Visual Inspection of Connecting Rods

Relazione di: Lorenzo Balducci

Progetto di:

Davide Giordano

Lorenzo Balducci

Gattoni Giacomo

The problem is a 2D object recognition problem that can be faced in various ways. The solution proposed is based on a quite-standard image processing pipeline followed by a contours searching and geometrical considerations for the extraction of the rods characteristics.

First step : smoothing

The smoothing step has the purpose to:

1) Partially remove dust in order to simplify successive computations.
2) Smooth the image in order to remove peaks in the contours angle ([3]).

The best smoothing filter to be used is a 5x5 gaussian filter (empirically verified in the image dataset given: https://iol.unibo.it/pluginfile.php/420187/course/section/156548/ispezione-bielle-immagini.zip).

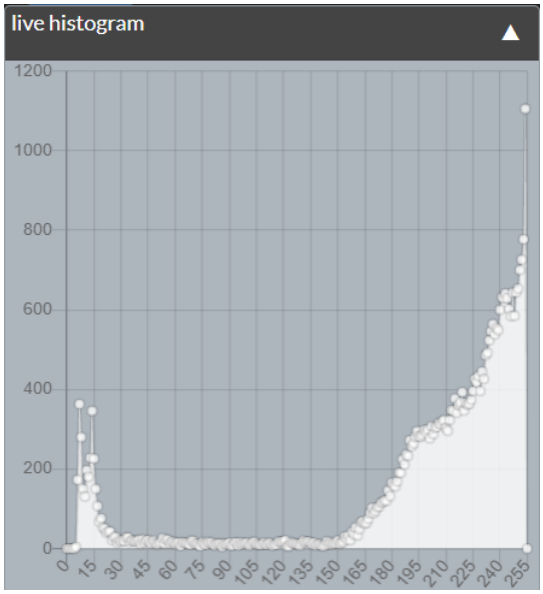
This is achieved with the GaussianBlur function of openCV

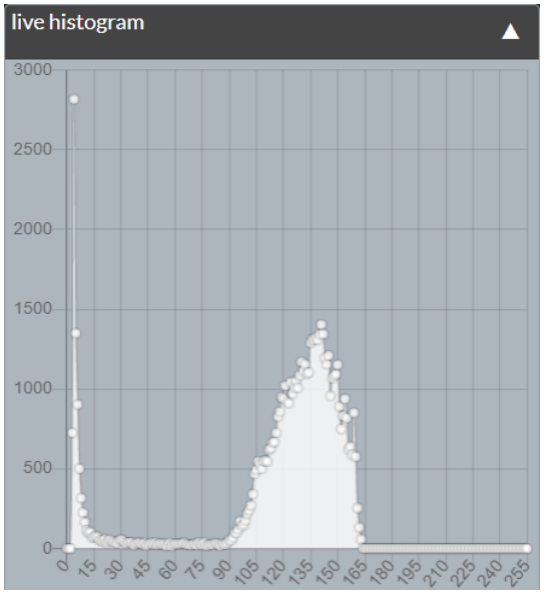
```
smoothedImg = cv2.GaussianBlur(img,(3,3), 0)
```

Note that the actual source code provide the possibility to use a different filter accordingly to the settings of the ui.

Second step : thresholding

From the moment that the images are taken on top of a backlighted transporting tape, pixels belonging to foreground would be always brighter than pixels belonging to objects, even if (as stated in the requirements sheet) the global illumination environment can change.





overexposed image (img TESI49.png) *underexposed image (img TESI50.png)*

From these considerations, is possible to stand that a statically thresholding fixed level will not be suitable, but a simple dynamically set thresholding level with OTSU algorithm is right for this environment.

This is achieved by threshold function of opencv

```
threshold, threshImg =
cv2.threshold(smoothedImg, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

Note that the actual source code provides in any case the possibility to use a manually set thresholding level accordingly to the settings of the user interface.

Third step: dust filtering

Accordingly to the requirements sheet, some images could have some grain of dust. Removing this dust with just a smoothing is infeasible because some piece of dust are considerably big and using an erosion is not as simple as it looks like (reference [2]).

The method used exploits the characteristic of the rods and isolate them in the image from the dust (not from the distractors).[1]The characteristics of a rod are:

1) The contour is the father in the contour's hierarchy: his index is -1 (see [1])
2) His shape is bigger than a minimum size

From the moment that a discretized image is available, the findCountours function can be used:

```
_, approxContours, approxHierarchy =
cv2.findContours(threshImg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

The CHAIN_APPROX_SIMPLE parameter approximate the shape with border points in order to reduce ram usage (see [1]). The RETR_TREE parameter is the way in which the hierarchy of the contours are retrieved, these data will be the key for exploiting holes inside the rods.

The output image is created starting from a mask of zeros with the resolution of the input image. In this mask, all the pixels inside a contour of a possible rod (distractors are not filtered here) are "marked" with 1 with fillPoly function of opencv.

The 2 characteristics of the rods exploited here are verified with the 2 if-statements inside the cycle: a check for the -1 in the hierarchy and a check for the minimumSize (this is the check that will actually filter out the dust).

Once the mask is ready the input image is multiplied with the mask.

```
mask = np.zeros(threshImg.shape, dtype="uint8")
for i,cnt in enumerate(approxContours):
    if(approxHierarchy[0][i][3] == -1):
        if(cv2.contourArea(cnt) > minimumRodsSize):
            cv2.fillPoly(mask, [cnt], 1)
threshImg = threshImg * mask
```

[1] Distractors are not filtered here because are filtered after the "single rod isolation" for handling the case in which a distractor is connected to a rod (an example of this is not provided in the image dataset but could be possible).

TESI92.png: example of grains of dust quite big

In this image, the process of gaussian smoothing followed by otsu thresholding will provide the following image (tons of the dust are filtered but not all).

The mask computed accordingly to the algorithm explained before will provide the mask in the equation below (purple is 0, yellow is 1).

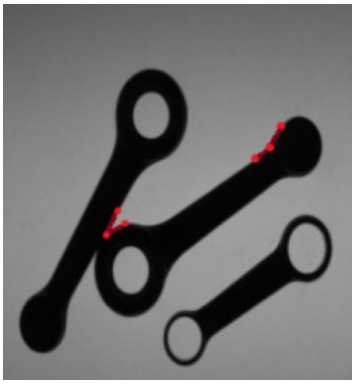
Fourth step: detaching connected rods - search of connected points

As stated in the requirements sheet, and as can be seen in the example below, the rods can be connected but not overlapped.

*TESI*51.png

The feature that can be exploited is that the contour of a rod forms in every 3 consecutive points, an angle less than 90 degrees.

When there is a connection, the angle is, for obvious but not demonstrated geometrical considerations, less than 90 degrees.



This consideration can be aggressed:

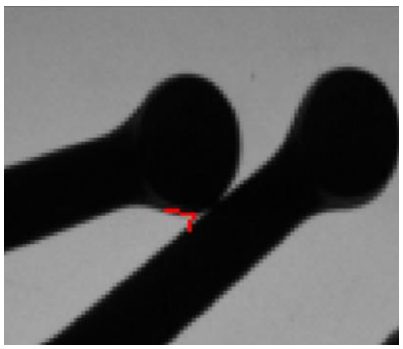
1) In a straightforward way analyzing the contours points angle
2) In an image processing way in which a corner detection algorithm like Harris's one should find that connection points as a corner

The first option is taken into account as a default one and will be explained here.

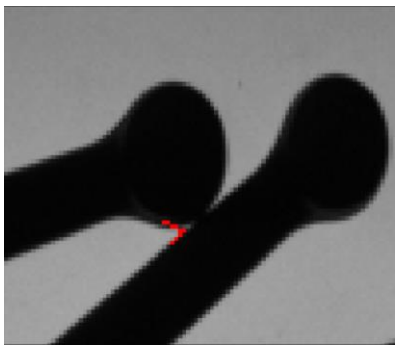
Contours angle algorithm:

For all the points with index j of the findContours method with the APPROX_SIMPLE parameter[2], is feasible to compute the angle between the points of indexes j-3, j and j+3 (the distance of 3 instead of 1 is taken for avoiding computing an angle affected by quantization error between near pixels).

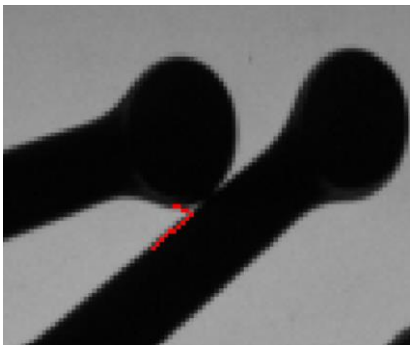
After this computation, multiple points are found for a single point of connection, even if NMS against the smallest angle found could be performed, a more simple algorithm was chosen. This because this algorithm is no intended to search for the best corner to perform image tracking or something similar but must find 2 good points to perform a "cut" for dividing rods. Because of this, the first corner point found is taken and the successive are discard until a point with angle > 90 degrees is found.



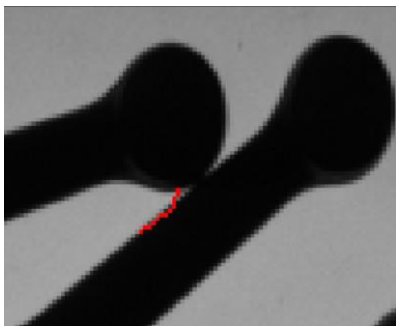
Found angle < 90 => taken



found angle < 90 => discard



found angle < 90 => discard

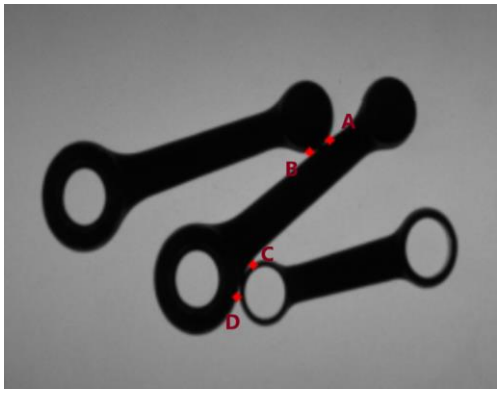


found an angle > 90 => from now, the next angle < 90 will be taken.

[2] Even if is used the approximated points of the contours, the corner will be in any case found. The points that are not available in the approximated contours are the one that forms a straight line: in those points there will never be a connection point (the algorithm is looking for corners, not straight lines).

Fifth step: detaching connected rods: matching connecting points

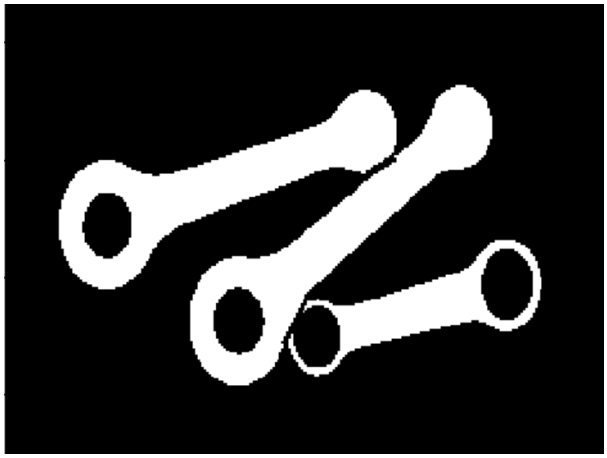
After the computation of the connecting points, a matching must be performed for allow the division of the rods.



In this example after the computation of connection points A,B,C and D, of course, the point A has to be matched with the point B and the point C with the point D.

The geometrical consideration that can be exploited is that “twins” points are the nearest, so matching nearest points 2 by 2 would lead to detach the connected rods with a white line that divide the 2.

The output result is show in the picture below



As a result, the rod at the right bottom corner gets opened because the width of the extremum of the rod was just 1 pixel and the algorithm explained before have “erased” that extremum contour.

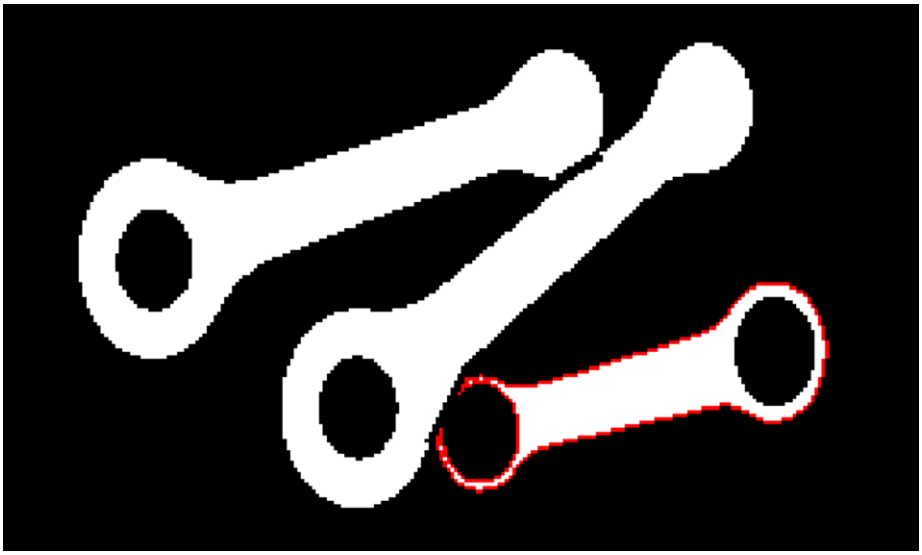
This problem can be resolved with 2 approaches:

1) Redraw 2 parallels lines with an offest of 1 pixel with respect to the one drawn before in order to recostruct a closed rod and recomputing a clean findContours.
2) The algorithm introduced in the next page (“single rod isolation”).

Sixth step: single rod isolation

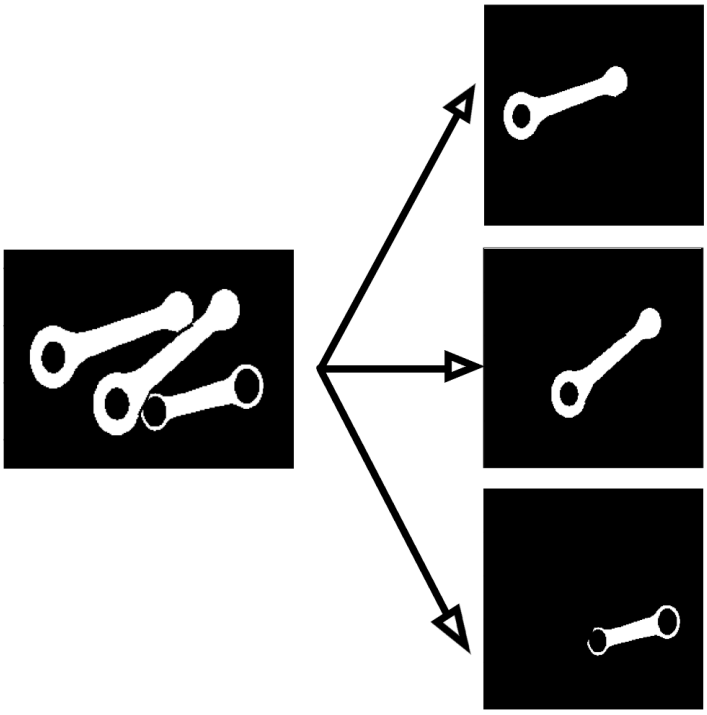
After the rods gets divided as explained before, the next step is to:

1) Recompute the contours (that will be wrong), for example in this image, the third rod contour (shown in red) is opened on the extremum for the reason explained before).



2) Extract, for every rods external contours, a mask for extracting pixels of the image belongin to that rod.

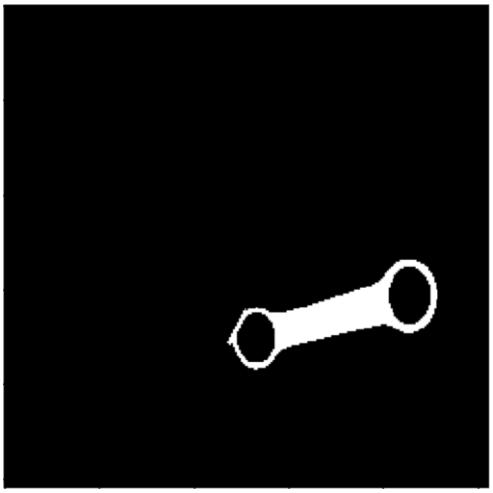
```
For cnt in contours:
      imageForSingleRods = correctThreshImg.copy()
      mask = np.zeros(imageForSingleRods.shape, dtype="uint8")
      cv2.fillPoly(mask, [cnt], 1)
      imageForSingleRods = imageForSingleRods * mask
```



3) For every image with just one rod, “rewriting” in white, the lines of cut that had been previously wrote in black for dividing the rods. This will produce an image with the rod considered closed and detached from everything.

   The code that implements this has a test on the position of one of the 2 points for checking if the line now taken into account is related to the rod now considered.

```
for pair in defectPairs:
    if(abs(cv2.pointPolygonTest(cnt,tuple(pair[1]),True)) < 4):
        cv2.line(imageForSingleRods, tuple(pair[0]), tuple(pair[1]), 255, 2)
```



The output is an image with the rod closed and detached from everything.

Calling now the find contours function will finally create the contours and the hierarchy for the rods now taken into account.

```
 _, rodsContours, rodsHierarcy =
cv2.findContours(imageForSingleRods, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
rodsContours = np.array(rodsContours)
finalContours.append(rodsContours)
finalHierarcy.append(rodsHierarcy)
```

A list called finalContours and finalHierarchy will handle the contours and the hierarchy related to every rod.

Last step: extracting rods metadata and dealing with distractors

From the finalContours and finalHierarchy lists provided from the previous step the metadata requested by the requirements sheet can now be computed.

The test on finalHierarchy[i][0][j][3] == -1 determine if the contour is an external contour of a rod (or of a distractor) or the internal contour of a rod, that is a hole.

In case a possible rod is identified, the algorithm starts.

```
for i, rodsContours in enumerate(finalContours):
        for j, cnt in enumerate(rodsContours):
            if (finalHierarchy[i][0][j][3] == -1):
```

1) **Computing number of holes for extracting type of rod and filtering screws**

   For the extraction of the hole's number in the shape, instead of computing the Euler number, is possible to rely on the hierarchy available: the number of contour "sons" of the actual shape will be the number of holes of the actual shape.

   The rod will be of type A or B depending on the number of holes and is also possible to recognize if the shape is a screw (it will have no holes).

```
holesCount = np.sum(finalHierarchy[i][0][:, 3] != -1)
```

2) **Computing center**

   The center of the rod is computed with the help of the moments function of opencv

```
M = cv2.moments(cnt)
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

3) **Computing perimeter and compactness for filtering washers**

   After the computation of the perimeter, the compactness of the shape is computed. The formula used for the compactness is C = P^2/A and there is no need to use the haralick's circularity (that is more complex and require more computations).

```
perimeter = cv2.arcLength(cnt, True)
compactness = (perimeter ** 2) / area
```

   if the compactness is greater than 20 (a perfect circle has 4π) the shape is a rod and not a washer.

4) Length, width, and orientation of the rod

The length and the width of the rods should be calculated using the MER's major axis and minor axis. Is possible to take advantage of the fitEllipse function of opencv to compute them in an easy way.

```
(x, y), (width, length), angle = cv2.fitEllipse(cnt)
```

5) Width of the rod at the barycenter

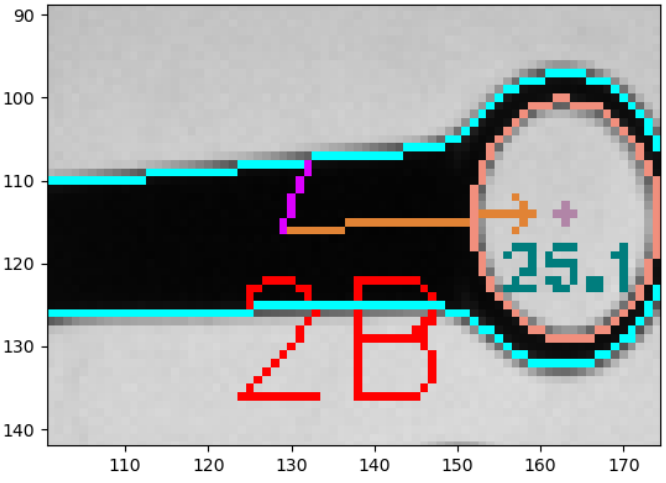
The width of the rod at the barycenter can be computed finding the nearest contour pixel of the barycenter pixel. This is possible from the moment that in the last findCountour, the approximation was not used and now, all the contour pixels are available (see example below).

```
nearestPoint = cnt[np.argmin([euclidean(c, (cx, cy)) for c in cnt])]
finalImage = cv2.line(finalImage, (cx, cy), tuple(nearestPoint[0]), (220,0,0))
rodsDict['width_on_center'] = euclidean(tuple(nearestPoint[0]), (cx, cy)) * 2
```

Of course, the width found is between the barycenter and the nearest contour pixel and must be multiplied by 2.

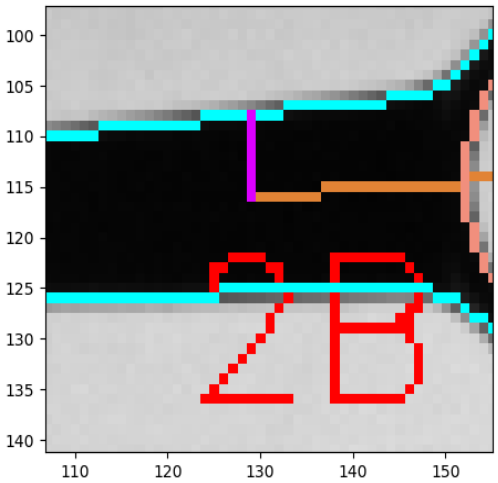
Result with CHAIN_APPROX_SIMPLE

Nearest pixel available is not the real nearest one



Result with CHAIN_APPROX_NONE

Nearest pixel correctly found



6) Holes positions and diameter

The way in which the holes of a shape are found with the use of the hierarchy had been extensively covered. The computation of the center of the hole is done with the moments method explained for the rods. The computation of the diameter is done with the formula below for taking advantage of the contourArea function of openCV.

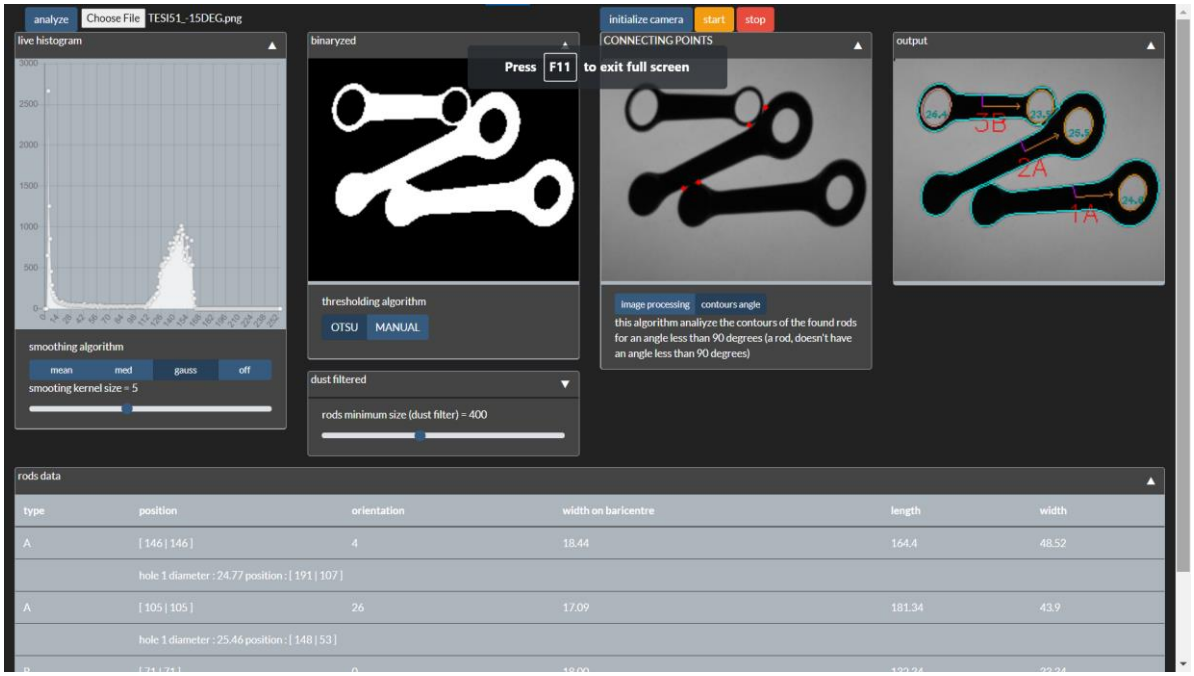
```
holeArea = cv2.contourArea(hole_contour)
equi_diameter = np.sqrt(4 * holeArea / np.pi)
```

Extra: Distribution and user Interface

The program, initially developed as a script in jupyther-notebook, has been wrapped into a standalone eel python gui program. The gui has been developed in web technologies for taking advantage of html-css template availability. The python script with the eel package offers the gui with a chrome browser (a graceful fallback to microsoft-edge is handled in case of windows 10 and chrome not installed). The logic of the gui is to show the pipeline process from left to rigth and give the possibility the change some settings on some specific parameters.

The gui allow to analyze a file with the choose file button (the blue button “analyze” allow to recompute the same image, useful to see different output after the changing of a parameter).

Is also provided the possibility to use the camera: click on initialize camera and after the initialization, and only after the initialization, (usually the camera led on the notebook turn on) press start.



References and attachments

[1] : Extracting countours with findCountours function in opencv https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

[2]: Why not using the erosion to remove dust?
attachment 1

[3]: Finding contours on a non-filtered image lead to a wrong attachment point detection
attachment 2