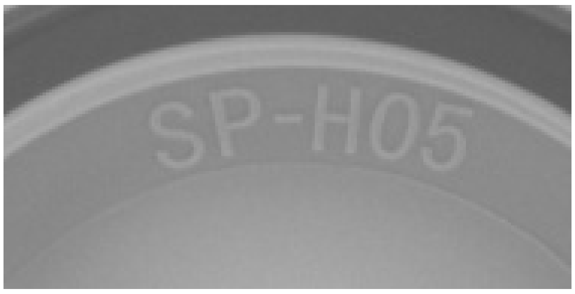


Figure 4.5: Crop of the cavity number.

4.3.5 Step 6 - Polar rectification and cropping again

The crop obtained so far, although human readable, is not ready to be passed down to an OCR tool. So, the last step is to rectify the crop, by applying a polar transformation. Applying a polar mapping here means implicitly assuming that the input image (or its final crop) is given in polar coordinates and it will be mapped into a Cartesian system.

The transformation is described by:

$$\begin{aligned}\rho &= k_\rho \sqrt{(x - x_c)^2 + (y - y_c)^2} \\ \theta &= k_\theta atan2((y - y_c), (x - x_c))\end{aligned} \tag{4.3}$$

where k_ρ and k_θ are two constants relative to image size, while x_c and y_c are the origin of the coordinate system. The definition of the system origin is an important aspect that influences the transformation. Indeed, in this case, the center of the transformation is taken outside the crop: it is the cap center expressed in the crop reference system $(a_{cap} - x_1^{crop}, b_{cap} - y_1^{crop})$. The radius of the mappping, instead, is the cap radius.

Applying a polar to cartesian mapping produces the result shown in 4.6

At this point I would like to have a rectangular crop. To do this I have implemented a search algorithm that finds the biggest rectangle contained in the crop 4.6.

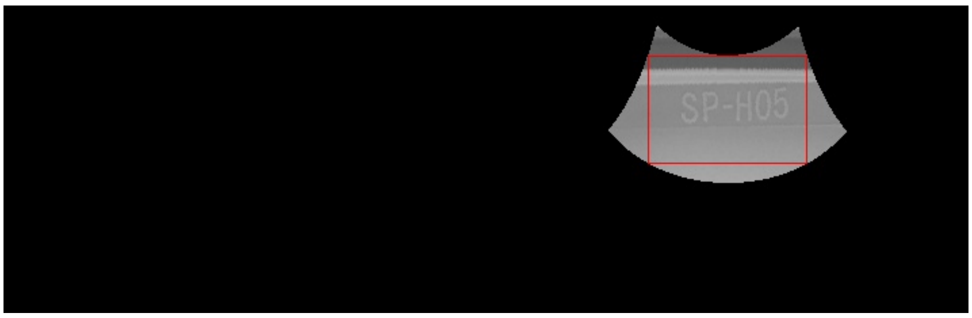


Figure 4.6: Biggest contained rectangle.

The algorithm first gets the polar transformed crop contours. Under the assumption that a rectangle with at least two vertices on the contour gives a good enough solution, it computes and store the area for each combination of two contour points. It sorts the found rectangles by decreasing area and starting from the first one check if it goes outside the crop borders (contours). If so, it is discarded and the algorithm jump to check the next one. When the first valid rectangle is found the algorithm stops.

Of course such an approach could be employed in this problem thanks to the small dimension of the rectified crop, otherwise it would not be a wise choice.

The pipeline ends by cropping again the just found rectangle and outputting an image as shown in 4.7.

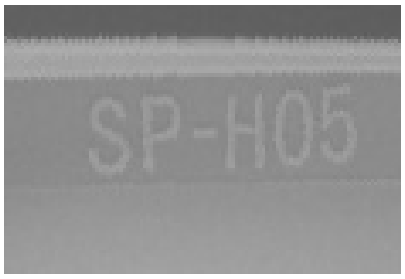


Figure 4.7: Polar rectified cropped cavity number.

4.4 Results and conclusion

Results of the pipeline are very good: it is able to find the cavity number, to rotate, crop and rectify it for the 100% of the images. The average processing time per image is around 0.5 ms.

The proposed approach is not the only possible one. I think that a valid alternative could be to employ pattern matching algorithms, like SIFT or SURF, to find the tab of the cap and avoid all the masking and connected component detection step.

For the tab finding stage, the algorithm showed a 100% accuracy in recognizing tab of the cap, so I didn't need to fall back to the Haralick's circularity metric. The reason is that the form factor takes its minimum value in the discrete plane for an octagon or a diamond (depending on the chosen connectivity): in this work 4-connectivity has been chosen and so the form factor would take its minimum value for a diamond, a shape very similar to the hexagonal shape of the tab after morphology transformation.

In my opinion, the most interesting parts in this problem were the morphological transformation, the form factor and the polar transformation. The first because I could see with my eyes how powerful it is to employ such techniques for manipulating shapes in binary images; the second because I think it has been used as a smart trick to automatically choose the wanted connected component; the third because it is always fascinating for me to apply geometry and mathematics to concrete applciations.

5 Project 3 - Off center decoration

5.1 Problem outline

The goal of project 3 is to develop a program to compute the off-center of a decoration in an aluminum cap. The color of the cap is red while the decoration is grey. The off-center is defined as the distance between the center of the decoration and the center of the cap. The provided ground thruth image set contains 27 RGB images (5.1).

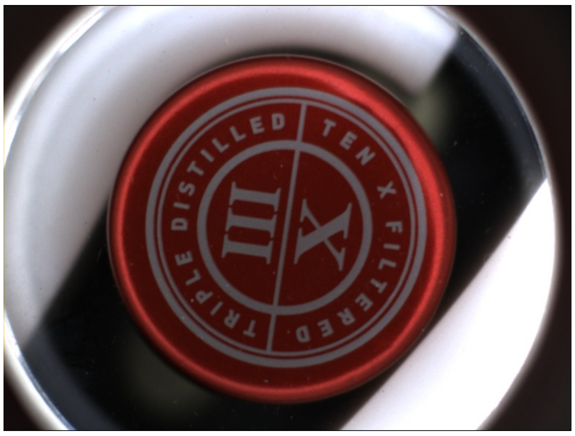


Figure 5.1: Example of provided images.

5.2 System description

The implemented procedure, as shown in 5.2 is:

1. Read input image
2. Cap color segmentation
3. Smoothing and cap detection through CHT
4. Decoration color segmentation
5. Smoothing and decoration detection through CHT
6. Distance computation

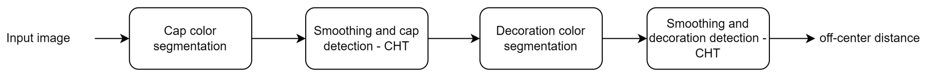

Input image
Cap color segmentation
Smoothing and cap detection - CHT
Decoration color segmentation
Smoothing and decoration detection - CHT
off-center distance

Figure 5.2: Off-center decoration detailed pipeline.

In the next section, a detailed description of each step is given.

5.3 Experimental setup and processing details

Following step by step the procedure described in the previous section, here I will describe how an input image is processed and what information is extracted.

Before starting to process the single image, a preliminary step was needed in order to get colors to be used for the segmentation. By problem definition, the two colors to look for are dark-red for the cap contour, and grey for the decoration contour, and they are always the same in the given image set. On this assumption, it is safe to fix the color range when segmenting. It is worth to notice that what will be fixed here is not the color code but a color range, in order to address changes in illumination.

In order to get the color values to search for, I read the color image and converted it in HSV color space: in this way I can reason on colors by only taking into account the first and second channel, Hue (color) and Saturation (amount of color), and leave a quite wide range for Value (amount of light) in order to take into account brightness variation among the image set. Basically, what I needed was the hue value for dark-red and grey. Instead of looking at the whole image, I cropped a 250 $\times$ 250 region around the center, where I was sure both the sought colors were. Again, I was not interested in the number of pixels of a certain color, but only in the spanned range, mainly in the Hue channel.

5.4 shows the hsv channels splitting for a sample image in the region shown in 5.3.

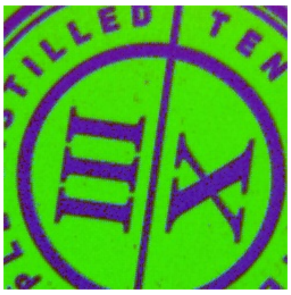


Figure 5.3: 250x250 region of d_01.bmp.

The two peaks in the Hue channel identify the two colors: red and grey. The two peaks in Saturation channel indentify two different sets of pixel: the peak on the right (the greater one) corresponds to the set of high contrast dark red pixels, while the weakest one (because a smaller number of pixels of that kind has fallen in the cropped region) corresponds to the grey contours. It represents the decoration. In the Value channel there is an almost continuous brightness in the region of interest, so I can take a wider the range.

The result of this preliminary analysis was the definition of the dark-red and grey color ranges that will be used to perform color segmentation:

- red_lower_bound = (0, 210, 50)
- red_upper_bound = (7, 250, 180)

and

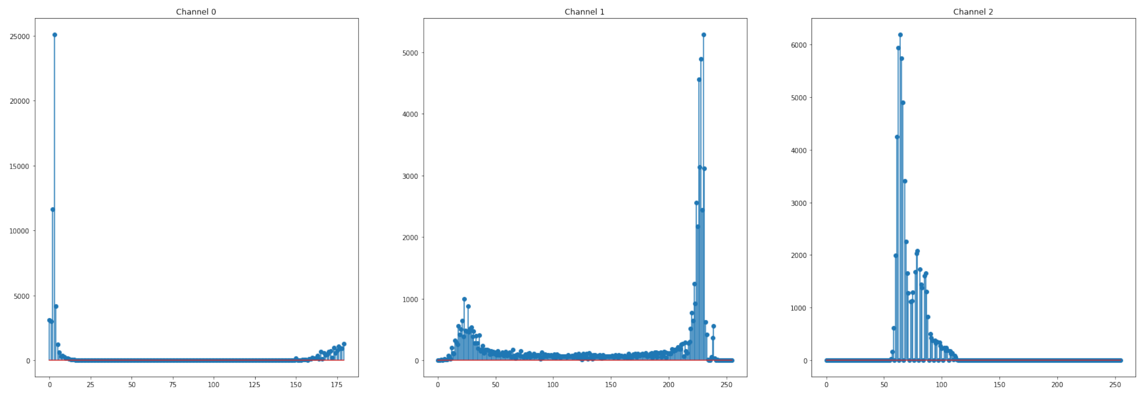


Figure 5.4: Hue, Saturation and Value of a 250x250 region of d_01.bmp.

- grey_lower_bound = (149, 0, 50)
- grey_upper_bound = (179, 50, 180)

5.3.1 Step 1 and 2 - Read image and Cap color segmentation

In this phase the input image is segmented based on color range defined in the previous section. The result of this step is 5.5

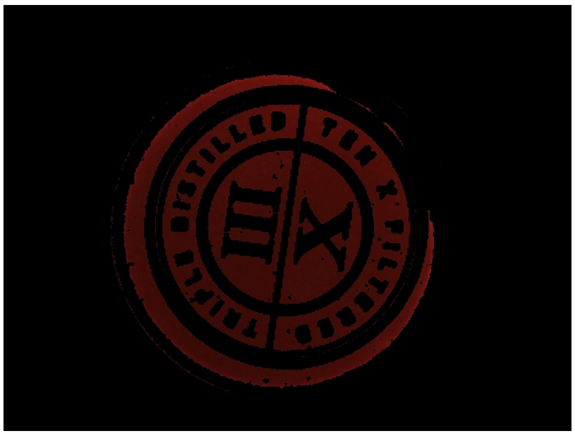


Figure 5.5: Dark red pixels of a sample image.

5.3.2 Step 3 - Cap detection

The result of previous section is filtered by a 5×5 median filter and then a Circle Hough Transform is applied in order to look for the cap borders. CHT parameters are

- cap_minDist=5000
- cap_param1=200
- cap_param2=10

- cap_minRadius=205
- cap_maxRadius=0

The result of this step, visualized on the original input image is shown in 5.6

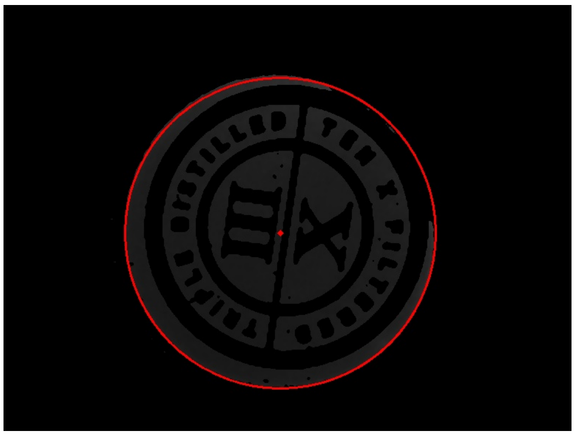


Figure 5.6: Dark red cap detected through CHT.

5.3.3 Step 4 - Decoration color segmentation

In this phase the input image is segmented based on color range previously defined for the grey decoration. The result of this step is 5.7

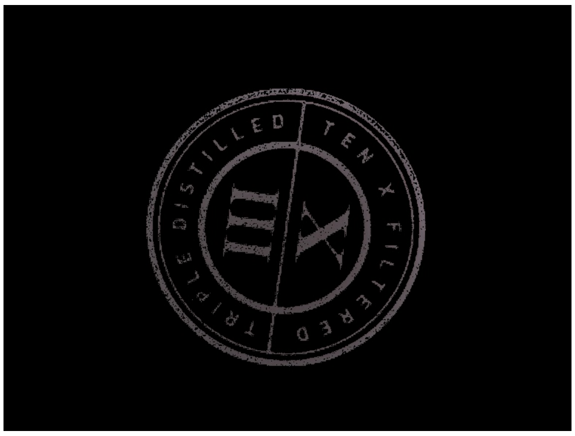


Figure 5.7: Grey pixels of a sample image.

5.3.4 Step 5 - Decoration detection

The result of previous section is filtered by a 3×3 bilateral filter and then a Circle Hough Transform is applied in order to look for the cap borders. CHT parameters are

- decoration_minDist=5000

- decoration_param1=250
- decoration_param2=1
- decoration_minRadius=187
- cap_maxRadius=cap_maxRadius

The result of this step, visualized on the original input image is shown in 5.8

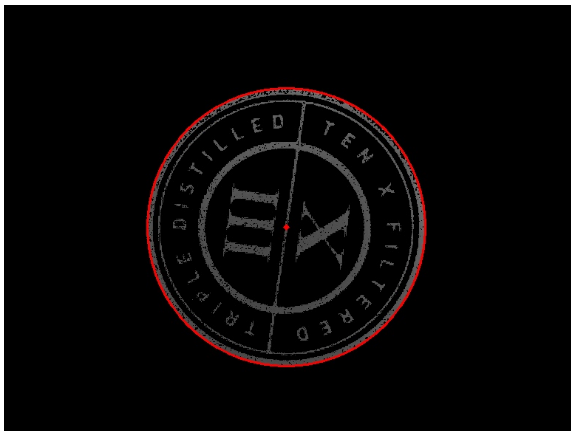


Figure 5.8: Grey decoration detected through CHT.

5.3.5 Step 6 - Off-center distance computation

In the last step, the Euclidean distance between cap center (a_{cap}, b_{cap}) and decoration center $(a_{decoration}, b_{decoration})$ is finally computed.

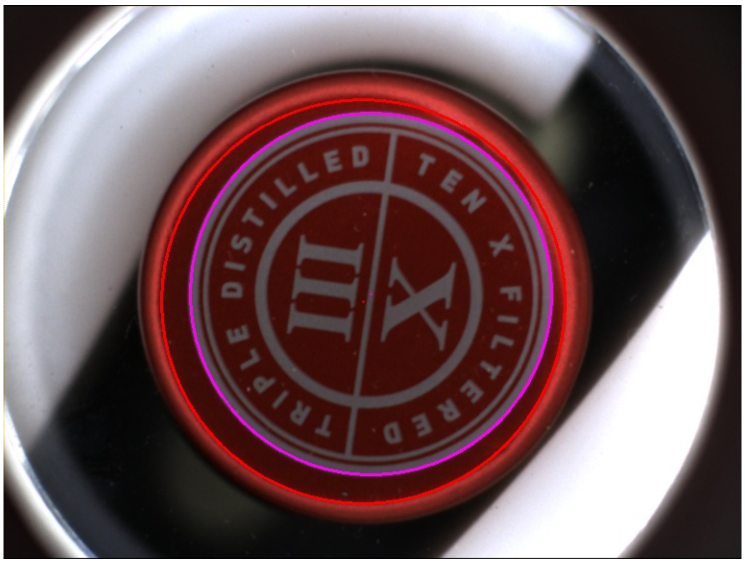


Figure 5.9: Example of final output.

5.4 Results and conclusion

In absence of a ground thruth it is hard to asses model performance. However, a quality assesment based on manual check of each single processed image shows that the approach works in practice. Because this problem took place in a controlled environment I could expect low variations of brigthness and noise, and so I could define a static color range to look for.

This approach, although not generalizable to caps or decorations of different colors, is faster with respect to more general statistical based segmentation techniques (e.g. k-means) that would require also the recognition of segmented colors before proceeding. The average processing time per image is around 0.45 ms.