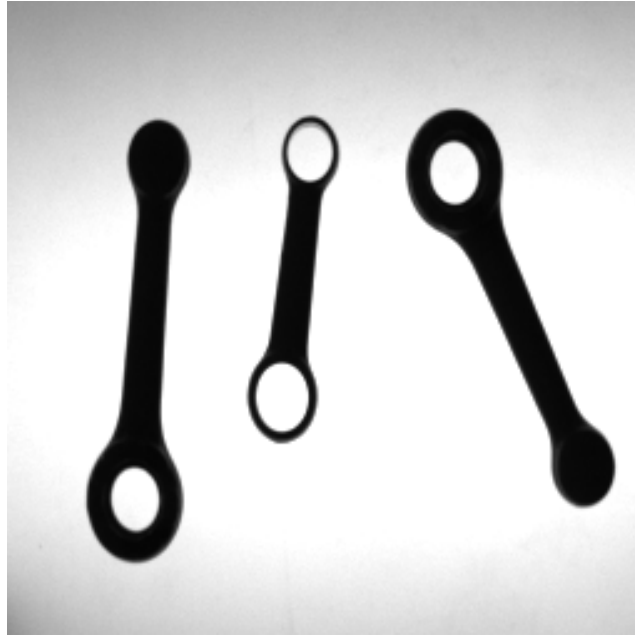


Visual Inspection of Motorcycle Connecting Rods

Tonioni Alessio - Matricola: 0000707766



Requirements

The purpose of this project is to develop a software application for the visual analysis of photos of motorcycle connecting rods obtained with the backlighting technique.

From a photo containing more than one rod the system must be able to compute for each one:

1. Type. → A if it contains only one hole, B if it contains two.
2. Position and orientation.
3. Length, width and width at the barycenter.
4. For each hole, position of the centre and diameter size.

The project is divided into two phase:

- In the first one the images will feature only well separated rods and there won't be others disturbing elements or noise, however the system should be robust to light intensity changes.
- In the second phase there could be in the same picture rods and other elements as well as noise due to the presence of scattered iron powder in the working plane, last but not least the rods could touch each other but won't overlap.

In both these scenarios the system must deliver the informations defined above in a reasonable amount of time and without sensible errors.

Implementation

The final implementation is a simple console application developed in **c++** using the open source library **opencv** (version 3.0.0) for the image processing algorithms. The output images are visualized using the opencv high-gui API and the statistics for each rod are printed on the console.

Phase one

For the first phase it can be used this elaboration pipeline:

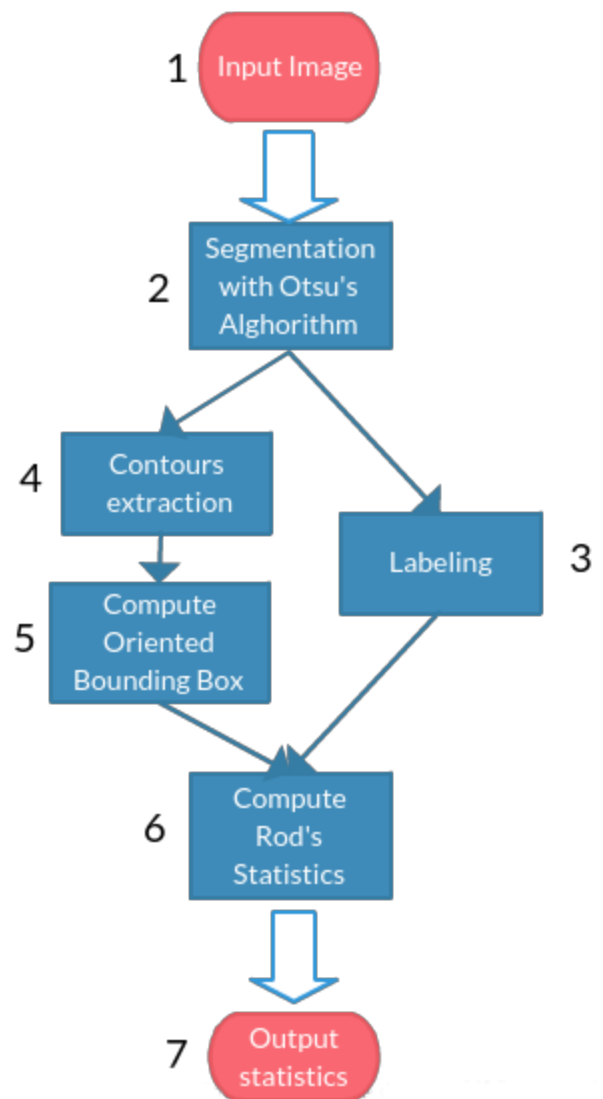
1. The input image is loaded and converted into grayscale for efficiency reasons.
2. A **segmentation** of the input image is performed to separate the pixels of the foreground objects (dark) from the ones of the background (bright).

A first implementation of this step used a simple “*intensity thresholding*” with a fixed threshold, however this approach did not work very well in the presence of a changing brightness.

To gain robustness an automatic thresholding algorithm is used and specifically the “*Otsu’s Algorithm*”[\[1\]](#) both because it is one of the most robust to changes and because a working implementation is already present in the `threshold` function of opencv. The performances of this step are slightly slower using this algorithm but the robustness is drastically improved.

The output from this phase is a binarized image where the foreground object pixels are identified with white (255 in gray scale) and the background ones with black (0 in gray scale).

3. A standard **labeling** algorithm is performed on the binarized image to count and label the connected components. If everything is correct the number of connected components should correspond to the number of rods in the image. As a side effect the image, after normalization, can be used as base image to display the final result of the system: each rod will have a different shade of grey in order to be easily identifiable. To perform this step it is used the `connectedComponentsWithStats`



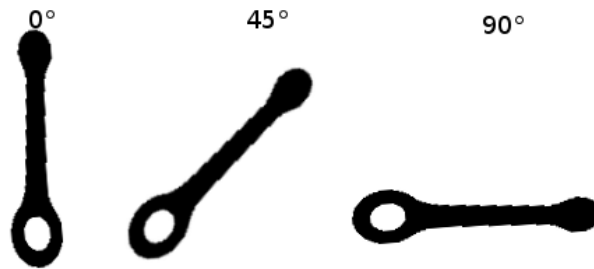
function of opencv, that computes some useful informations on the blob such as its area, its barycenter and a axis oriented enclosing rectangle while labeling the components. To perform the labeling an eight connectivity was preferred to better capture circular shapes.

4. In this step the `findContours` function of opencv is used in order to **extract the contours** and some useful informations from the binarized image. This function compute all the contours of the foreground blobs and organize them in a hierarchical tree structure that later will be used to easily associate each hole, identified in this step as a contours with a father and without a son, with its rod. The function uses the algorithm [2] and some approximation to efficiently handle the representation of contours points.
5. From the contours extracted at the previous step it is possible to **compute the rotated minimum enclosing rectangle** of the shape using the function `minAreaRect`. The output of this phase is a c++ `vector` containing for each contour the `RotatedRect` that enclose it, characterized by its orientation, its width, its length, its center and the coordinates of its four vertices. The computation of the bounding box is executed for all the contours extracted at the previous point, for both rods and circles.
6. The **information** coming from step 3 and 5 are now **merged** in order to properly create an array of `Rods`, a c++ object that encapsulate all the information requested from the system specifications. In order to do so, each labeled component must be associated with its bounding box. This is done by searching the minimum distances between the connected component barycenter and the centers of the `RotatedRects`; once found there's a direct correspondence with one of the contours computed at the step 4 that can be associated with its labeled component.

The hierarchical tree structure computed at point 4 is used to find the contours associated with holes inside this blob: if a contour has no sons and has as parent the outline associated with the current rod, it can be marked as one of its holes. Given the stringent specifications of the problem no coherence check is performed on the contours of the holes to check if it is really a circumference, however if the specifications will change, simple checks deployable are the form factor or the Haralick circularity[3].

Now all the geometrical information on the rods are computed:

- a. the **type** of the rods can be determined counting the number of holes associated with it.
- b. the **position** is obtained from the barycenter computed at the step 3 for each blob.
- c. the **orientation**, the **length** and the **width** are all obtained from the rotated bounding box associated with the blob. The orientation is expressed in degrees(modulo π) following this schema.

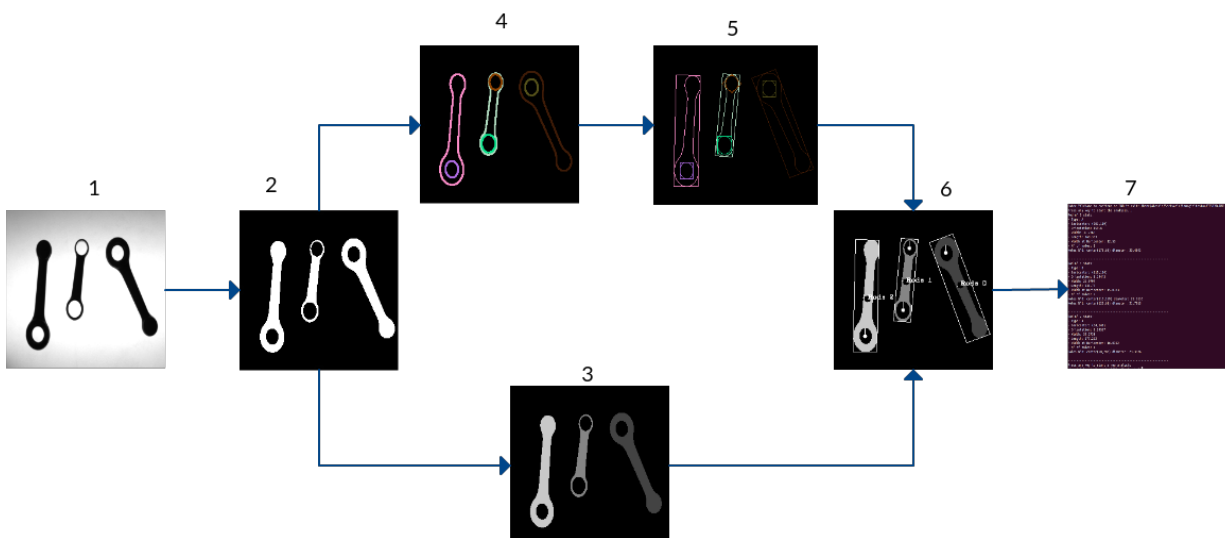


A more robust implementation to determine the orientation of the rods can rely on moments instead of using the one associated with the bounding rectangle and is already implemented as a method in the rods object.

- d. the **width at the barycenter** is computed from the equation of a straight line passing through the barycenter and perpendicular to the major axis of the rod. Its angular coefficient must be $mB = -1/mA$ where mA is the angular coefficient of the major axis. Once computed the proper equation the line is scanned starting from the barycenter in both directions searching for the first *edge changes*: places where the pixel intensity changes from 255(foreground) to 0(background). Once founded, the euclidean distance between them is computed and saved as the width at barycenter. Given this two extreme points it is also possible to later draw this line in the final output image.
- e. the **center** of the holes is approximated with the barycenter of the contours of the hole computed at the step 4, the **radius** is computed averaging the euclidean distance between the barycenter and all the contours point.

After this computation all the informations for each rod are printed on the console and it is displayed a summary image where each object is labeled with a progressive number and it is identified by a different shade of grey. In addition all the geometrical constructions(rectangles, barycenters, radius, widths at barycenter) are drawn to have the informations at a glance.

This graphics shows the image evolution during the various pipeline stages



This is an example of the console output for a rod:

```
Rod n° 0 stats:  
- Type: A  
- Baricenter: (201,119)  
- Orientation: 159.6  
- Width: 39.7763  
- Length: 169.353  
- Width at Baricenter: 12.53  
- N° of holes: 1  
Holes N°0: center(178,68) diameter: 23.4843
```

With this pipeline the analysis of a single image takes approximately 4-5ms on a Intel® Core™ i5 CPU M 430@ 2.27GHz from just after the loading of the image till when the system has printed the statistics on the screen. However the first run is a lot slower, ~100ms, probably due to the loading of the external opencv modules and functions used.

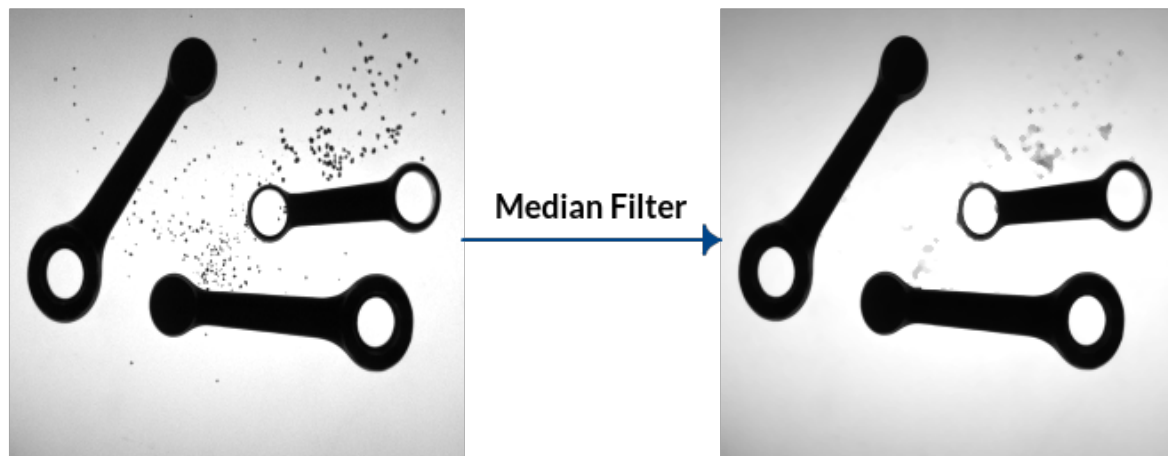
Phase Two

In order to be able to fulfill all the requirements above with image that present some degrees of noise or distractor elements some changes must be made to the elaboration pipeline.

Inspection area dirty due to the presence of scattered iron powder:

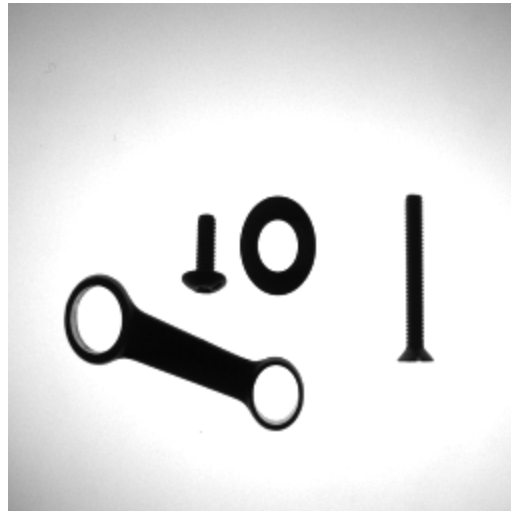
The background of the image can be uneven due to the presence of scattered iron powder that create a “*salt and pepper*”-like noise. A fast way to get rid of this type of noisy pixels with little loss of quality on the source image is a nonlinear **median filter** applied before the segmentation step.

After some experiments with the noisy images given, three iterations with a smaller kernel were preferred against one single iteration with a bigger kernel in order to maintain the details of the rods that would otherwise be lost. For example a bigger kernel will spoil the thin contours of the two holed rods.



Most of the noisy pixels after the filter are completely blended with the background, but some of them remains slightly darker, however they will be above the automatic threshold of the segmentation step and thus will be discarded.

Images may contain other objects that need not to be analysed by the system:



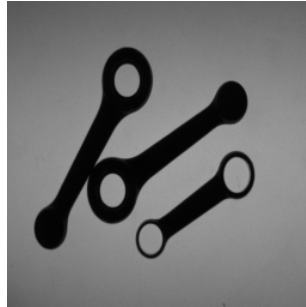
The image can now contains three different type of object: **rods**, **screws** and **washers**: the system must be able to analyze the rods while ignoring the distractors objects.

This can be achieved by adding a series of coherence checks in the sixth step of the elaboration, the distractor elements will be extracted as blob from the labeling algorithm but they will not be cataloged as rods in this step.

To ignore the **screws** it is sufficient to check if the connected component has an hole inside of its contours or not, in the latter case it can be skipped.

The **washer**, instead, has an hole inside, but it has a shape completely different from the one of a rod and in particular its oriented enclosing rectangle will have nearly the same width and height (if there was no noise it would have been a rotated square). To ignore this category of objects is thus sufficient to check the two dimension (width and height) of the `RotatedRect` associated with the labeled item; if the absolute difference between the two is below a certainly static defined threshold the item is not a rod and thus it can be skipped.

Rods can have contact points but do not overlap one to another

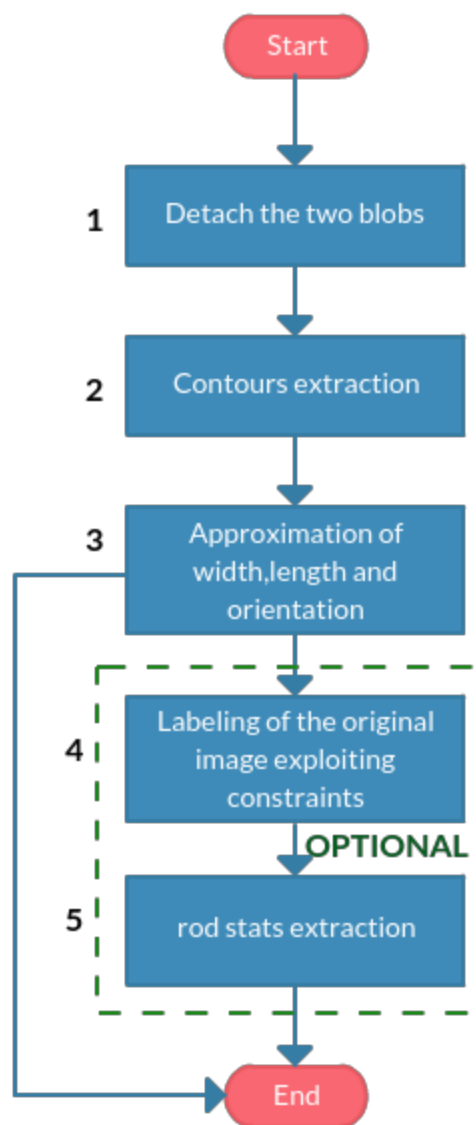


With the elaboration pipeline used so far this problem can not be solved: neither the labeling algorithm nor the contours extractor are able to distinguish two or more connected rods. However it's possible to recognize if a region of the image contains more than one rod because the area of the blob associated with it will be bigger than the one associated with a normal single rod region. Exploiting this simple test a new analysis pipeline can be deployed to further analyze this difficult area. Even this can be decomposed in two parts: one mandatory at the end of which it is possible to have approximated results, and an optional one whose aim is to further refine the partial result especially on the computed width and length of the rods. This is useful from a performance point of view because the refinement part tend to make the system a lot slower.

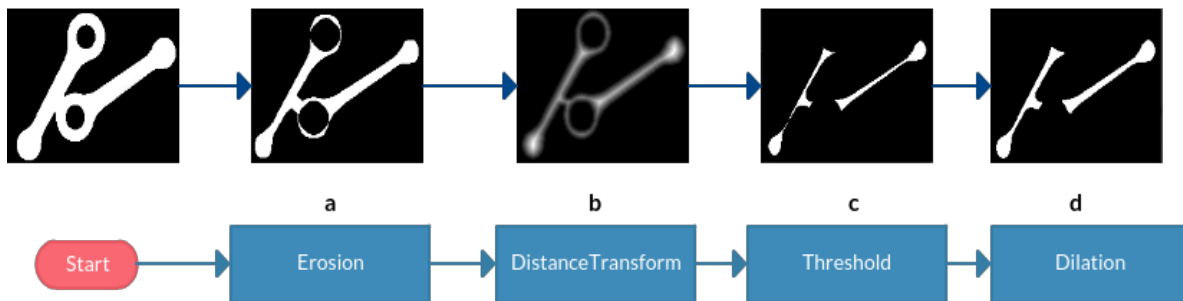
For all this elaboration were used only blob analysis technique in order to keep the system simpler and faster. This elaboration pipeline works after the labeling of the original image and **take as input: the ROI of the thresholded image that contains the touching rods and an array containing the contours associated with the holes** extracted at the 4th step of the original pipeline.

Each point of the pipeline will be analyzed:

1. This is the most tricky part of the whole analysis: the two rods are connected and there isn't a function to identificate where the contact point is , however they must be



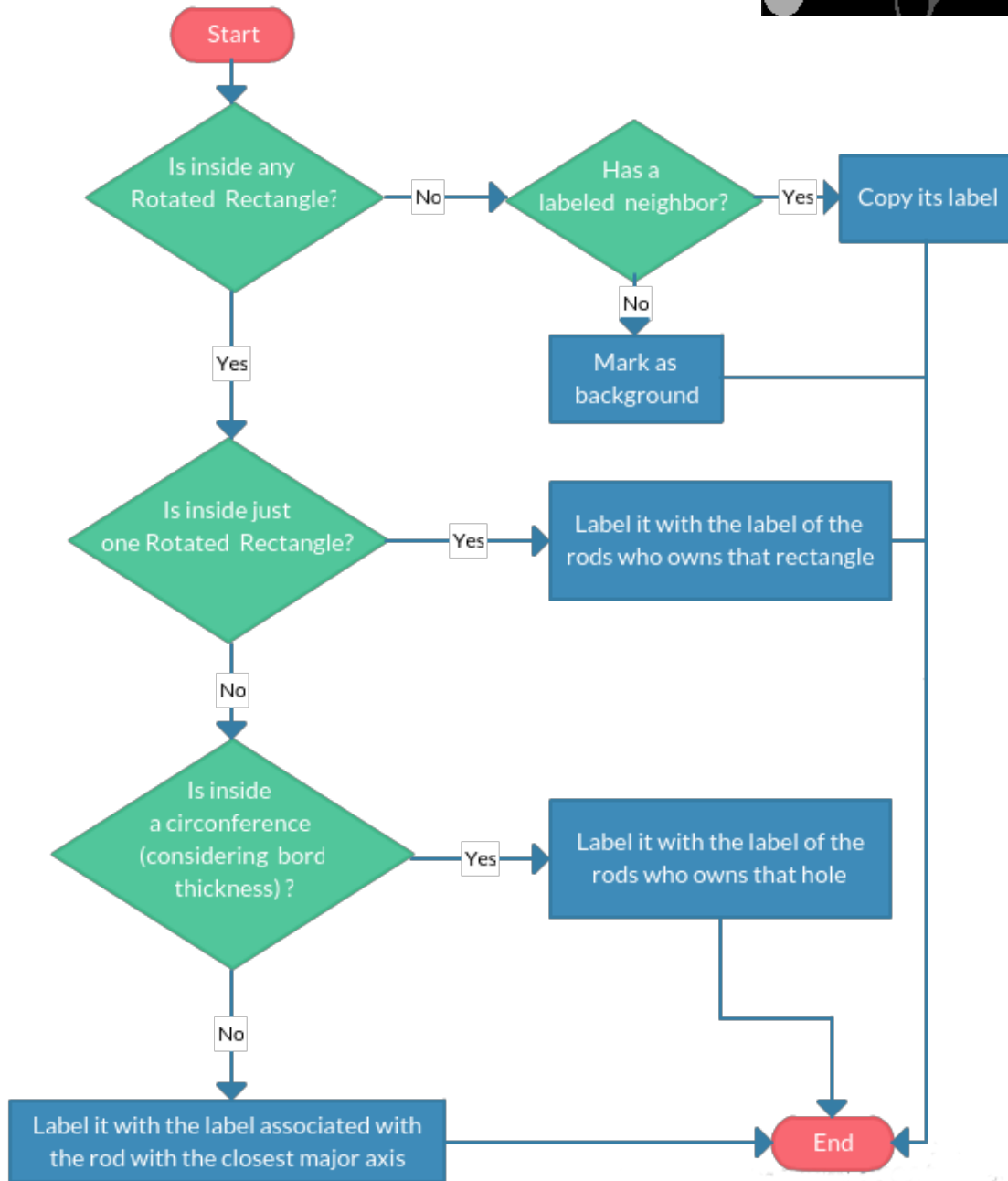
detached in some way in order to count and identificate them. A series of passages are deployed to do that: first of all an erosion with a 3x3 kernel is executed to reduce the size of the rods bodies and the contact area between them(**a**). Then it is used the `distanceTransform` function of opencv which uses the algorithm [4] to substitute each pixel of the foreground object with its L2-distance from a background pixel (**b**). The resulting image will have peaks of intensity in the central parts of the rod's bodies, then using a normalization between 0 and 1 followed by a threshold at 0.5 is possible to keep only the central parts of the rods detaching them (**c**). Eventually a dilation can now be applied to have bigger rods bodies that can be better analysed later for extracting useful informations (**d**). At the end of this step the result is a binary image that exhibits only the main bodies of the rods: the holes and the contact point are eliminated in order to make it easier to count how many objects there are and where they are placed.



2. Having the rods bodies, their contours can now be extracted with the `findContours` function in order to compute the barycenters, the bounding boxes and thus the orientation associated with each blob. The first and the last are fundamental to compute an approximation of the same stats of the original rods.
3. The statistics associated with each rod can now be approximated leveraging on the barycenter position, the major axis orientation and the holes position are passed to the function as argument; even if the rods were touching, the `findContours` function performed on the original image in the main pipeline was able to correctly identify inner contours associated with holes as well as their center and radius.

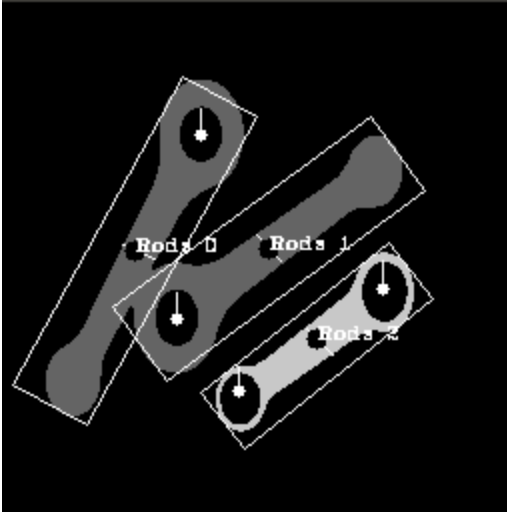
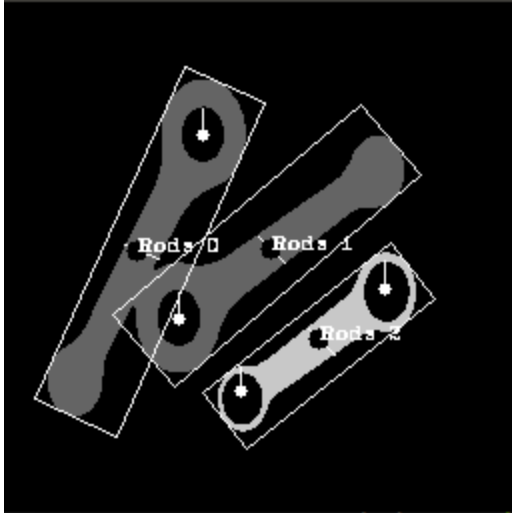
First of all each rod is associated with its holes: the ones with the center at a distance smaller than a threshold from its major axis. Then for each hole the thickness of its border is computed searching for points at three “*edge changes*” starting from the hole center along the major and minor axis directions. This information together with the ones extracted so far are used to determine an approximation of the width and length of the rod and to build an enclosing rotated rectangle that contains it. After this step the system can stop, all the rods stats are computed even if with some approximations.

4. If more precise measurements are needed two additional steps are performed. Firstly all the pixel of the original (pre-detaching) thresholded image are labeled leveraging on the approximated measurements computed so far following this algorithm.



5. Once the above step is done an image with semantically labeled rods is produced and it can be analyzed with techniques similar to the one used in the main elaboration pipeline in order to extract more precise stats for each rod.

Here is a quick comparison between the results obtained after step three and going all the way through the additional two steps.

Fast (Stop at 3rd step)	Slow (All the steps)
	
<p>Rod n° 0 stats:</p> <ul style="list-style-type: none">- Type: A- Baricenter: (67,124)- Orientation: 28.8108- Width: 42- Length: 176.193- Width at Baricenter: 17.4642- N° of holes: 1 <p>Holes N°0: center(100,66) diameter: 25.6061</p> <p>Time: 6.667 ms</p>	<p>Rod n° 0 stats:</p> <ul style="list-style-type: none">- Type: A- Baricenter: (67,124)- Orientation: 24.2093- Width: 44.3159- Length: 182.064- Width at Baricenter: 18.3848- N° of holes: 1 <p>Holes N°0: center(100,66) diameter: 25.6061</p> <p>Time: 48.12 ms</p>

The differences are quite significant, but so is the slowdown of the system (~30ms slower). However this happens only if the image presents touching rods, for normal images the elaboration is the same as before and so the time remains the same as the one presented at the end of the first phase, the median filter and the additional tests hasn't introduced significant delays.

Given that the rods can only have two predictable shapes, a completely different approach to the solution of the touching rods problem can rely on **shape recognition**. However this technique was not thorough because it seems to be heavier to implement and deploy, having to deal with images that can present rods scaled or rotated compared to the model image.

Bibliography

1. Otsu, Nobuyuki. "A threshold selection method from gray-level histograms." *Automatica* 11.285-296 (1975): 23-27.
2. Suzuki, Satoshi. "Topological structural analysis of digitized binary images by border following." *Computer Vision, Graphics, and Image Processing* 30.1 (1985): 32-46.
3. Haralick, Robert M. "A measure for circularity of digital figures." *Systems, Man and Cybernetics, IEEE Transactions on* 4 (1974): 394-396.
4. Felzenszwalb, Pedro, and Daniel Huttenlocher. "Distance transforms of sampled functions." 1 Sep. 2004.