LAB: Grayscale Image Segmentation

Date: 2024-March-31

Author: Gyeonheal An 21900416

Github: https://github.com/AnGyeonheal/DLIP_GH

Demo Video:

Introduction

I. Objective

Goal: Count the number of nuts & bolts of each size for smart factory

There are 2 different size bolts and 3 different types of nuts. You are required to segment the object and count each parts

- Bolt M5
- Bolt M6
- Square Nut M5
- Hexa Nut M5
- Hexa Nut M6

Requirements

1. Apply appropriate filters to enhance image
2. Explain how the appropriate threshold value was chosen
3. Apply the appropriate morphology method to segment parts
4. Find the contour and draw the segmented objects.
5. For applying contour, see Appendix
6. Count the number of each parts

II. Preparation

Software

- Clion [Cmake]
- OpenCV 4.9.0

Dataset

This image includes various type and size of bolts and nuts. These bolts and nuts are scattered irregularly.

Dataset link: Download the test image

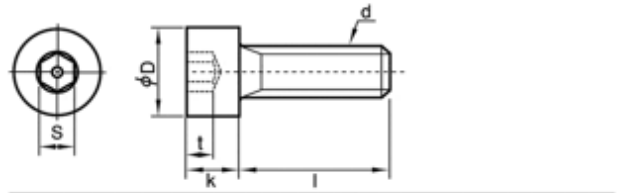
Algorithm

I. Overview

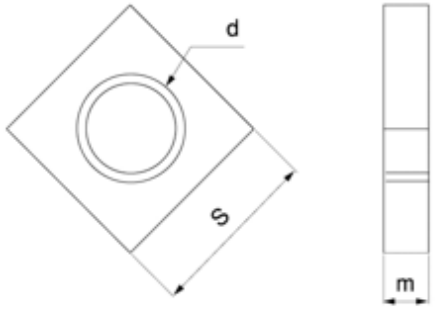
i. Analyzing Dataset

Before creating the algorithm, we must know what the components are in the datasets.

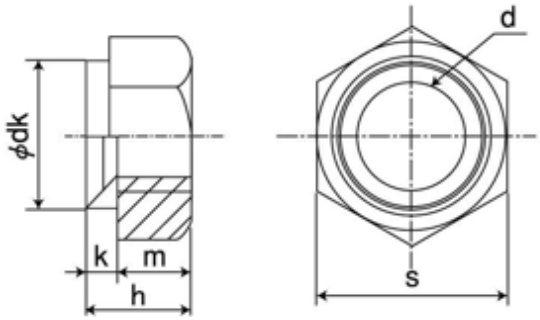
In the datasets, we can see the various type of bolts and nuts.



	헤드부 높이 K	헤드부 직경 D	렌치 사이즈 S	육각 홈 깊이 T(최소)
M3	3.0	4.5	2.5	1.3
M4	4.0	5.5	3.0	2.0
M5	5.0	7.0	4.0	2.5
M6	6.0	8.5	5.0	3.0
M8	8.0	10.0	6.0	4.0
M10	10.0	13.0	8.0	5.0
M12	12.0	16.0	10.0	5.0



나사 호칭	S	m
M3	8.4	1.2
M4	8	3.2
M5	10	3.0



나사의 호칭 (d)	육각 몸 직경(S) 기준 치수	육각 몸 직경(S) 허용 차	타원끝부 외경 (dk) 기준 치수	타원끝부 외경 (dk) 허용 차	타원끝부 높이 (k) 기준 치수	타원끝부 높이 (k) 허용 차	전체 높이(h) 기준 치수	전체 높이(h) 허용 차	육각부 높이(m) 참고
M5	8	0 −0.2	5.8	+0.05 −0	0.8	± 0.15	5.2	± 0.15	4.0
M6	10	0 −0.2	7.9	0 −0.2	1.4	± 0.1	6.3	± 0.1	4.9
M8	13	0 −0.25	9.5	+0.05 −0.1	1.8	± 0.1	8.2	± 0.1	6.4
M10	17	0 −0.25	11.55	± 0.1	2.1	± 0.1	9.9	± 0.1	7.8
M12	19	0 −0.35	15.2	0 −0.15	2.4	± 0.15	12.1	± 0.15	9.7
M16	24	0 −0.35	19.5	± 0.1	3.0	± 0.15	15.7	± 0.15	12.7
w3/8	17	0 −0.25	11.6	0 −0.2	2.2	+0.2 −0	10.2	± 0.1	7.9

Figure 1. Bolt & Nut Data Sheet

In data sheets, we can see differences between those components.

1. M6 bolt has bigger size than M5 bolt.
2. M5 rectangle nut has bigger size than M5 hexa nut. (M5 rectangle nut: 100mm^2 < M5 hexa nut: 55.2mm^2)
3. M6 hexa nut's hole has bigger size than M5 rectangle nut's hole. (M5 hexa nut: 5.8mm < M6 hexa nut: 7.9)

ii. Diagram

With these characteristics, I created algorithm through sequences of block diagrams below.

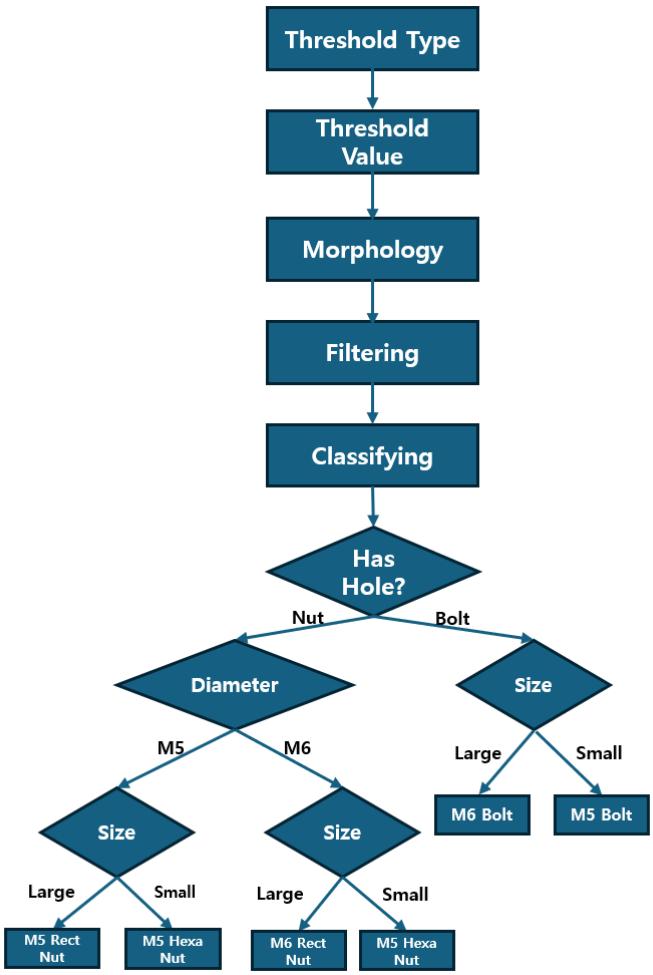


Figure 2. Algorithm Flow Chart

II. Procedure

i. Gray Scale

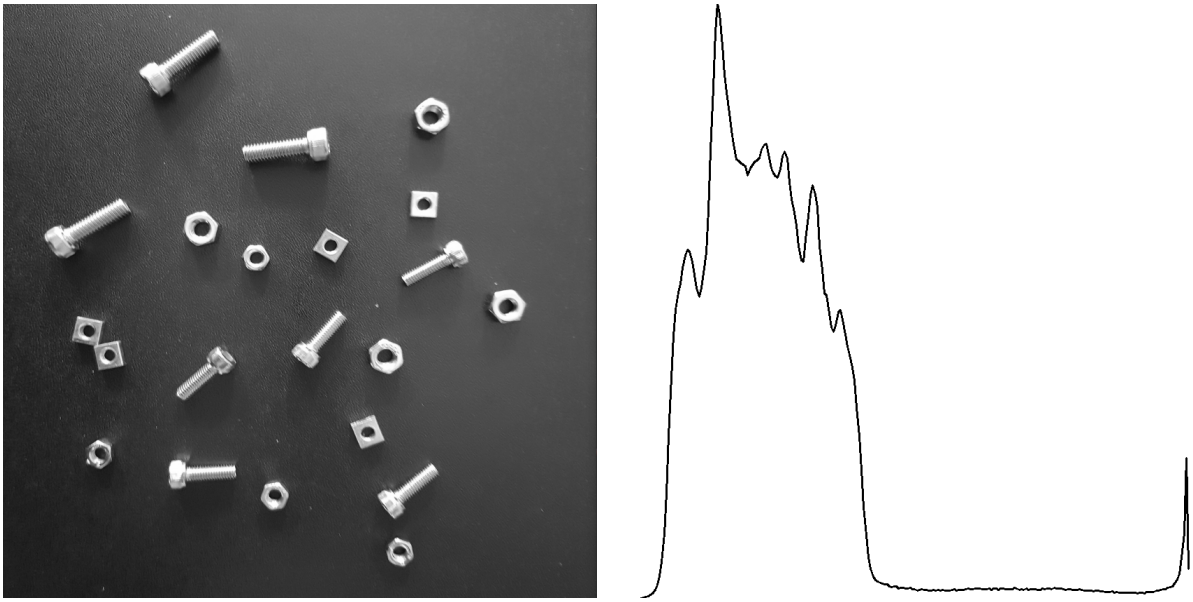


Figure 3. Gray Scale Image and Histogram

When analyzing an image converted to Gray Scale, the top right of the image is bright due to the angle of the lighting, while the bottom left is dark, resulting in an uneven brightness. Upon examining the histogram, the contrast of the object in question and the background does not show a distinct distribution. Therefore, to improve this, a process was conducted to clearly divide the object and the background based on a Threshold.

ii. Thresholding

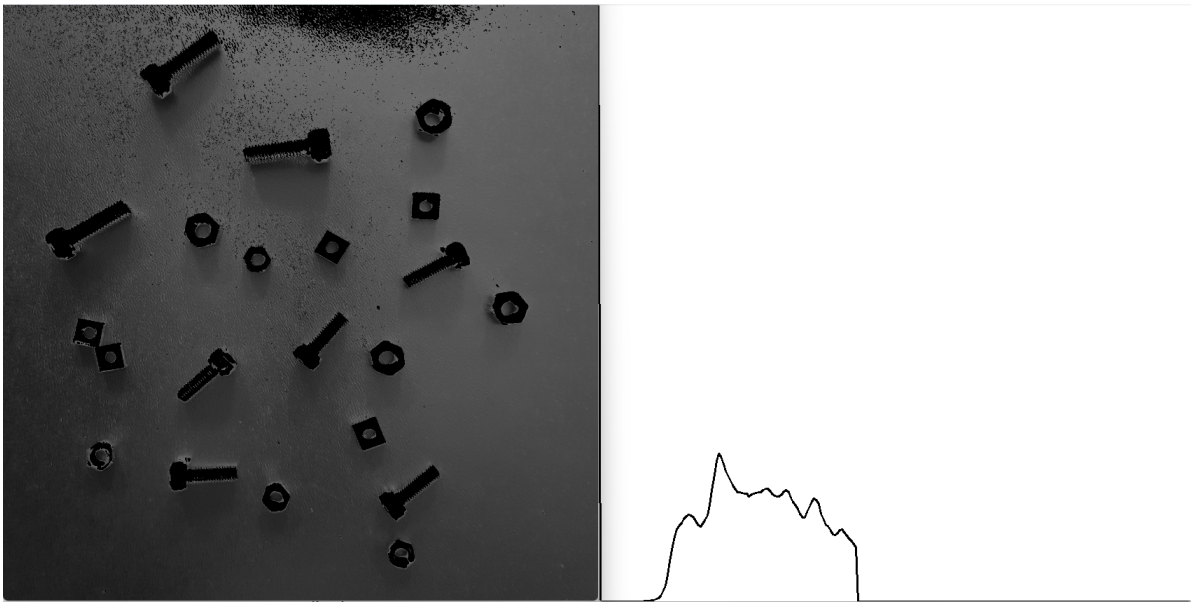


Figure 4. Thresholding Image and Histogram

The used Threshold type is 'THRESH_TOZERO', where if the pixel value is greater than the specified Threshold Value (111), the pixel value is maintained as is, and if it is less than or equal to the Threshold Value, the pixel value is set to 0.

In the histogram, the objects that need to be separated are all set to a pixel value of 0, while the background maintains its original pixel values. This resulted in a distribution where many pixels are concentrated at the pixel value of 0 for the object, separating it from the background pixel values. However, this process caused pixels at the top of the image that did not exceed the Threshold Value due to lighting to create noise, which interfered with the classification of objects.

iii. Morphology

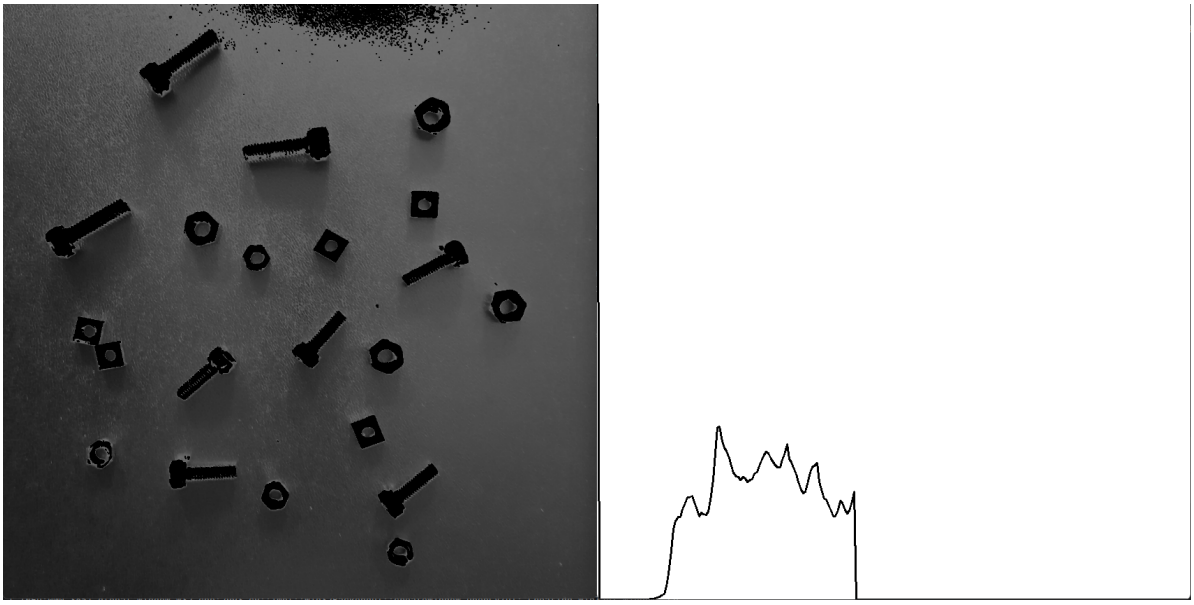


Figure 5. After Morphology Image and Histogram

The applied Morphology Type is CLOSE. This process first expands the pixels of the image (Dilate) and then contracts them (Erode), making it useful for filling small holes within the object or removing small black dots around the object. As a result, the shape of the object becomes smoother, and the separation between objects becomes more distinct.

The size of the Kernel was set to 3 because increasing the size of the Kernel results in the disappearance of the nuts in the bottom left, which did not receive much lighting.

iv. Filtering

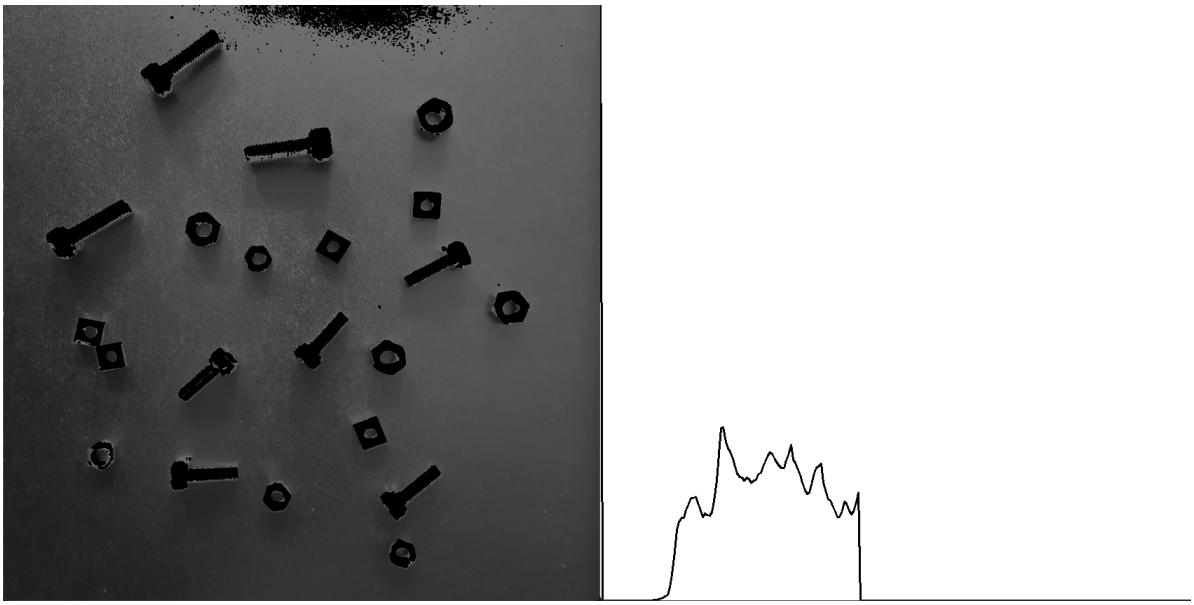


Figure 6. Filtered Image and Histogram

To remove smaller noises, the MEDIAN filter was used with a Kernel Size of 3. The reason for using the MEDIAN filter is that it can very effectively remove impulse noises such as salt-and-pepper noise. Even if pixels with noise have extreme values, using the median value of neighboring pixels allows the removal of noise while maintaining the main characteristics of the original image.

The Kernel Size was not increased to preserve the nuts in the bottom left, as mentioned earlier. The noise at the top will be ignored during future contour processing and will not be included in the calculations.

v. Classifying

The sequence for distinguishing types of bolts and nuts is as follows:

1. Classify noise, nut thread holes, and other contours based on the size of the contour.
2. Check whether the x,y coordinates of the thread holes exist within the area of other contours.

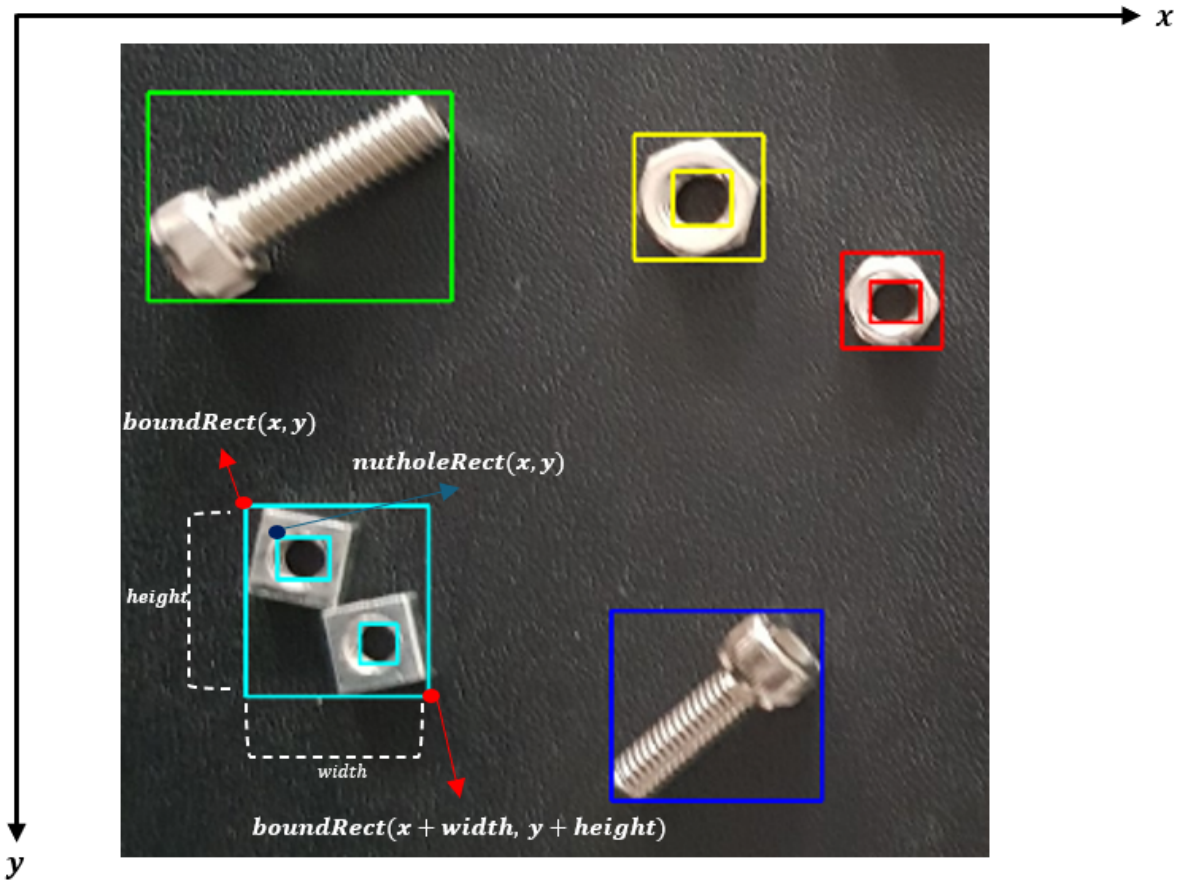


Figure 7. Algorithm description for detecting the location of threaded holes

3. If the coordinates of the thread holes exist within the area of another contour, distinguish between M5 and M6 nuts based on the diameter of the thread hole.
4. Since Rectangle Nuts and Hex Nuts have different total areas even if the diameter of the thread holes is the same, they are distinguished based on the area.

   ※ In the case of nuts being attached: Utilize the difference in size due to the difference in area between Rectangle Nuts and Hex Nuts when they are attached.
5. Save the index of the nut's contour in a vector from Other Contours.
6. Define those not corresponding to the nut's contour index in Other Contours as bolts.
7. Distinguish between M5 or M6 bolts based on the size of the contour.

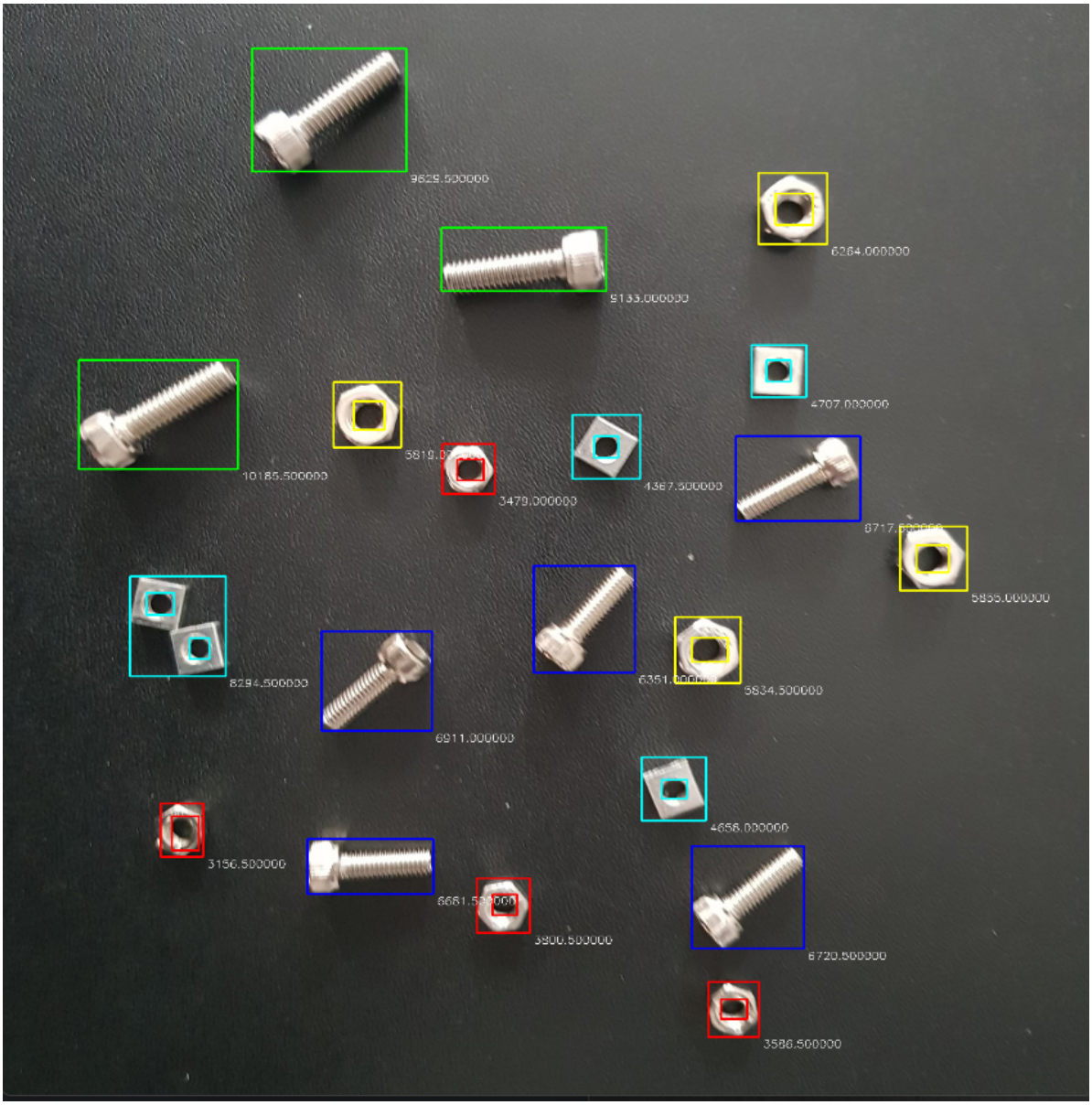


Figure 8. Differences in ContourArea size according to types of bolts and nuts

Result and Discussion

I. Result Image

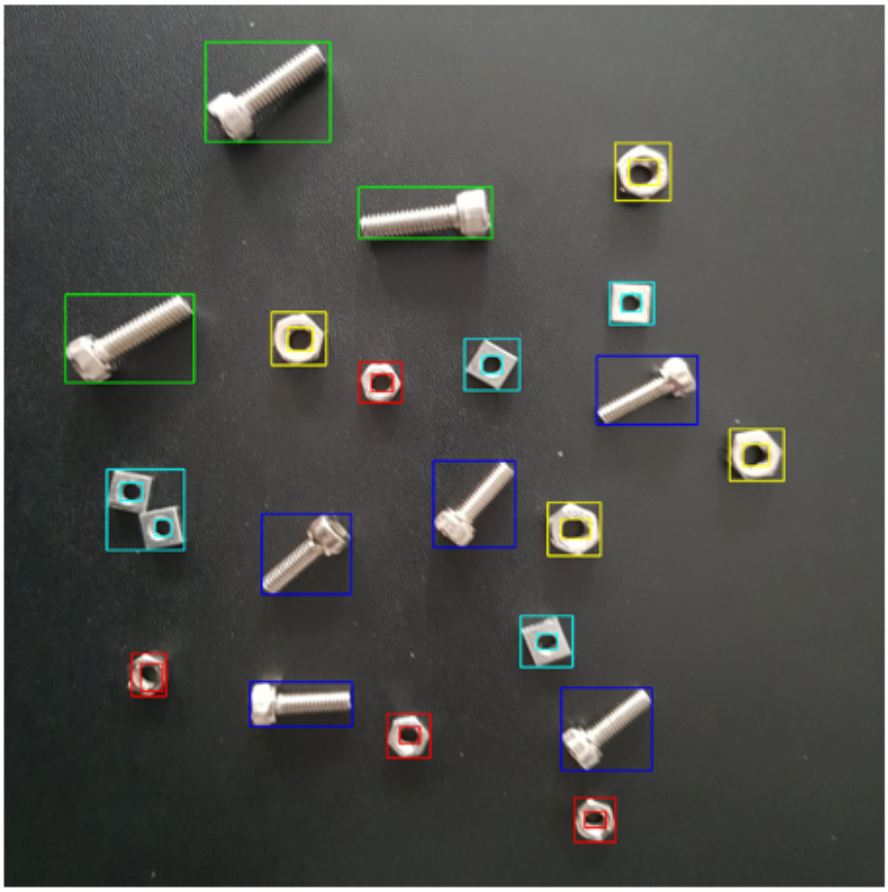


Figure 9. Result Image

II. Discussion

Items	True	Estimated	Accuracy
M5 Bolt	5	5	100%
M6 Bolt	3	3	100%
M5 Rect Nut	5	5	100%
M5 Hex Nut	4	4	100%
M6 Hex Nut	4	4	100%

Since the objective of this project is to obtain a detection accuracy of 100% for each item, the proposed algorithm has achieved the project goal successfully.

Conclusion

This project was conducted using OpenCV to distinguish bolts and nuts by size and type in a smart factory.

Since the data used in the project were static images, it was sufficient to adjust the pixel values of the images.

The given images, due to the angle of the lighting, did not clearly separate the pixel values of the background and the objects, necessitating the adjustment of the Threshold value to solve this issue.

Noises created during the image processing phase could be removed with CLOSED Morphology and MEDIAN Filter, and noises that could not be further removed were exception-handled in the process of recognizing objects, minimizing the impact of noise.

Recognizing and classifying objects was done by utilizing OpenCV's Contour to categorize the types and sizes of bolts and nuts according to the size of the object's contour area.

Since the images used in the project were static, the optimal method for pixel adjustment suited for these images was used. However, a drawback is that if the angle of lighting or the shooting angle changes, the image processing in this algorithm must be redone. Therefore, to overcome this, it seems necessary to improve the algorithm so that objects can be distinguished under any condition through learning with deep learning.

Appendix

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

void contour(Mat& src, Mat dst);

cv::Mat origin, src, src_gray, dst, dst_filtered, dst_morph, dst_morph2,
dst_normalized;
int element_shape = MORPH_RECT;
int n = 3;
Mat element = getStructuringElement(element_shape, Size(n, n));

vector<vector<Point>> contours;
vector<vector<Point>> contours_save;
vector<vector<Point>> contours_nuthole;
vector<int> nutIndex;
vector<Vec4i> hierarchy;

Scalar color_M5_B = Scalar(255, 0, 0);
Scalar color_M6_B = Scalar(0, 255, 0);
Scalar color_M5_HN = Scalar(0, 0, 255);
Scalar color_M5_RN = Scalar(255, 255, 0);
Scalar color_M6_N= Scalar(0, 255, 255);

int M5_B = 0;
int M6_B = 0;
int M5_HN = 0;
int M5_RN = 0;
int M6_N = 0;
```

These are the global variables used in image processing.

```
int main(){
    // |************************ Read, Resize, Convert to GrayScale Sourc Image
************************|
    src = cv::imread("../Image/Lab_GrayScale_TestImage.jpg", 1);
    cvtColor(src, src_gray, COLOR_BGR2GRAY);

    // |************************ Tresholding and Morpholgy
************************|
    threshold(src_gray, dst, 111, 255, 4);
```

```
    morphologyEx(dst, dst_morph, MORPH_CLOSE, element);

    // |************************ Filter ************************|
    dst_filtered = filter(dst_morph, MEDIAN, 3);
    imshow("dst_filtered", resizing(dst_filtered, 850, 850));

    // |************************ Ploting Histogram ************************|
    plotHist(dst_morph,"", 850, 850);

    // |************************ Draw Contours on the original image
************************|
    contour(src, dst_filtered);
    imshow("Contours", resizing(src, 850, 850));

    std::cout << "M5 Bolt: " << M5_B << std::endl;
    std::cout << "M6 Bolt: " << M6_B << std::endl;
    std::cout << "M5 Hex Nut: " << M5_HN << std::endl;
    std::cout << "M5 Rect Nut: " << M5_RN << std::endl;
    std::cout << "M6 HEX Nut: " << M6_N << std::endl;

    waitKey(0);
}
```

In the main function, the image is loaded, converted to Gray Scale, and then Thresholding, Morphology, and Filter are applied.

Drawing the histogram is done using a function, which is located within the "OpenCV_Setting.h" header file.

```
Mat filter(Mat src_gray, int filter_type, int ksize){
    Mat dst_filtered, dst;

    int kernel_size = 3;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    switch(filter_type){
        case NONE: src_gray.copyTo(dst_filtered); break;
        case GAUSSIAN: GaussianBlur(src_gray, dst_filtered,
cv::Size(ksize,ksize), 0, 0); break;
        case MEDIAN: medianBlur(src_gray, dst_filtered, ksize); break;
        case LAPLACIAN:
            cv::Laplacian(src_gray, dst, ddepth, ksize, scale, delta,
cv::BORDER_DEFAULT);
            src_gray.convertTo(src_gray, CV_16S);
            dst_filtered = src_gray - dst;
            dst_filtered.convertTo(dst_filtered, CV_8U);
            break;
    }
    return dst_filtered;
}
```

The filter() function is designed to easily apply the type of filter and the kernel size of the filter when applying a filter.

```
Mat resizing(Mat src, int x_size, int y_size){
    Mat sized;
    resize(src ,sized, Size(x_size, y_size));
    return sized;
}
```

The resizing function is used to conveniently view images displayed on the screen through imshow() when the image exceeds the screen resolution.

```
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);
}
```

The plotHist function displays the histogram of the Matrix of interest during the image processing process. To make it visually easier to distinguish, the background is adjusted to white and the graph to black.

```
void contour(Mat& src, Mat dst){
    /// Find contours
    findContours(dst, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE);

    for(int i = 0; i < contours.size(); i++)
    {
        if(contourArea(contours[i]) < 330 || contourArea(contours[i]) > 30000 ){
            continue;   // useless contours
        }
        else if(contourArea(contours[i]) < 1500){
            contours_nuthole.push_back(contours[i]);    // saving nut hole
        }
        else{
            contours_save.push_back(contours[i]);   // saving others
```

```
        }
    }

    for(int i=0; i<contours_nuthole.size(); i++){
        Rect nutholeRect = boundingRect(contours_nuthole[i]);
        for(int j=0; j<contours_save.size(); j++){
            Rect boundRect = boundingRect(contours_save[j]);
            if(boundRect.x < nutholeRect.x && nutholeRect.x <
boundRect.x+boundRect.width && boundRect.y < nutholeRect.y && nutholeRect.y <
boundRect.y + boundRect.height){    // Finding the hole location
                nutIndex.push_back(j);      // Save the Index of nuts
                if(contourArea(contours_nuthole[i]) > 1000){    // Finding the
M6 Hex nut
                    rectangle(src, boundRect.tl(), boundRect.br(), color_M6_N,
2);
                    rectangle(src, nutholeRect.tl(), nutholeRect.br(),
color_M6_N, 2);
                    M6_N++;
                }
                else{
                    if(contourArea(contours_save[j]) > 8000){       // Finding
the M5 Rectangle nuts that are stuck together
                        rectangle(src, nutholeRect.tl(), nutholeRect.br(),
color_M5_RN, 2);
                        rectangle(src, boundRect.tl(), boundRect.br(),
color_M5_RN, 2);
                        M5_RN++;
                    }
                    else if(6500 < contourArea(contours_save[j]) &&
contourArea(contours_save[j]) < 8000){      // Finding the M5 Hex nuts that are
stuck together
                        rectangle(src, nutholeRect.tl(), nutholeRect.br(),
color_M5_HN, 2);
                        rectangle(src, boundRect.tl(), boundRect.br(),
color_M5_HN, 2);
                        M5_HN++;
                    }
                    else if(contourArea(contours_save[j]) < 4000){  // Finding
the M5 Rectangle nuts
                        rectangle(src, nutholeRect.tl(), nutholeRect.br(),
color_M5_HN, 2);
                        rectangle(src, boundRect.tl(), boundRect.br(),
color_M5_HN, 2);
                        M5_HN++;
                    }
                    else if(contourArea(contours_save[j]) > 4000){  // Finding
the M5 Hex nuts
                        rectangle(src, nutholeRect.tl(), nutholeRect.br(),
color_M5_RN, 2);
                        rectangle(src, boundRect.tl(), boundRect.br(),
color_M5_RN, 2);
                        M5_RN++;
                    }
                }
            }
        }
    }
    for(int k=0; k<contours_save.size(); k++){
```

```
        if (find(nutIndex.begin(), nutIndex.end(), k) == nutIndex.end()) {
 // Finding the Bolts
            Rect boundRect = boundingRect(contours_save[k]);
            if(contourArea(contours_save[k]) < 9000){
                rectangle(src, boundRect.tl(), boundRect.br(), color_M5_B, 2);
 // Finding the M5 bolts
                M5_B++;
            }
            else {
                rectangle(src, boundRect.tl(), boundRect.br(), color_M6_B, 2);
 // Finding the M6 bolts
                M6_B++;
            }
        }
    }
}
```

The Contour() function is used to classify the types and sizes of bolts and nuts after the image processing has been completed.

Initially, to exclude noise from classification, contours below a certain size were handled as exceptions, and holes and non-holes were classified and saved in vectors separately.

Following that, the process was carried out in the manner described in Algorithm-II. Procedure-v. Classifying. If a nut hole exists within another contour, it was classified as a Nut, and the index of that contour was saved in a separate vector for later use. Overlapping cases were also considered for classification. Contours recognized as Nuts were classified as Rect or Hex based on their size.

Finally, indices not present in the vector that stored Nut indices were classified as bolts, and the size of the contour was used to classify them as M5 or M6.