COMPUTER VISION AND IMAGE PROCESSING PROJECT:

RODS INSPECTION

BALJINDER SINGH BAL

FIRST TASK

TYPE OF ROD

To classify the rods first we binarize the images (find foreground and background regions). Since the images are obtained using backlighting tecniques, the associated graylevel histograms are bimodal and, to choose an appropriate threshold value , the Otsu algorithm has been used. This algorithm allows to obtain a suitable value even when the intensity of the backlighting is variable (due to power changes in the back light).

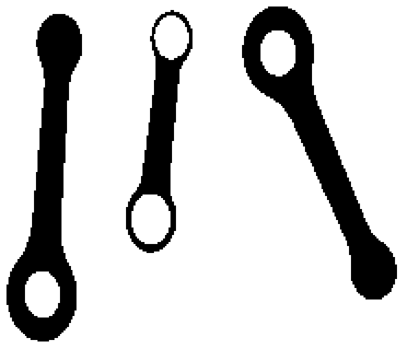


Illustration 1: TESI00.BMP BINARY

Illustration 2: Tesi33.bmp BINARY

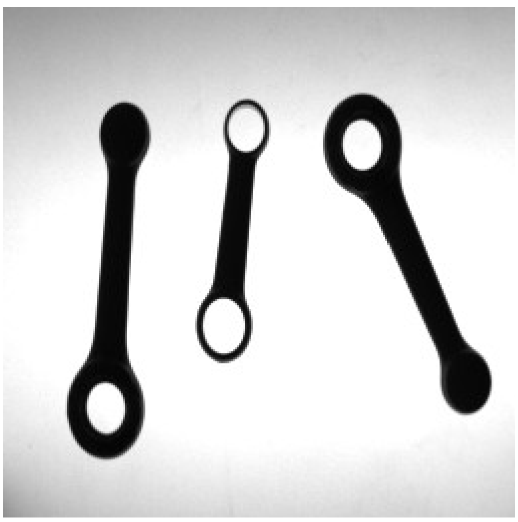
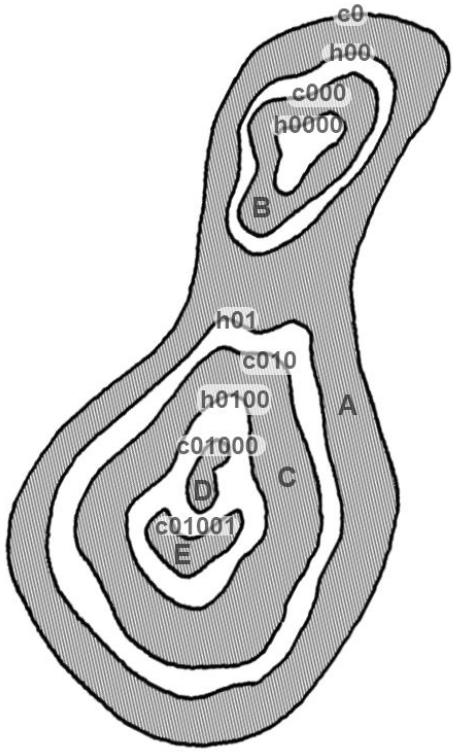


Illustration 4: TESI00.BMP GRAYSCALE

Illustration 3: Tesi33.bmp GRAYSCALE

After the binarization step, the different objects must be detected and classified using their peculiar characteristics(for example different dimensions). Since the two type of rods that must be detected differ from the number of the holes, the OpenCV findContour() function has been used. This function looks for the contours of the different objects(connected components) and classify them in a hierarchical structure that defines each contours position in a tree-structure depending on their relationships to the other contours. So if a contour is contained in an outer contour it is said to be its “child” and the containing contour the “parent”. If on the other hand a contour is at the same level in the hierarchical structure as some other contour then they can be considered “siblings”.



idx	contour
0	c0
1	h00
2	h01
3	c000
4	c010
5	h0000
6	h0100
7	c01000
8	c01001

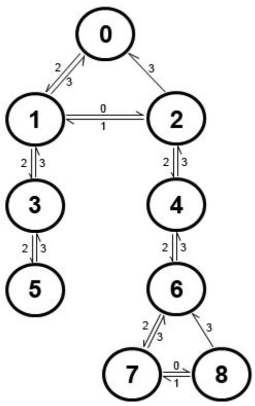


Illustration 5: Hierarchycal structure of the contours returned by findContours() function

The informations regarding the hierarchical structure for each contour are returned in a four element vector for each contour as in the table below .

Meaning of each component in the four-element vector representation of each

node in a contour hierarchy list

Index	Meaning
0	Next contour (same level)
1	Previous contour (same level)
2	First child (next level down)
3	Parent (next level up)

This tree structure has been exploited to classify the two different rods and the holes obtaining a list of the searched objects:

-rods with two holes called ROD B;

-rods with one hole called ROD A;

- the holes belonging to the found rods.

The holes are simply those contours that don't contain themselves other contours(don't have any "child" contour in our terminology):

```
for (int i=0;i<contours.size();i++){
            //check if it has no hole and if its parent is not the first element
            in the hierarchy        vector(to exclude from analysis the objects
            that have no hole(that are the screw)

            if(hierarchy[i][2]==-1  && hierarchy[i][3]!=0 {

                        holes.push_back(i);

            }
```

A contour is considered as a rod ,as said before, depending on the number of holes it has:

```
for(int j=0;j<contours.size();j++){
            //if it has at least one child(one hole contour)
            if(hierarchy[j][2]!=-1){

                        if(hierarchy[hierarchy[j][2]][0]!=-1){

                                    //store the contour as a rod of type A

                                    rodA.push_back(j);
                        }
                        //if its child has no siblings
                        else{
                        //put it in the list of rod B type
                        objects

                                    rodB.push_back(j);
                        }
            }
```

Below we can see a typical result of the classification process.

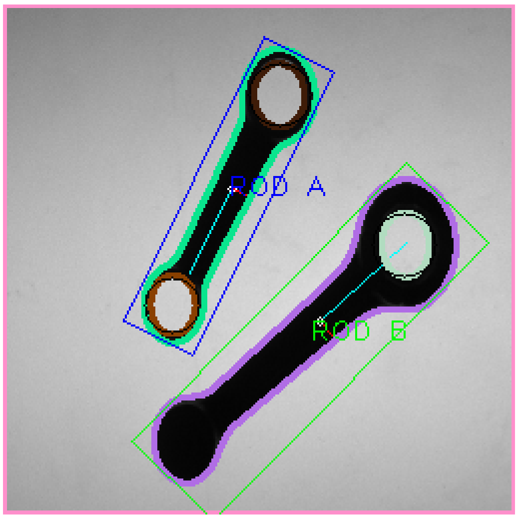


Illustration 6:TESI21.BMP   Typical result of the detection process

POSITION, ORIENTATION, LENGHT AND WIDTH

The simplest way to compute all these features for each rod is to compute the minimum enclosing rectangle. This is easly done with the OpenCV function minAreaRect() that returns a rotated rectangle (rotated along the object orientation) and with the minimum area. This rectangles center and angle are respectively the position and orientation of the object and its height and width the object's length and width.

An alternative way is also provided with the **`computeOrientation()`** and the **`computeBarycenters()`** functions that have been written considering the objects as being a mass distribution(each pixel has unitary mass). In this way the inertia matrix can be computed and the barycenter and orientation extracted .

The barycenters are simply the ratio between the contour moment of order (1,0) and the area (x coordinates) and moment of order (0,1) and the area (y coordinate). In the **`computeBarycenters()`** function we find:

```
for(int j=0;j<rod.size();j++){
                        barycenters[j].x=(m[j].m10)/(m[j].m00);
                        barycenters[j].y=(m[j].m01)/(m[j].m00);

                  }
```

for the value of the orientation we can take the orientation of the major axis using the following relationship

$$\theta = -\frac{1}{2}\arctan\left(\frac{2M'_{1,1}}{\left(M'_{0,2}-M'_{2,0}\right)}\right)$$

in the **`computeOrientation()`** function we therefore find:

```
for(int j=0;j<rodA.size();j++){
                        thetaA[j]=
                  -0.5*atan((2*mA[j].mu11)/(mA[j].mu02-mA[j].mu20));
```

WIDTH AT THE BARYCENTER

This value corrisponds to the distance between the two points of the object's contour that intersect the minor axis (through the barycenter). In order to find such intersections, the contour's points have been rotated as to to see them from an eigenvectors reference system (centered on the barycenter). Of all the rotated points,to compute the width along the minor axis (through the barycenter), it is sufficient to locate the two points that have zero as their coordinate along the major axis. We find these calculations in the function `hotellingTrans()`:

```
for(int l=0;l<rod.size();l++){

            vector<Point2i> temp(contours[rod[l]].size());
            for(int j=0;j<contours[rod[l]].size();j++){
                        vector<double> diff(contours[rod[l]].size());
                        Point2i centered = 
                  (Point2i)contours[rod[l]][j]- (Point2i)bar[l];
                        temp[j].x= 
eigen_vec[l].at<double>(0,0)*centered.x+eigen_vec[l].at<double>(0,1)*centered.y;


                        temp[j].y= 
eigen_vec[l].at<double>(1,0)*centered.x+eigen_vec[l].at<double>(1,1)*centered.y;
```

where the temp[] vector stores simply the rotated points.
Find the intersections:

```
if((int)temp[s].x==0){
            candidates[l].push_back(temp[s]);
}
```

In the image below we can see a type A rod rotated and seen from the eigenvectors reference system.

Illustration 8: TESI21.BMP

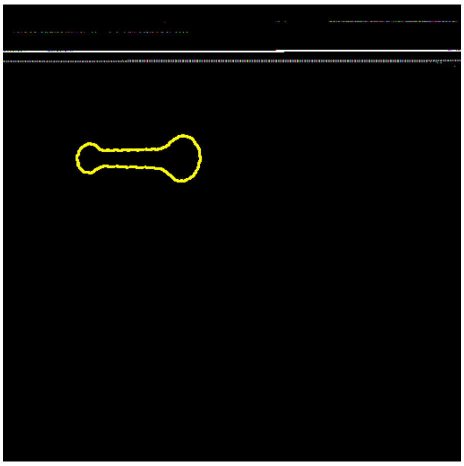


Illustration 7: Rotated contour of the type A rod found in TESI21.BMP

CENTER AND DIAMETER OF THE HOLES

As for the rods, here too is simpler to find a minimum enclosing geometrical figure, but instead of a rectangle a circle can better approximate the hole. The proper OpenCV function is `minEnclosingCircle()` that returns the looked for circle whose center and diameter are the sought features.

RESULT’S TABLE

A table is displayed to show all the requested informations.

```
Total number of rod found: 3
*****************************************
         instance  position        (lenght x height)          orientation     mytheta     width at the barycenter

Touching Rod found:
Rod A:        0    [175, 166]              [113 x 40]            -18            -17          20

Touching Rod found:
Rod B:        0    [92, 105]               [154 x 54]            -25            -20          21

Touching Rod found:
Rod B:        1    [140, 128]              [166 x 49]            -45            -41           19


Total number of holes found: 4
*******************************************
         instance  position     diameter
Hole :        0    [132, 176]      25

Hole :        1    [99, 169]      28

Hole :        2    [214, 154]      29

Hole :        3    [44, 128]      27
```

Illustration 9: Table of results

SECOND TASK

PRESENCE OF DIFFERENT OBJECTS("DISTRACTORS")

Some images contain objects different than the rods(screws and washers). To not consider them in our detection of the rods it can simply noted that the screw has a contour with no holes. And the fact that in the hierarchical structure discussed above it has as the "parent" contour(containing contour) the frame of the image (always stored as the first contour) is exploited to not consider it as a rod hole:

```
for (int i=0;i<contours.size();i++){
            //check if it has no hole and if its parent is not the first element
            in the hierarchy        vector(to exclude from analisys the objects
            that have no hole(that are the screw)
             if(hierarchy[i][2]==-1  && hierarchy[i][3]!=0 {
                        holes.push_back(i);
            }
```

It won't be even considered as a rod since it has no holes.

The washers have a hole so they might be wrongly classified as both a rod and a contour. To not consider it as a hole we check the rectangularity of it's father. Indeed the holes of the rods have as their containing contour (father) the outer contour of the rod whose minimum enclosing rectangle is not a square(as in the washer's case):

```
//check the rectangularity of its father(outer contour)
            float rect_holes;
            RotatedRect r=minAreaRect(contours[hierarchy[i][3]]);
            if ((r.size.height/r.size.width)<=1){
                        rect_holes=r.size.height/r.size.width;
            }
            else{
                        rect_holes=r.size.width/r.size.height;
            }
            // check
            //cout<<"rectangularity "<<rect_holes;
            if(rect_holes<0.75){
                        holes.push_back(i);
            }
```

the rect_holes variable for the washers is very close to 1 and for the rods is smaller(0.75 value has been chosen as a limit value by checking thorough the images).

To not consider the washer as a type A rod (a rod with one hole), the same consideration on the rectangularity with a similar code used for the holes:

```
//check the contours rectangularity to cancel out the non rod objects with holes
            RotatedRect s=minAreaRect(contours[j]);
            float rect_ext;
            if ((s.size.height/s.size.width)<=1){
                        rect_ext=s.size.height/s.size.width;
                        }
            else{
                        rect_ext=s.size.width/s.size.height;
                        }
            // check
            //cout<<"rectangularity "<<rect_ext;
            if(rect_ext<0.75){
            //if it passes the check then we can classify it as a rod
                  if(hierarchy[hierarchy[j][2]][0]!=-1){
                        /*classify as either a rod A or B type */
                  }
```
In the images we can see the result.

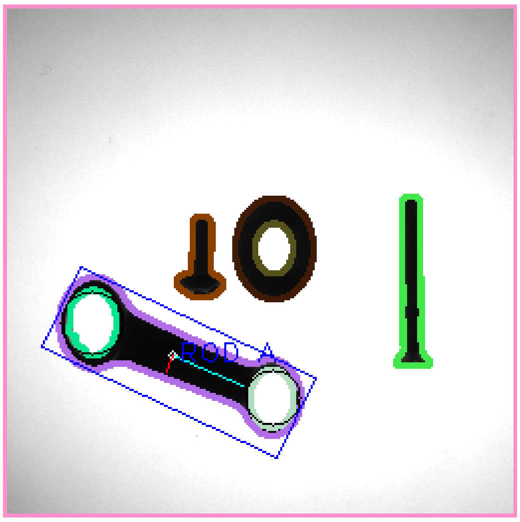


Illustration 11: TESI49.BMP Presence of "distractors"

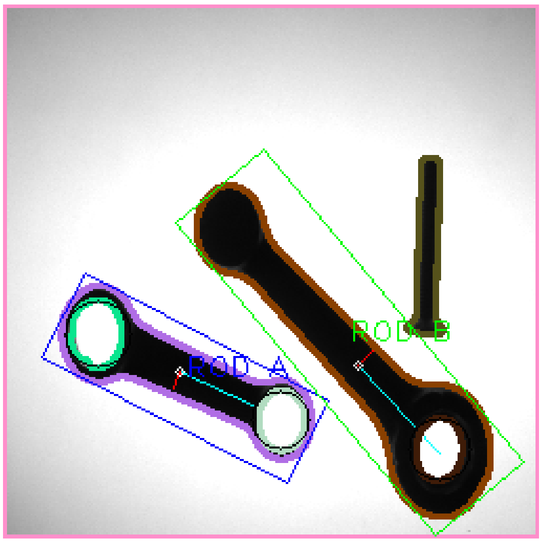


Illustration 10: TESI48.BMP Presence of "distractors"

PRESENCE OF IRON POWDER IN THE INSPECTION AREA

The small powder can interfere in the classification but it can be dealt with by noting that these dust points have a very small area. So in the search for holes and rodes we exclude the the contours that have a very small area. The area is computed as the moment of order (0,0).

```
contour_moments=moments(contours[i],true);
if(contour_moments.m00>50{
      /*continue the classification*/
}
```

In order to calcel out from the image the very small dust a median filter could be applied to the image as we can notice from the images below. The colored contours are those that persist after the median filter has been applied(in the second image practically all the dust is removed from the contour extraction process).

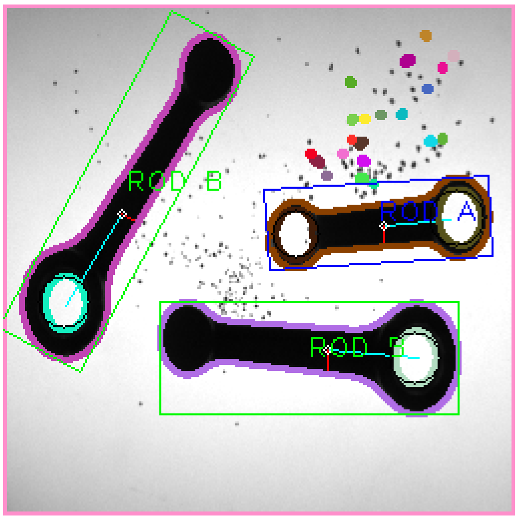


Illustration 12: TESI.92.BMP Presence of iron powder in the inspection area

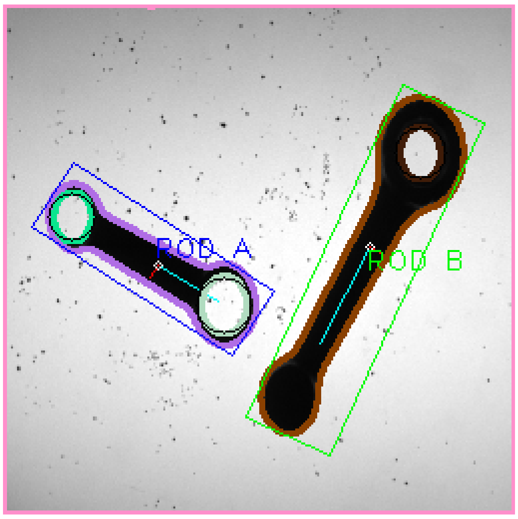


Illustration 13: TESI98.BMP Presence of iron powder in the inspecion area

RODS WITH CONTACT POINTS

Some rods appear having a contact points with others and after the binarization they are considered as a single blob (having one big contour)and to detect this scenario, we can notice that the found touching-rods blob have an area that is a sum of the areas of the single non touching rods. After finding this region of interest a possible solution is to separate the blob by detecting the two extreme points in the touching zone. These points will give a high response to the Harris corner detector.

To separate the two rods using these points, a simple solution is to draw a line between tho nearby detected corners with the same color of the background(i.e. black) and to try again to detect the contours in the modified image. This procedure is iteratively carried out until no too big contours are found (i.e. with an area greater than the one of the biggest single rod) .

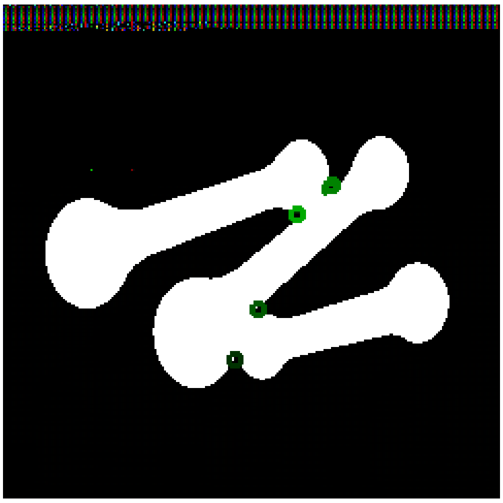


Illustration 15: TESI51.BMP Touching rods and the corners detected by Harris

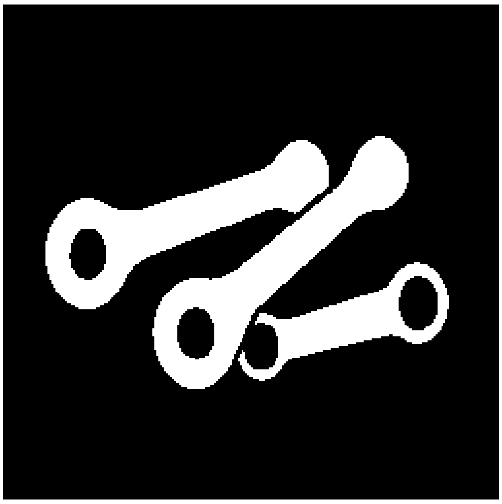


Illustration 14: TESI51.BMP A line is drawn between two nearby corners to separate the rods

The complete process is carried out in the function **separateUsingCorners() .**

```
int count=1;
      while(count!=0){
      //find the contours
      findContours(temp_binary,contours_updated,hierarchy,
      CV_RETR_TREE,CV_CHAIN_APPROX_NONE);
      count--;
      //check all of them
      for(int a=0;a<contours_updated.size();a++){
            Moments cont_moments=moments(contours_updated[a]);
            //if they are too big continue
            if(cont_moments.m00>4600 && cont_moments.m00<40000){
            Mat temp_color(inputImage.size(),CV_8UC3);
            drawContours(temp_color,contours_updated,a,Scalar(255,255,255),
            CV_FILLED,8);
            //look for strong corners in the image
            //NOTICE: WE LOOK FOR 4 OF THEM
            vector<Point2f> corners;
            goodFeaturesToTrack(temp_binary,corners,4,0.01,5,noArray(),5,false);
               //find the two nearest detected corners and draw a line
            double min=100;
            int index;
            for(int b=1;b<corners.size();b++){
            Point2f diff=corners[0]-corners[b];
            double di=sqrt(diff.x*diff.x+diff.y*diff.y);
            if(di<min){
                  min=di;
                  index=b;
                  }
            }
      line(temp_binary,corners[0],corners[index],Scalar(0,0,0),2,8);
      findContours(temp_binary, contours_updated,hierarchy,CV_RETR_TREE,
      CV_CHAIN_APPROX_NONE);
      for(int c=0;c<contours_updated.size();c++){
            Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255),
            rng.uniform(0,255));
            drawContours(inputImage,contours_updated,c,color,1,8);
            Moments cont_mom=moments(contours_updated[c]);
            if(cont_mom.m00>4600 && cont_mom.m00<40000 && count!=1){
                  count++;
                  }

            }
      }
```

A counter has been used to keep looking for corners until the contours found in the image are small enough (that means no touching rods are present).The count value is kept different than zero until such a condition is not reached.

The result can be seen below.

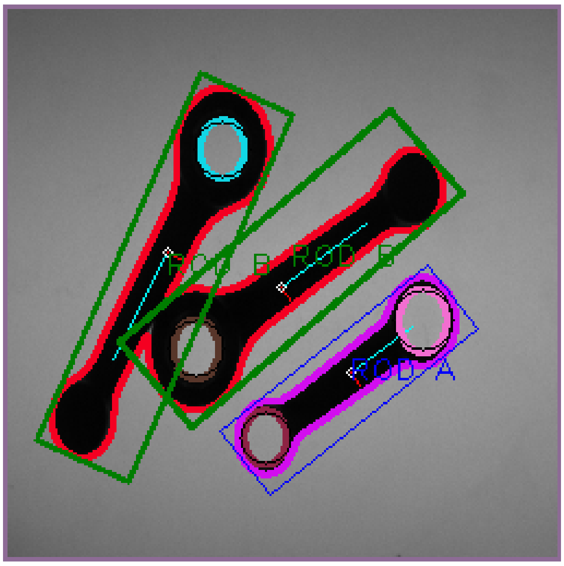


Illustration 16: TESI50.BMP Detected touching rods

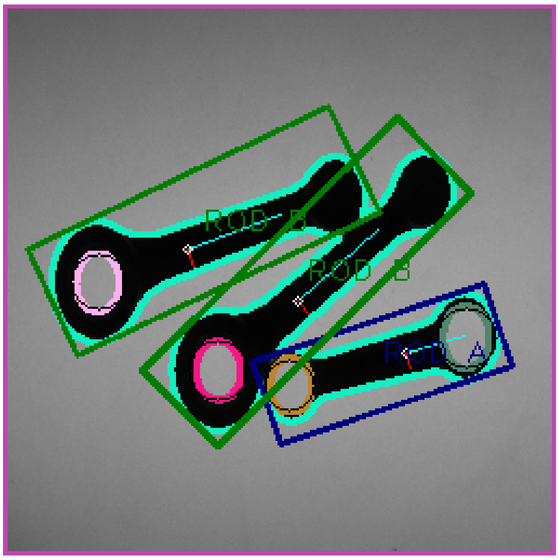


Illustration 17: TESI51.BMP Detected touching rods