**Problem Statement :**

Detecting defective needle-pin bearings that have missing pins or needles from images of the top view of the bearings.

This approach requires certain assumptions to be met in order to work on a given image:

* The image is taken with the bearing as the primary object such that the diameter of the bearing covers at least 1/4th of the size of the smaller dimension of the image.
* All the rollers/pins in any given bearing are identical although they may differ from bearing to bearing.
* Rollers must of different grayscale shade than the background.
* All the input images must have a correct image extension else imwrite will throw an exception as the output image is written to result folder with same name .

**Steps for execution:**

Make sure that detectRollers.h, bearing.cpp, result and test folder are in the same directory and configure build task . If the bearing is of a different kind with different number of rollers, set the constant ‘max\_rollers’ to that number in *detectRollers.h .*

Provide the name of the folder that has test images though command line input.

**Overview of the approach used:**

The key steps of this approach are:

1) Pre-processing (done thrice for different tasks)

2) Finding the surrounding/outer circle (Circumference of the bearing)

3) Detecting possible roller circles using Hough Circle Transform

4) Fitting a circle through the centre points of the detected circles from the above step

5) Performing moving window comparisons along the circumference of the fitted circle to detect all the rollers

6) Refining the boundaries of the detected rollers

1. **Pre-processing:**

The initial image is always resized to 640x480 at the start.

Slightly different approaches for pre-processing were used for the steps 2,3, and 5 that seemed best for the particular tasks.

For step 2, simple binary thresholding on the grayscale image and a couple of median blurs seemed to suffice as the boundary of the bearing was clearly distinguishable from it’s surroundings. This is coded as ‘*preprocess1*’

For step 3, adaptive thresholding was used because it would produce sharper binary image. It works by looping over all possible pixels that have a well-defined neighbourhood . Then it compares the pixel intensity with average intensity of its neighbourhood (15x15 used here). If the pixel deviates much from the average then it’s assigned max value.

But adaptive also amplifies noise, so rigorous median blurring was applied.

Then open and close morphological operators were applied for enhancing the contours.

Finally , Sobel filter for sharpening the edges (subtracting Sobel image ) .This is coded as ‘*preprocess2’.*

For step5, some additional median blurring was applied to the result of ‘*preprocess1*’.

All of these steps are applied on the grayscale version of the image.

1. **Finding the surrounding/outer circle:**

Hough Circle transform was used to get the bigger circle of the bearing. Depending on the noise, the outer circle or the inner circle surrounding the rollers may be detected. But parameters to the Hough accumulator are given such that only one circle with radius between than 1/8th and 3/4th of the smaller dimension get detected. It doesn’t matter which of the two circles get detected.

Briefing on Hough Circle transform:

It is used to detect circles in imperfect images. Firstly edges are generated by applying Canny operator( Canny operator works by applying Sobel operator two times with different thresholds and comparing the results to obtain significant contours) . After the contours are obtained, a voting based system is used to detect circles. A 3D matrix called accumulator matrix is maintained. Each entry corresponds to a specific x\_c, y\_c (centre coordinates) and radius. Iterating through all the pixels, if a pixels lies on an edge(white pixel), all possible circles passing through that point whose(x,y,r are well defined) are incremented by a vote count of 1. Thus at the end of the iterations entries with vote count higher than minimum specified (as a parameter) are outputted.

I chose to use this algorithm functionality as it’s simple and faster than what I could have implemented from scratch. Faster variants of this transform are out there but have yet to be implanted in the OpenCV 4.

1. **Detecting possible roller circles using Hough:**

As the bearing might be worn out or due to noise in the image or due to lighting conditions or many other reasons, the rollers won’t be perfect distinguishable circles and thus Hough transform will never be able to detect all the rollers in all the bearings, but at least some rollers will be detected correctly due to rigorous median blurring and other preprocessing steps. These detections can be used for the next steps. Only those rollers that lie inside the outer circle(if detected) and beyond ½ the radius of the same will be considered as correct detections.

1. **Fitting a circle passing through the centres of the corrected rollers:**

To detect all the rollers, I chose to implement a moving window approach in which an already detected circle through Hough is considered as a reference mask/filter and then compare this mask with all 360 patches whose centres (at 1 degree from one other) lie on the circle passing through the centres of the corrected rollers. So, given a set of centres of the corrected circles, I fit a circle using basic Linear Algebra.

A circle of the form (x-x\_c)^2 + (y-y\_c)^2 = r^2 can be represented as a linear equation in three variables g,f,c (standard representation x^2 + y^2 + 2\*g\*x + 2\*f\*y + c = 0

* (2\*x) \*g + (2\*y) \* f + (1)\* c = -(x^2 + y^2)

Thus given any number of (x,y) pairs this turns into a system of linear equations in g,f,c

From LA, if A\*X = b is not solvable, (rank = column space < row space),

we find a projection of b onto column space of A as p and solve A\*X = p

p = A \* (A\_T \* A)^-1 \* A\_T \* b

Then we can find X using just any three equations.

X = (A\_3x3) ^-1 \* (p\_3x3) (Any of the three rows in A are independent so simply choosing first three)

Finally calculating x\_c, y\_c, and r from g, f, and c

Thus, atleast three corrected detection are required.

1. **Performing moving window comparisons:**

The radius of the rollers is estimated as 25th percentile among that of the corrected rollers. This circle is also chosen as reference roller for obtaining mask/filter.

A square enclosed in this reference roller is taken as mask.

Starting from 0 degrees to 360 with increments of 1 degree, an roi is computed at each point as the square of same size of mask, and simple difference is calculated.

Summing over these absolute differences and dividing by 255 gives the number of different pixels as the thresholded image is binary. If the error is lower than tolerance and the point is atleast at a diameter distance from the previous detection, this it is considered as a correct detection

A higher tolerance is set at first so that all the rollers get detected .

1. **Refining the boundaries of the detected rollers:**

As the initial tolerance is set high because some rollers might be very noisy and we don’t want to miss any roller, as a result we may not get the exact position of the rollers. Thus ,there is a trade-off between accuracy of the position and correct number of detections

(Bad analogy of Heisenberg’s uncertainty principle I know)

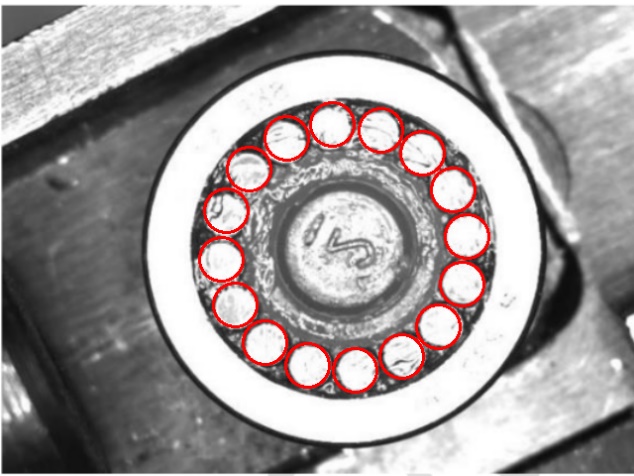
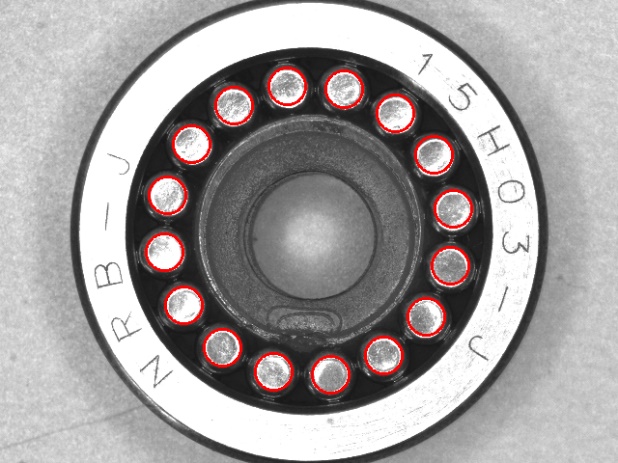
If it’s a **bad bearing** with less than correct number of rollers, I only try to minimize the tolerance such that maximum number of rollers are still being detected . But as bad bearings would probably require human intervention for deciding what needs to be done with them or maybe not but still I didn’t go further as there might be only minor error in the positions now.

But If it’s a **good bearing** , the bearing is probably uniform , symmetric throughout .

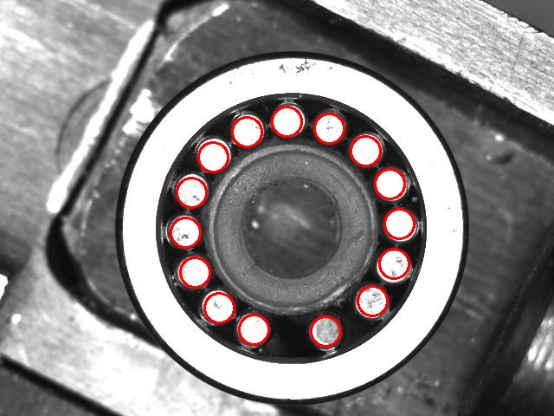
And exact positions of the rollers can be found by estimating the angle difference between one roller centre to another.

Examples:

**Good bearing :**

****

**Defective bearings:**

****