Visual Inspection of Motorcycle Connecting Rods

Antonio Morelli, antonio.morelli3@studio.unibo.it

***Abstract*— This report describes the work process followed to develop a computer vision system capable of analyzing the dimensions of two different types of connecting rods to allow a vision-guided robot to pick and sort the rods according to type and size, a project proposed for the exam of *Image Processing Computer Vision* of *University of Bologna's Artificial Intelligence Master Degree*. The problem is solved through the employment of a very well-known Python library in the computer vision and image processing field, *OpenCV*; the final product consists of a *jupyter notebook* that contains the commented code which solves all the four tasks proposed by the outline of the project and of a GUI application that allows to inspect single images.**

***Keywords*— Pick-and-place, Blob Analysis, Python, OpenCV**

I. Introduction

Pick-and-place is one of the most comprehensive tasks that can be required of a robotic arm; the ability to correctly classify the objects to be picked is useful not only for the picking itself, but possibly also for splitting the objects according to future needs. To ensure that this task is performed correctly, a careful analysis of the objects is compulsorily needed; here we will see a bunch of techniques aimed to extract information from pictures containing motorcycle connecting rods.
Pictures are taken by means of backlighting technique, one of the most common and useful lighting techniques, typically used for the establishment of objects' orientation/location, particularly when coupled with pick-and-place or vision-guided robotics applications; the primary source of light is behind the objects, so the objects appear darker than the background and more prone to be binarized and then analyzed.

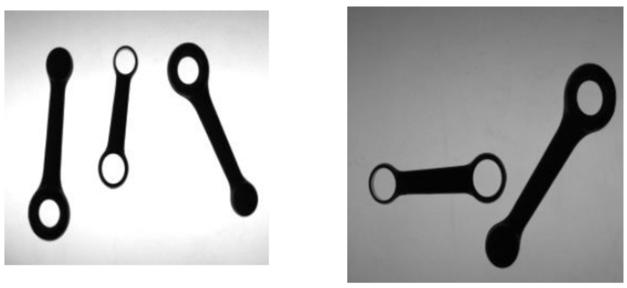

Fig. 1
Two exemplar working images.

For each of the connecting rods appearing in a picture, it is required to provide its type, A or B, according to the number of holes that it contains, its position and orientation, its length, width and width at the barycenter; for each internal hole the requirements are centre position and diameter size instead.
Three slight changes of the original task were proposed and solved:

- The first change causes the presence of distractors objects (i.e. screws and washers) in the images.
- The second change allows contact points between rods, without overlapping.
- The third change consists in the presence of scattered iron powder across the inspection area.

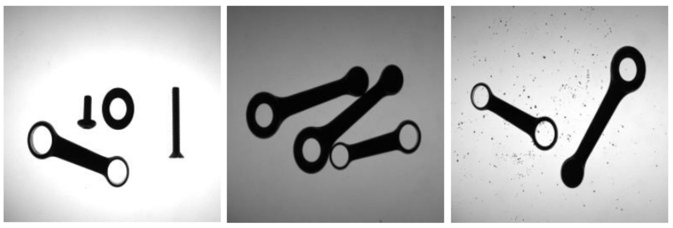

Fig. 2
Exemplar working images with changes.

The provided images are taken in close-word settings and this is taken in when making solution choices.

II. First Task

The seeking for a solution is conducted according to the typical feature extraction workflow on images that contains darker objects and a brighter background, in which the images are first binarized, then connected components are labeled and finally analyzed.

A. Binarization

The binarization task consists in the segmentation of image pixels into two disjoint regions corresponding to foreground and background; this is trivially achieved by applying a plain tresholding operator $\mathbf{T}$ if the gray-level histogram associated with the image appears to be clearly bimodal, but any change in lighting conditions can cause a failure.
Even if working with a close-word assumption, for flexibility reasons, the system should not need any change to work properly with lighting sources of different power. An automatic threshold selection algorithm is used, the Otsu's algorithm [1]. The key idea of this algorithm is to choose a threshold that maximizes (minimize) the between-group (within-group) Variance of resulting regions (the algorithm can be extended to more than two disjoint regions).
The images are first smoothed with a *median blur* filter; this is because some images were showing thin edges around the holes, resulting in gaps during thresholding and later failures when labeling connected components. The main advantage of median filter is that it can blur and reduce the noise levels while retains edges; in our case, the filter strengths the circles edges and allows to perform a better binarization.

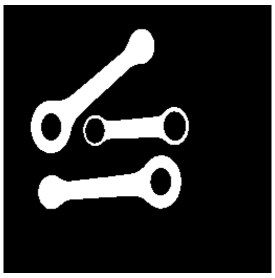

Fig. 3
EXAMPLE OF NEGATIVE BINARIZATION WITH OTSU'S ALGORITHM.

B. Labeling

1) findContours: Connected components labeling with OpenCV is mostly carried out using the function `connectedComponents`. However, the fact that it is asked to both analyze external rods and internal holes brought me to the decision of using `findContours` instead. This function finds the curves joining continuous boundary points having same intensity using the algorithm [2]. It can find both interior and exterior contours, and return the results in a variety of formats; given the parameter `RETR_TREE`, it retrieves all of the contours in the image and reconstructs a full hierarchy of nested contours, so that we can quantize how many holes are contained by a rod. Moreover, the execution times of former function depend more on the image size, while the ones of the latter are highly dependent on blobs sizes, which are relatively small in our case.
OpenCV's binarization works in such a manner that if the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to 255. On the other hand, `findContours` finds white object on a black background, so we perform the function over the negative of the thresholded images (i.e., `255-thresh`) as we can see in *Fig. 3*.

2) Hierarchy tree: Along with a list of objects contours, `findContours` with `RETR_TREE` parameter returns a *hierarchy* tree, a three-dimensional `NumPy` array, with one row, X columns, and a depth of four. The X columns correspond to the number of contours found by the function. Column zero corresponds to the first contour, column one to the second, and so on. Each of the columns has a four-element array of integers, representing indices of other contours, according to this scheme:

$$[next, previous, first\,child, parent]$$

Thus, if *first child* is equal to -1, the contour must belong to an internal hole while, if it has no parent, then it must be the one of a rod.

C. Analysis

1) Hole: For each internal hole, we return its list of contour points and an ellipse that fits (in a least-squares sense) the contour best of all, using `fitEllipse`; the ellipse gives us the two diameters values of the hole and its centre.
At first, I was trying to find the *Minimum Enclosing Circle* by means of `minEnclosingCircle`, but after some trials I decided to switch to the other function, since it was visually fitting them better.
Finally, the function `pointPolygonTest` tells us which rod the hole belongs to, testing if the contour of the latter is circumscribed by the one of the former. The contours are analyzed in hierarchical order, so it is guaranteed that a rod contour is analyzed before its internal holes. Each rod has an integer variable `type` which is initialized to 0 and then increased for each contained hole. This is feasible since the components are not overlapping.

2) Rod: For each rod found, we compute its *Minimum Enclosing Rectangle*(*MER*) using the function `minAreaRect`, its moments using `moments`, its list of contours point and again its best fitting ellipse with `fitEllipse`.
The MER is used to establish the component's position, width and lenght, the moments to retrieve its barycenter, the contours point list to establish its center width, which is the euclidean distance from the centre to the nearest contour point times two. Lastly, the orientation is computed according to the ellipse, since it seemed to give more accurate results than the one computed with `minAreaRect`.

3) Drawing: After finding all the requested features, it is now time to highlight them. A previous assignment of an `rgb` value allows to distinguish the rods. Its major and minor axis are drawn too, but this only after that the angle value, which is returned with respect to the *y-axis* by `fitEllipse` function, is rotated to be expressed according to the *x-axis*. Along with the output, a *.csv* report of the analysis is returned.

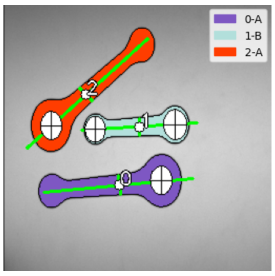

Fig. 4
ANALYSIS OUTPUT.

III. SECOND TASK

While still meeting the requirement of the first task, we were asked to modify the system in order to deal with one (or more) of the three previously seen changes in the characteristics of the working images. All the tasks are solved within the `component_analysis` function.

A. Change 1

Again, we exploit the hierarchy of contours to solve this problem.

1) Washers: If the distractor is a washer then, when subjected to the `if` statement that controls if it has inner children, it would be wrongly marked as a rod, since all the washers contain another hole inside; but a washer is way more circular than a rod, so thresholding over its *circularity* value allows to avoid their analysis.

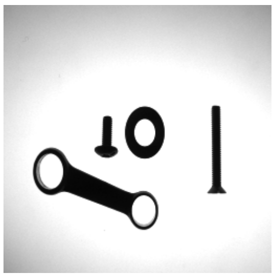

Fig. 5
EXAMPLE OF FIRST CHANGE.

2) Screws: If the distractor is a screw then, when subjected to the `if` statement that controls if it has inner children, it would be wrongly marked as an internal hole, since they do not contain any internal contour; but a screw is less circular than an internal hole, so we do the same and opposite reasoning as the one made for washers.

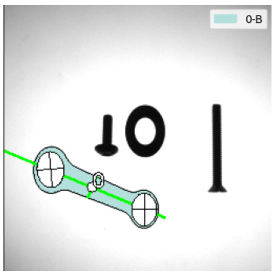

Fig. 6
ANALYSIS OUTPUT WITH FIRST CHANGE.

B. Change 2

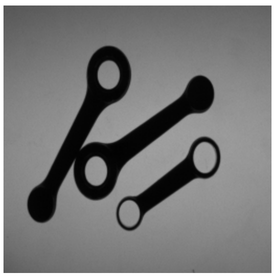

Fig. 7
EXAMPLE OF SECOND CHANGE.

This change involves the introduction of contact points between rods. My first intuition was to use some kind of *morphological operator* to get rid of touching parts and correctly label different regions, but this was corrupting the thin edges of some rods' holes, so I ended up using another method which searches for convexity defects points of blobs' *convex hull*.

1) Convex Hull: A convex object is one with no interior angles greater than 180 degrees. A shape that is not convex is called *non-convex* or *concave*. Hull stands for the exterior of the object. Therefore, the convex hull of a group of points is a tight fitting convex boundary around them. Any deviation of the object from this hull can be considered as convexity defect.

2) Contact points: Once these convexity defects points are found, what we do is threshold them according to the number of non zero pixels contained on the negative of the binarized image (i.e., the foreground pixels) that lie inside a 5×5 patch taken around the defect point; it is visually perceptible that neighborhoods of contact points contain more non zero points (foreground pixels) than the rest of the possible defects points, so a threshold is chosen according to the value of the 90 percentile of the list of defects points ordered by the number of non zero points (foreground pixels) contained in their neighborhoods.

3) Disjunction segments: Since for every couple of touching rods there are two contact points, we finally seek for couples whose distance is minimum, and draw a line that separates the components. Although this is not a good-looking solution from a visual point of view, it does not affect the features' computation.

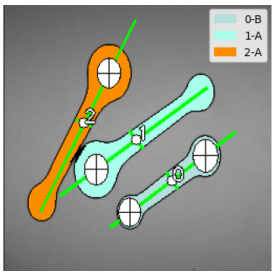

Fig. 8
ANALYSIS OUTPUT WITH SECOND CHANGE.

C. Change 3

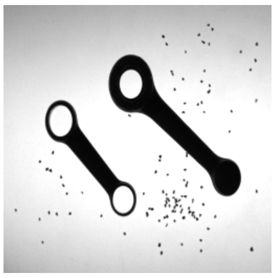

Fig. 9
EXAMPLE OF THIRD CHANGE.

The last change introduces the possibility of scattered iron powder's presence; this looks like a typical *salt and pepper* noise, so I first tried to use morphological operations and a stronger median filter, but it ended up again in corrupting the shape of some rods' thins circles. Thus, I noticed that the contours of specks of dust were pretty small compared to the others, so I simply avoided to analyze small contours.

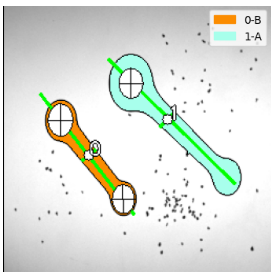

Fig. 10
ANALYSIS OUTPUT WITH THIRD CHANGE.

IV. USAGE

Follows some instructions to run both the notebook and the application.

A. *Requirements*

- Python v. 3.7.x or later. Never tested with previous versions.
- While placed in the project main directory, run the command `pip install -r requirements.txt` to install the necessary dependencies.

B. *Notebook*

The execution of the entire notebook produces the creation of outputs that can be found inside `notebook/outputs`, and one *.csv* report for each task into `notebook`.

C. *GUI Application*

Along with the `jupyter notebook`, it is provided a simple application that allows to analyze single images and export the result. The application is written using the Python library `PySimpleGUI`, which permits fast development but does not seem customizable enough. To perform the analysis, it is highly recommended to choose one folder among `app/images/1/`, `app/images/2/c1`, `app/images/2/c2` and `app/images/2/c3` and to maintain unaltered the radio buttons as the script automatically chooses the right methodology to solve the task according to folder name.

V. CONCLUSION

To ensure the robustness of the system, one should be able to work with a substantially larger set of data; however, the first instance of the problem seem to have been solved successfully, since it should act coherently even with unseen images. The solution of the first change works as long as the distractors maintain the characteristic peculiarities of the washers and screws, but it should not be difficult to extend the reasoning to shapes of different types. Solving the task with the introduction of contacting points was challenging, and the main problem with the solution is that the correct threshold value over non zero count applied to defects points may differ for unseen data. Lastly, the solution to third change seems robust enough, at least until there are no agglomerates of specks of dust whose contours exceed the surfaces of other objects, which is very unlikely.

REFERENCES

[1] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: `10.1109/TSMC.1979.4310076`.

[2] Satoshi Suzuki and KeiichiA be. "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46. ISSN: 0734-189X. DOI: `https://doi.org/10.1016/0734-189X(85)90016-7`. URL: `https://www.sciencedirect.com/science/article/pii/0734189X85900167`.