LAB: Gray scale Image Segmentation

Date: 2024-04-02

Author: Soonho Lim 21900610

Github: repository link

Introduction

1. Objective

This lab introduces counting different size of bolts and nuts positioned in random way shown in the image. Basically this lab processes the image in gray scale which will have 1 channel.

Goal: Count the number of nuts & bolts of each size which are oriented in different ways.

There are two different size bolts and three different types of nuts. It is required to segment the object, count and print it out. The image is composed with:

- Bolt M5
- Bolt M6
- Square Nut M5
- Hex Nut M5
- Hex Nut M6

2. Preparation

Software Installation

- OpenCV 4.9.0
- Visual Studio 2022

Image

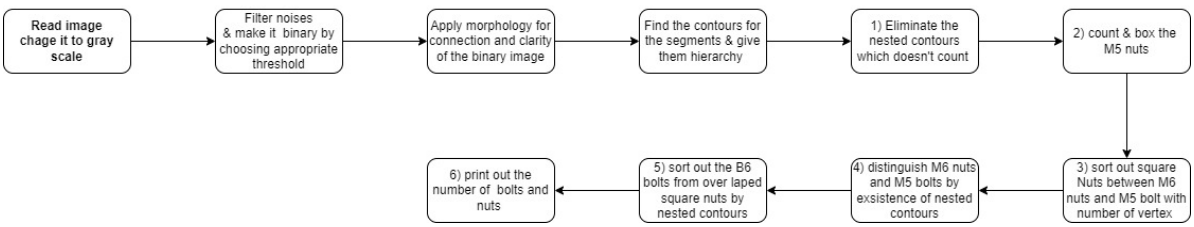
The image shows 20 different kinds of bolt and nuts oriented in different ways.

Image link: Download the test image

##

Algorithm

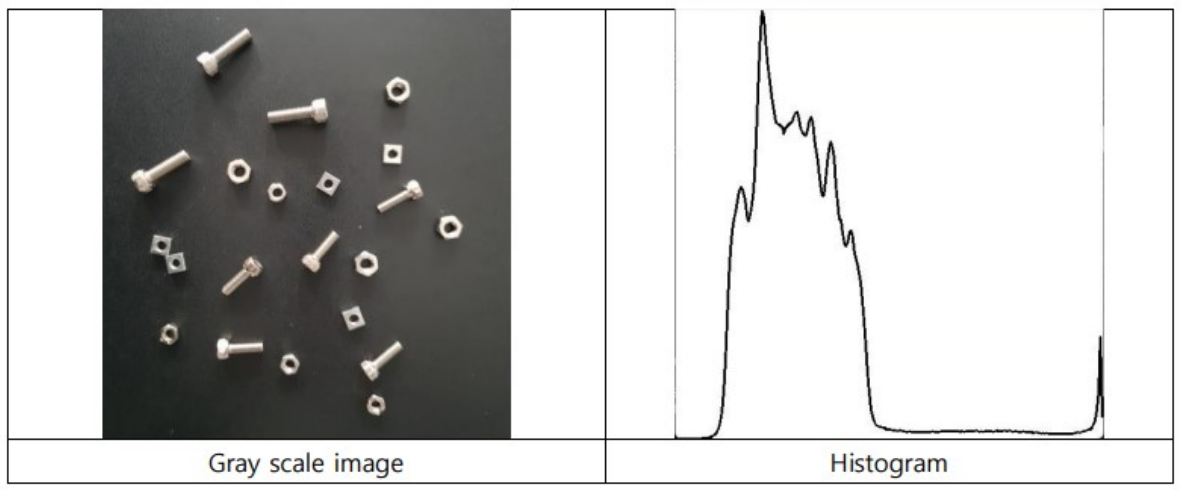
1. Overview



First, read the image and then filter the noises and make it to a binary data by choosing the appropriate threshold value to that later on it can find it's contours. Apply morphology, dilation and erosion, to give binary image clarity. Then find contours for each segments with use "RETR_CCOMP" mode which will give hierarchy vector for sorting indistinguishable segments. To get rid of nested contours eliminate extreme small value for contour's area. M5 nuts' value of contour area is actually distinguishable so first sort out the M5 nuts. Oppose to the M5 nuts, M6 nuts, three square nuts on the right side of the image and M5 bolts has a very similar values for the contour area. To sort the square nuts it calculates the vertex(actually its a side but the value is same) of the contour and when the value is 4, because the square has 4 vertex, it is counted as a square nut. For the M6 nuts and M5 bolts, hierarchy values for "First child" is used to know whether it has a nested contour. If it has its a nut and bolt if none. Last problem is dividing M6 bolts with overlap square nuts. Same method is used as M6 nuts and M5 bolts, it looks for the nested contours and distinguishes. Because it was hard to completely separate the two overlap square nuts, it is bound to be one contour. So to count the correct number of the nuts, vertex is calculated and divided in to 4(square has 4 vertex) and round up to added to the counting.

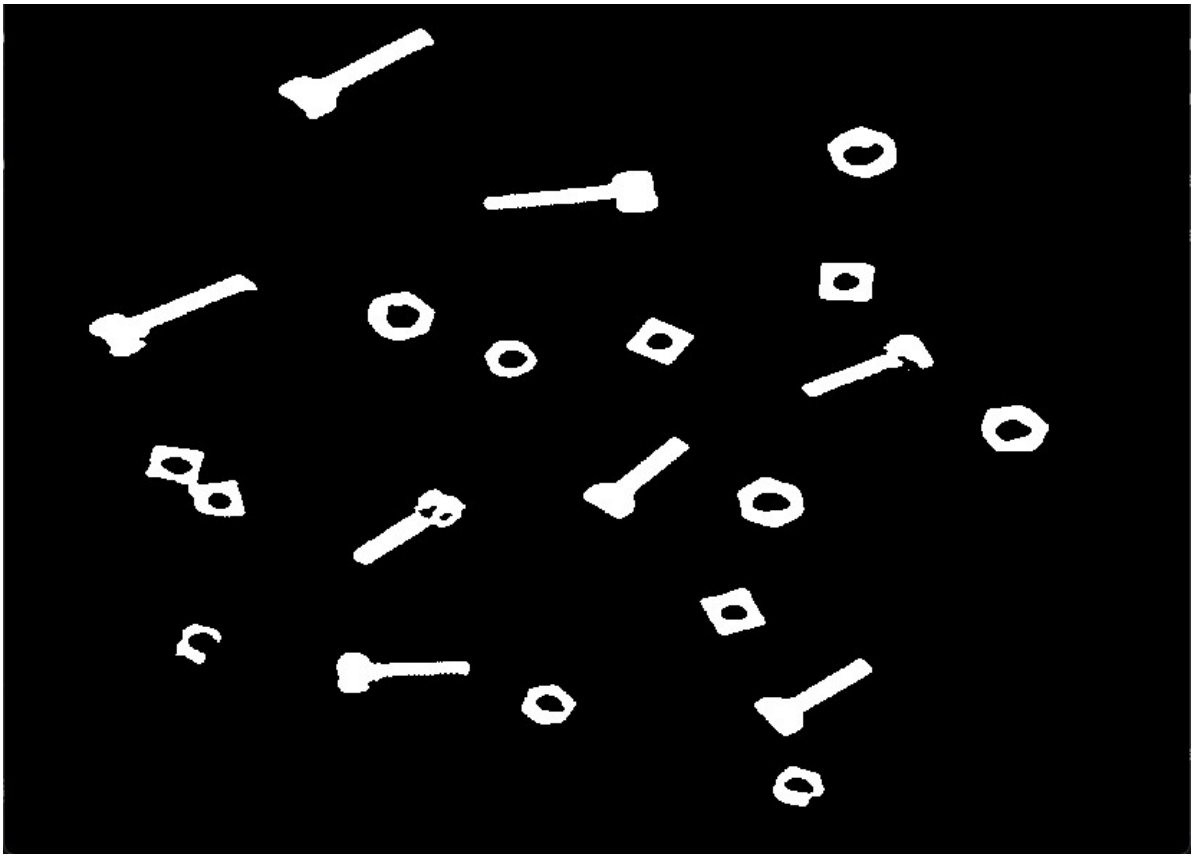
2. Procedure

Histogram Analysis



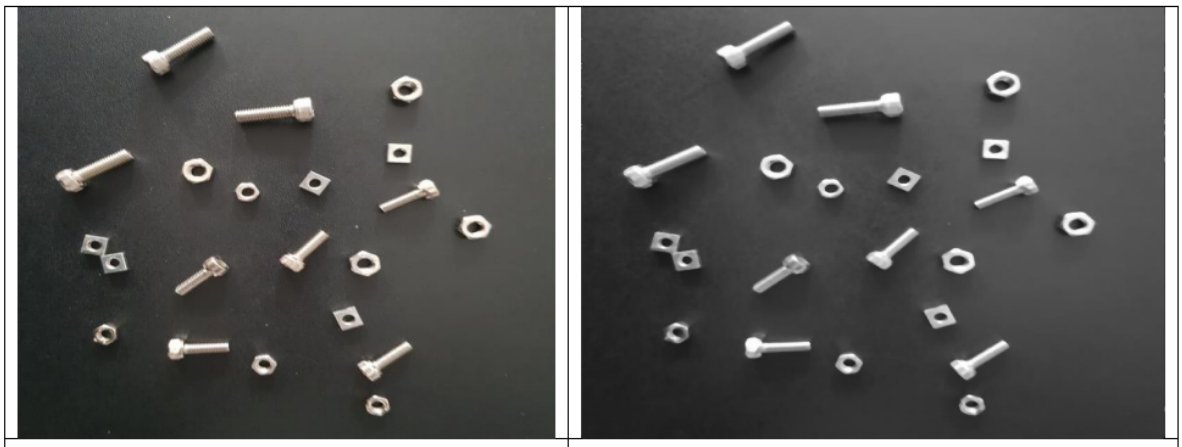
As shown in the histogram the bright component of objects can be segmented from mostly dark backgrounds. The appropriate threshold value will be in the some where middle in the histogram around the value of 110 ~ 200. For this reason threshold value of 150 is selected and for the threshold type OTSU method is chosen.

Before making it as a binary data, the background give a lot of unwanted noises. To get rid of it, median blur filter is applied and the size of the kernel is 11x11 because the noise is quite big. As a result the image turns to be like the image below.



Filtering

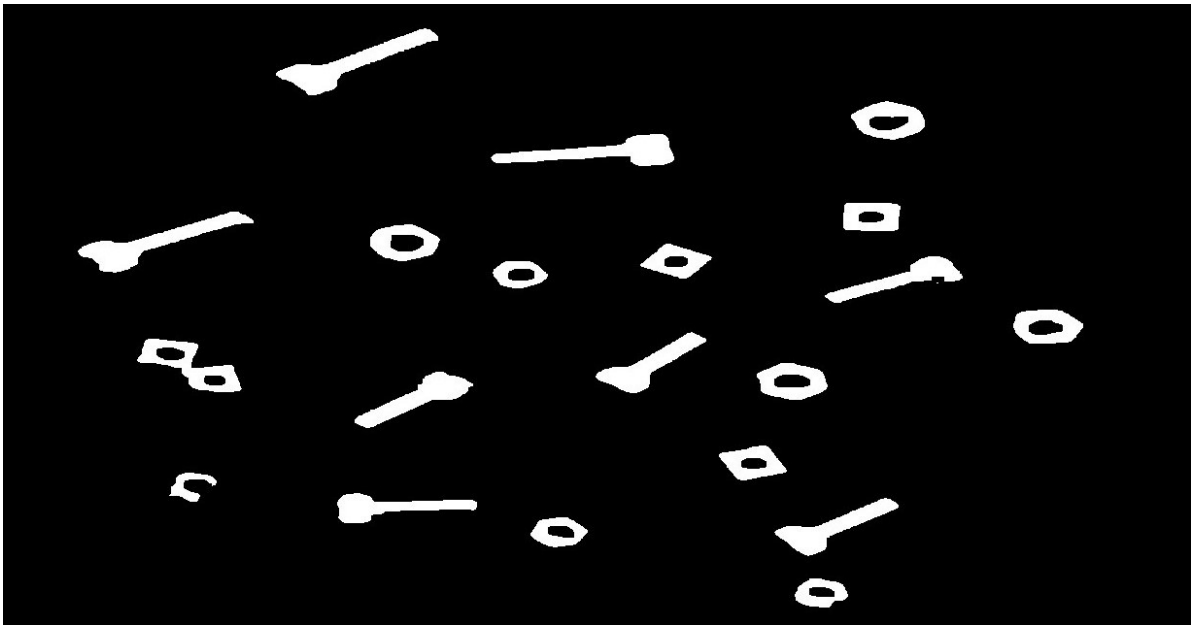
Mentioned earlier, the median filter is applied for the salt noises. The figure below shows the difference before and after the filter is applied.



Before Filtered	After Filtered

Thresholding and Morphology

Threshold values are analytically and experimentally chosen like discussed earlier. The result image is expected result in some way but yet, there are unexpected holes in some segments. To clarify and adjust this problem, dilation and erosion is used. The method used for morphology is 'closing'. It first dilate and then erode. The reason is that the if erosion take place first the holes will be bigger and it will need more bigger kernels meaning bigger changes from the original source image, so rather than using a 'opening' method, 'closing' seemed more appropriate. The result image after thresholding and morphology is below.

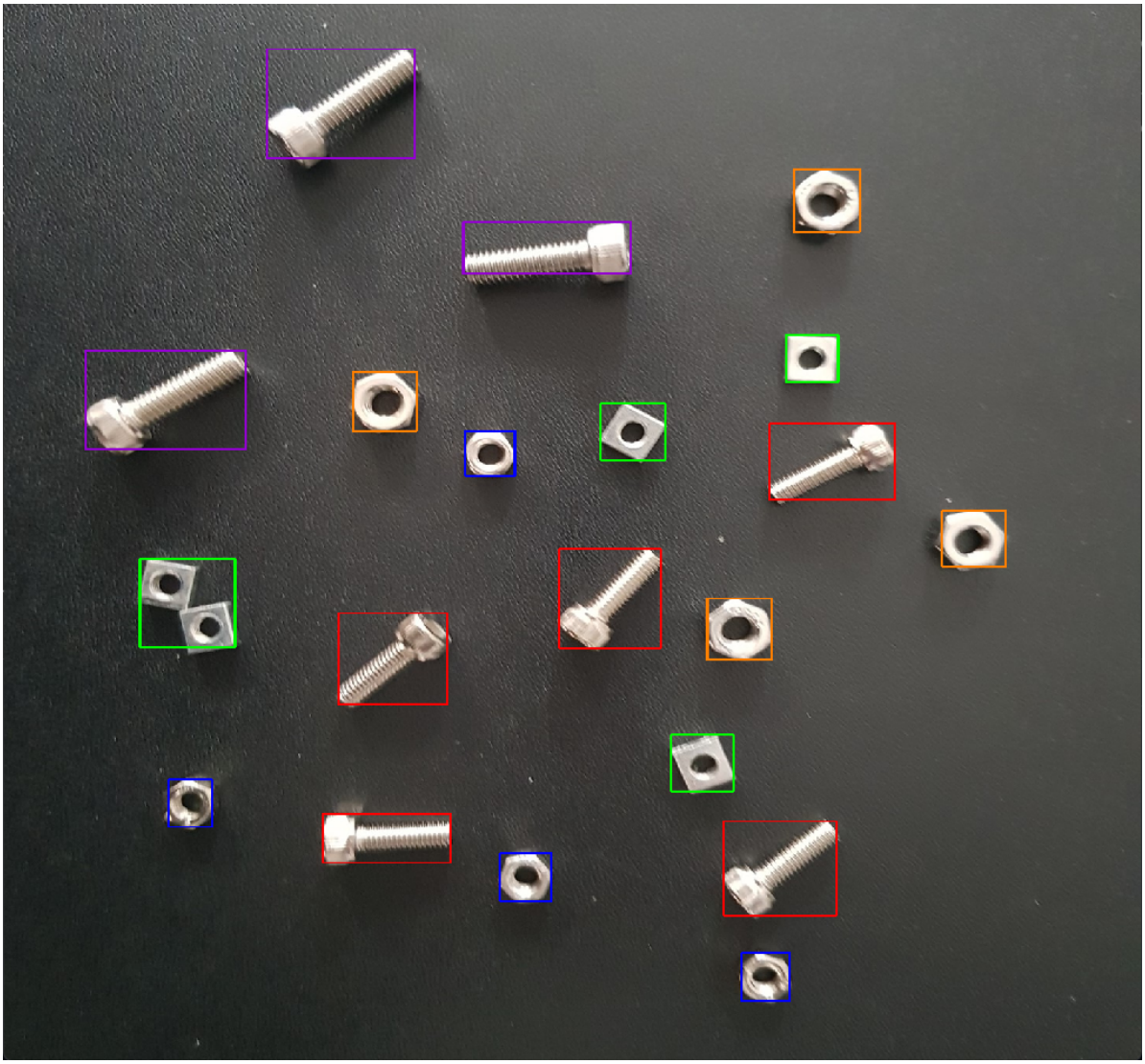


##

Result and Discussion

1. Final Result

The result image is shown below. First is the image and the second is the command window for printing out the numbers of bolts and nuts.



2. Discussion

There is focus points for the lab: 1. to get the right threshold value, apply appropriate filter and morphologies 2. distinguishing nuts and bolts to get the right result.

The first and the second points were interconnected to each other so if the value of filters and morphologies changes the values used for distinguishing was messed up again. So It first needed to fix the filter and morphology values and then step to the next point.

The first problem encountered was the overlap nuts which could not be separated. Eroding to much gave another problem that make the lower left M5 hex nut to vanish and less eroding make it still overlap. The way to solve this was not separating them but counting the number of vertex which can give information whether it is two or more.

The second problem was non reasonable contour area values for M5 bolts, M6 hex nuts, and square nuts. The way to solve was first count the vertex using function "approxPolyDP()" and sort the square nuts and then divide the two remaining type with existence of nested contour.

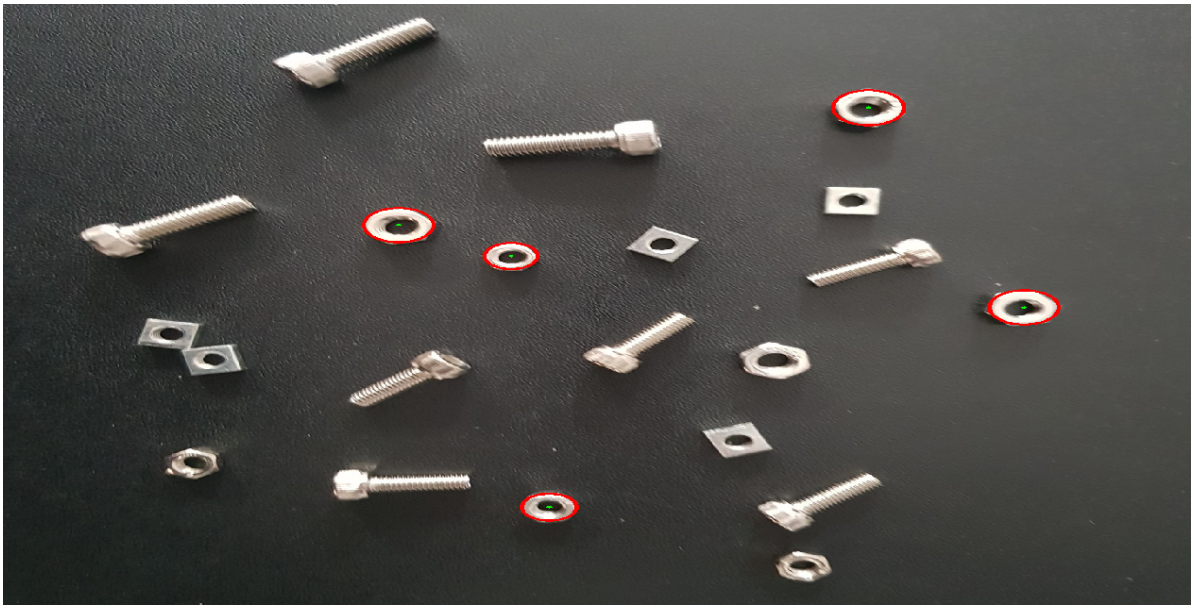
The last challenge was to distinguish M6 bolts with two overlap square nuts. Both of them had a similar value of the area, so what was done is also finding the nested contours.

- **Some Trials using Hough circle detection**

There was some trials for distinguishing problems. The first idea was to use "HoughCircles()" to sort the nuts which is more close to circle than the bolts relatively, but it did not work cause even though the "low threshold value was tuned it was not possible to find the perfect value the reason was there was some parts that looked like it is round after filtering and morphology process. The code used and the result it showed is posted below.

```
vector<Vec3f> circles;
HoughCircles(morph, circles, HOUGH_GRADIENT, 2, morph.rows / 64, 100, 70, 0,
100);

for (size_t i = 0; i < circles.size(); i++)
{
    Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
    int radius = cvRound(circles[i][2]);
    // draw the circle center
    circle(src, center, 3, Scalar(0, 255, 0), -1, 8, 0);
    // draw the circle outlinerfcv fdgdfgcv
    circle(src, center, radius, Scalar(0, 0, 255), 3, 8, 0);
}
namedWindow("circles", WINDOW_NORMAL);
imshow("circles", src);
```



The table below is the accuracy of the program.

Item	True	Estimate	accuracy
M5 bolt	5	5	100%
M6 bolt	3	3	100%
M5 hex nut	4	4	100%
M6 hex nut	4	4	100%
square nut	5	5	100%

This accuracy is calculated with the function "accuracy()" which calculates it for the following equation.

$$accuracy = (1 - \frac{|est - true|}{true}) \times 100$$

Conclusion

To summarize, there were two point for the lab, finding the right values for threshold, filter, and morphologies, and to distinguish the bolts and nuts. First point could be solved with analyzing histogram and taking tries and errors. Second point could be solve with hierarchy vectors showing the nested contours and the number of vertex because the square nuts shows almost exact number of 4 for the vertex value.

Dividing the bolts and the nuts also could be a good solution for the lab. The algorithm programmed abused the use of condition like 'if', 'else' so it will be better to shrink the size of the condition by finding more effective way to mark each segments.

Appendix

This is the header and the namespace used. DLIP.h is a header that is made personally for histogram plot and accuracy calculation.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <vector>
#include "DLIP.h"

using namespace std;
using namespace cv;
```

Inside the main function it first call the image and then change it to gray scale. "plotHist" function for plotting the histogram of the gray scaled image. "medianBlur()" is used for noise filter and the matrix 'th' gets the threshold applied image.

```
src =
imread("C:/Users/joshu/source/repos/DLIP/Images/Lab_GrayScale_TestImage.jpg");

cvtColor(src, src_gray, COLOR_BGR2GRAY);

plotHist(src_gray, "Histogram", 500, 500);

medianBlur(src_gray, src_gray, 11); // noise filter
threshold(src_gray, th, 150, 255, THRESH_OTSU);
```

The code below is the process of morphology which uses dilate and erode. It has the size of 13x13 and dilates first and next follows the erosion.

```
    /* Structuring Element for morphology */
    int shape = MORPH_RECT;
    Mat element = getStructuringElement(shape, Size(13, 13));

    /* Apply a morphology operation */
    dilate(th, morph, element);
    erode(morph, morph, element);
```

This is for finding the contours and also initializing the vector points for finding the vertex later on. It uses RETR_CCOMP mode to have hierarchy.

```
    /* Find contours */
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    vector<Point> approx;

    // use RETR_CCOMP mod which gives hierarchy values
    findContours(morph, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
```

"for loop" below is the process of distinguishing the segments and counting it one by one. For the sequence:

1. eliminates small values that represents nested contours.

2. sort out M5 nuts just by its value
3. sort square nuts from M5 bolts and M6 nuts by vertex value
4. divide M5 bolts and M6 nuts by existence of nested value
5. distinguish M6 bolt and overlap square nuts by nested value and count the overlap square nut number by calculating the vertex of the contour.

```
/* Draw bounding boxes & counting */
for (size_t i = 0; i < contours.size(); i++) {

    // make drawing for the box
    Rect boundingBox = boundingRect(contours[i]);

    // save area value to conArea
    float conArea = contourArea(contours[i]);

    if (conArea > 1500) { //eliminate small values of contour area ==> nest
contours

        if (conArea < 3400) { //M5 nuts
            m5nut_count++; //count up
            rectangle(src, boundingBox, Scalar(255, 0, 0), 2); //draw blue box
        }

        else if (conArea < 6000) { //M5 bolts, M6nuts, square nuts(no overlap)

            // calculate for the vertex(sides) of the index i contour
            approxPolyDP(contours[i], approx, arcLength(contours[i], true) *
0.02, true);

            if (approx.size() <= 4) { // this condition point to square shaped
contours
                squareNut_count++;
                rectangle(src, boundingBox, Scalar(0, 255, 0), 2); // green box
            }
            else if (hierarchy[i][2] > 0) { // this condition points to contour
that has nested contour
                m6nut_count++;
                rectangle(src, boundingBox, Scalar(0, 127, 255), 2); // orange
box

            }
            else {
                m5bolt_count++;
                rectangle(src, boundingBox, Scalar(0, 0, 255), 2); // red box
            }

        }

        else {
            switch (hierarchy[i][2])
            {
            case -1: //for the bolt
                m6bolt_count++;
                rectangle(src, boundingBox, Scalar(211, 0, 148), 2); // purple
box

                break;
```

```
            default:
                float add = approx.size(); // change data type by saving to
float typ / int --> float
                if (add/4 > 1) { // condition for two or more overlap
                    add = ceil(add / 4);
                    squareNut_count += add;
                }
                else {
                    squareNut_count++;
                }
                rectangle(src, boundingBox, Scalar(0, 255, 0), 2); //green box
                break;
            }
        }
    }
```

The code below is in DLIP.cpp for plotting the histogram and calculating the accuracy and print it out.

```
void plotHist(Mat src, string plotname, int width, int height) {
    /// Compute the histograms
    Mat hist;
    /// Establish the number of bins (for uchar Mat type)
    int histSize = 256;
    /// Set the ranges (for uchar Mat type)
    float range[] = { 0, 256 };

    const float* histRange = { range };
    calcHist(&src, 1, 0, Mat(), hist, 1, &histSize, &histRange);

    double min_val, max_val;
    cv::minMaxLoc(hist, &min_val, &max_val);
    Mat hist_normed = hist * height / max_val;
    float bin_w = (float)width / histSize;
    Mat histImage(height, width, CV_8UC1, Scalar(255));
    for (int i = 0; i < histSize - 1; i++) {
        line(histImage,
            Point((int)(bin_w * i), height - cvRound(hist_normed.at<float>(i,
0))),
            Point((int)(bin_w * (i + 1)), height - cvRound(hist_normed.at<float>
(i + 1, 0))),
            Scalar(0), 2, 8, 0);
    }

    imshow(plotname, histImage);
}


void accuracy(int m5bolt[], int m6bolt[], int m5nut[], int m6nut[], int sqnut[])
{
    float accuracyM5Bolts = (1 - abs(m5bolt[0] - m5bolt[1]) / (float)m5bolt[1])
* 100;
```

```
    float accuracyM6Bolts = (1 - abs(m6bolt[0] - m6bolt[1]) / (float)m6bolt[1])
* 100;
    float accuracyM5Nuts = (1 - abs(m5nut[0] - m5nut[1]) / (float)m5nut[1]) *
100;
    float accuracyM6Nuts = (1 - abs(m6nut[0] - m6nut[1]) / (float)m6nut[1]) *
100;
    float accuracySquareNuts = (1 - abs(sqnut[0] - sqnut[1]) / (float)sqnut[1])
* 100;

    // Overall accuracy
    float overallAccuracy = (accuracyM5Bolts + accuracyM6Bolts + accuracyM5Nuts
+ accuracyM6Nuts + accuracySquareNuts) / 5;

    // Print the accuracies
    cout << "\n\nAccuracy for each Bolts and Nuts:" << endl;
    cout << "M5 bolt accuracy: " << accuracyM5Bolts << "%" << endl;
    cout << "M6 bolt accuracy: " << accuracyM6Bolts << "%" << endl;
    cout << "M5 nut accuracy: " << accuracyM5Nuts << "%" << endl;
    cout << "M6 nut accuracy: " << accuracyM6Nuts << "%" << endl;
    cout << "Square nut accuracy: " << accuracySquareNuts << "%" << endl;
    cout << endl;
    cout << "Overall accuracy: " << overallAccuracy << "%" << endl;

    return;
}
```