LAB 1: Grayscale Image Segmentation

Date: 2024-03-24

Author: Kwak Jin, 21900031

Github: https://github.com/Kwak-Jin/DLIP

Demo Video: -

Introduction

1. Objective

This project is done to count the number of nuts&bolts of each size for smart factory only using image processing.

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

My personal goal is to keep the original edges clear and remove the noise to improve the performance of image processing.

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

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

2. Preparation

- Write a list of HW/SW configuration
- Image Download

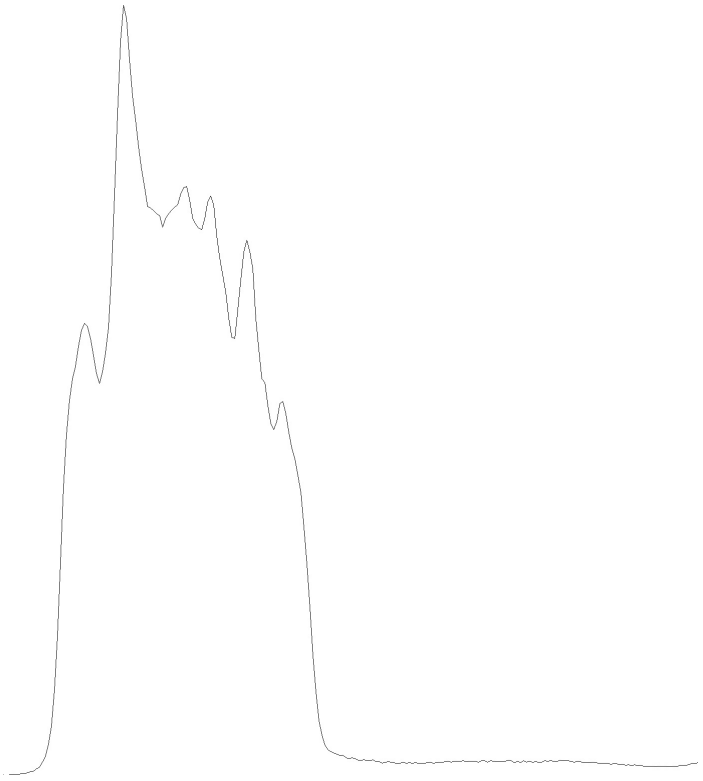
Software Installation

- OpenCV 4.8.0
- Jetbrains CLion (IDE)
- Language: C++17

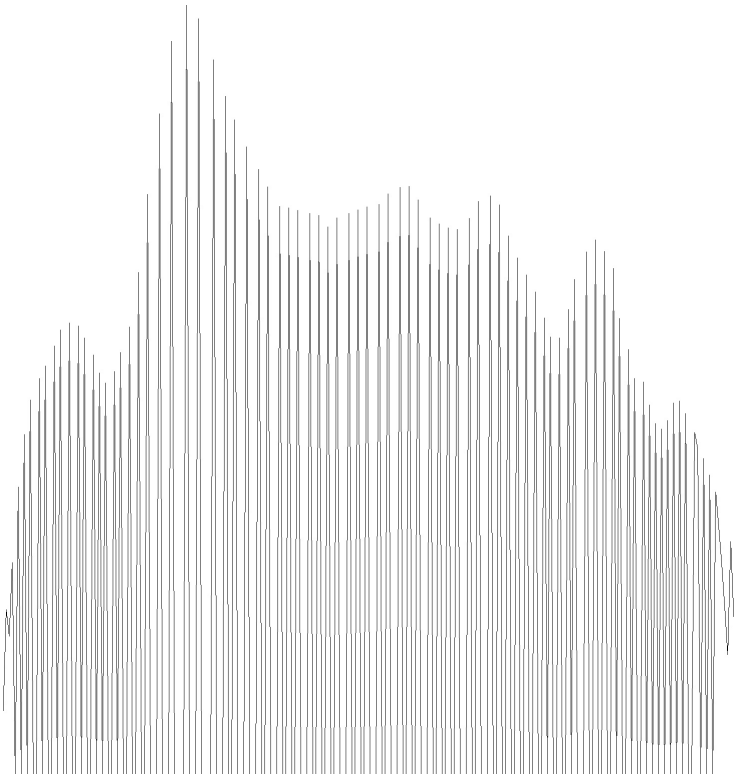
Dataset

The data set is shown as the following:

As in the histogram, the intensity is very concentrated near lower half. In other words, the contrast is relatively low and is not easy to distinguish between each objects.



Therefore, the histogram should be equalized using `equalizeHist()`. After this process, the histogram looks like the following:



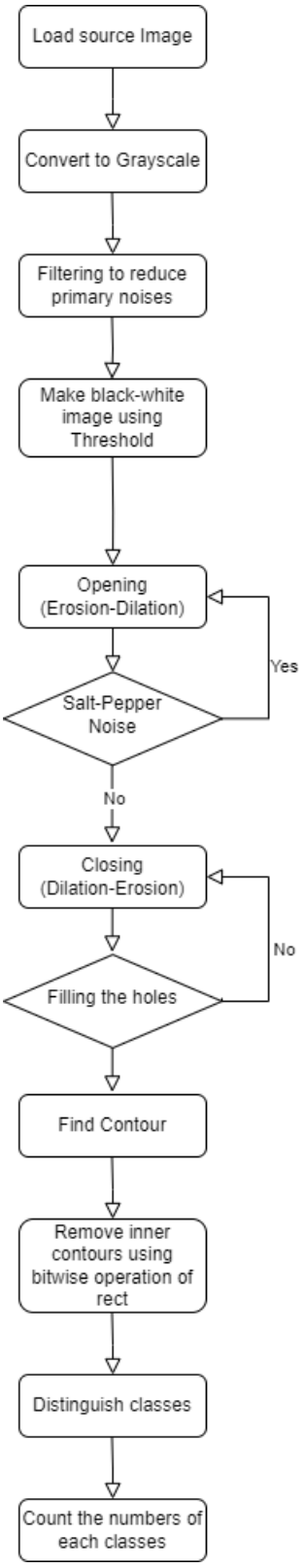
However, the equalized image was not used due to its indistinguishable characteristic.

Dataset link: Download the test image

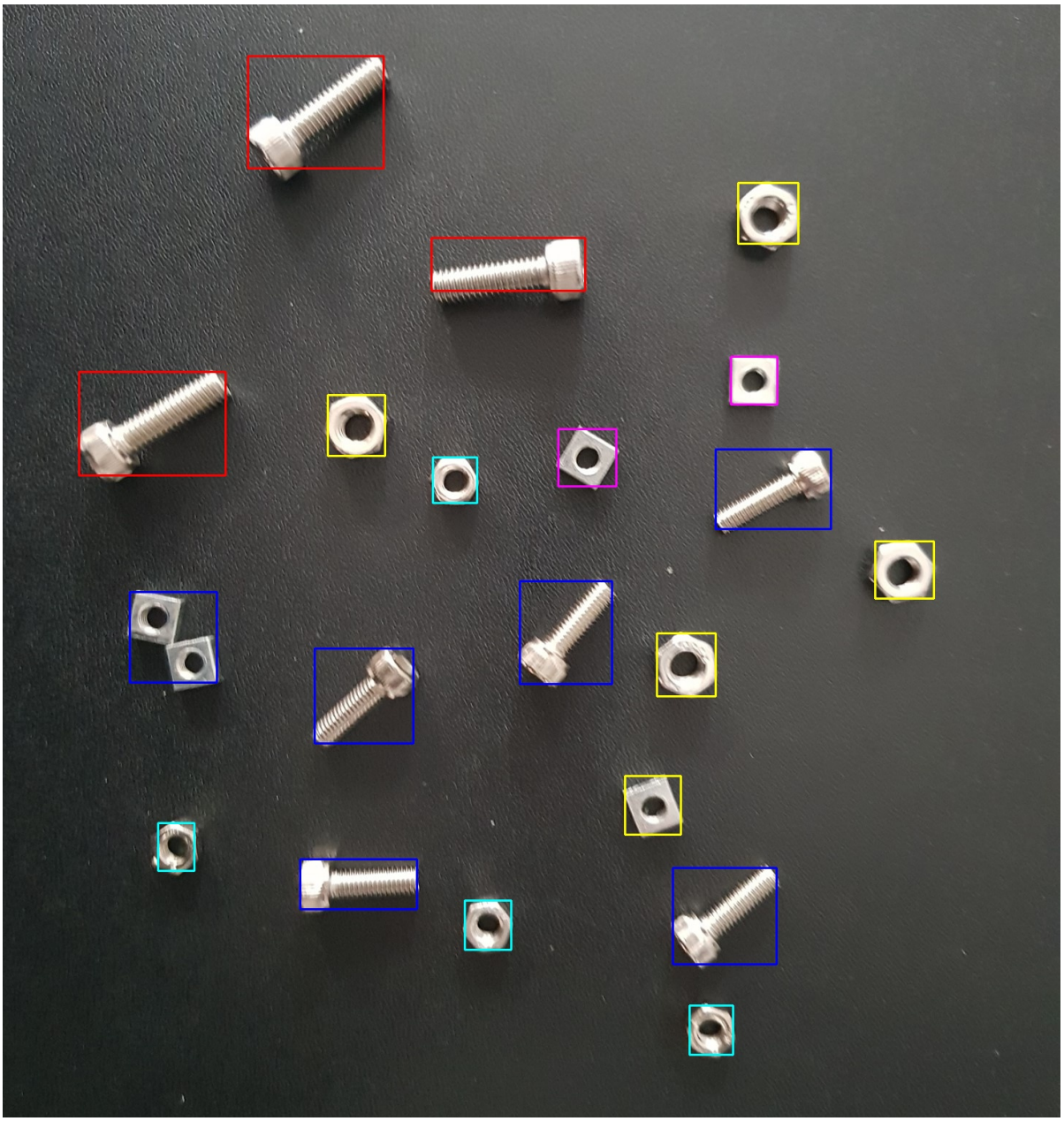
Algorithm

1. Overview

Flow Chart of the whole algorithm



Following the steps, the segmented image would look like this with the chosen threshold values and other constants. Only two instants are not classified correctly.



Filters used	Characteristics	Reason of the filter
median filter	Kernel size = 5	Erase salt and pepper noise
Gaussian Blur	Kernel size = 7, Variance = 1.5	Erase noise
Laplacian	Kernel size = 3, The laplacian filter is then weighted(0.7) and subtracted	Clear edges

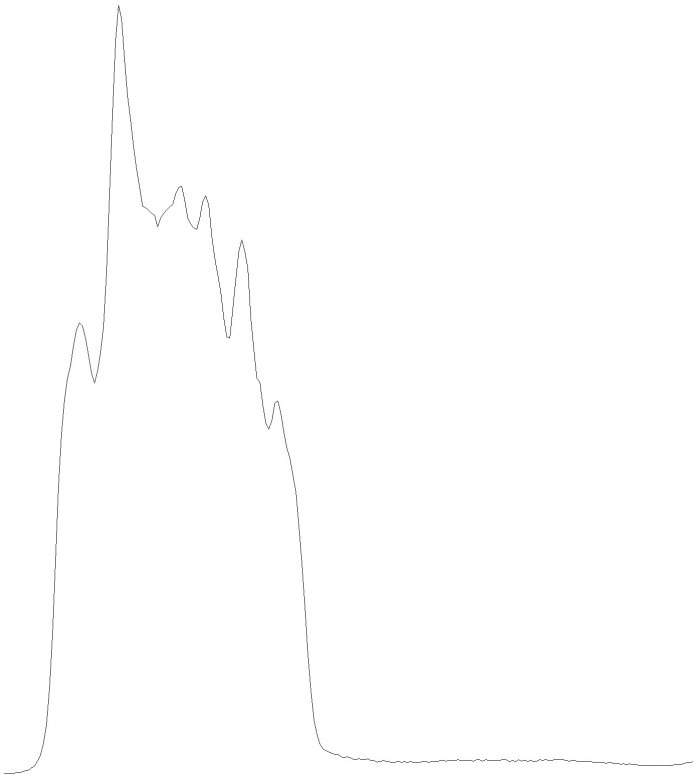
Morphology used	Characteristics
Dilate	Used to fill the holes. Kernel Size = 3
Erode	Used to erase salt and pepper. Kernel Size = 5
Open	Iteration = 2, Kernel Size = 3
Close	Iteration = 5, Kernel Size = 5

Each classes are detected with the following numbers:

```
The number of M6 Bolt: 3
The number of M5 Bolt: 6
The number of M5 HEX NUT: 5
The number of M5 RECT NUT: 2
The number of M6 HEX NUT: 4
The number of False Counts: 0
```

2. Procedure

Histogram Analysis

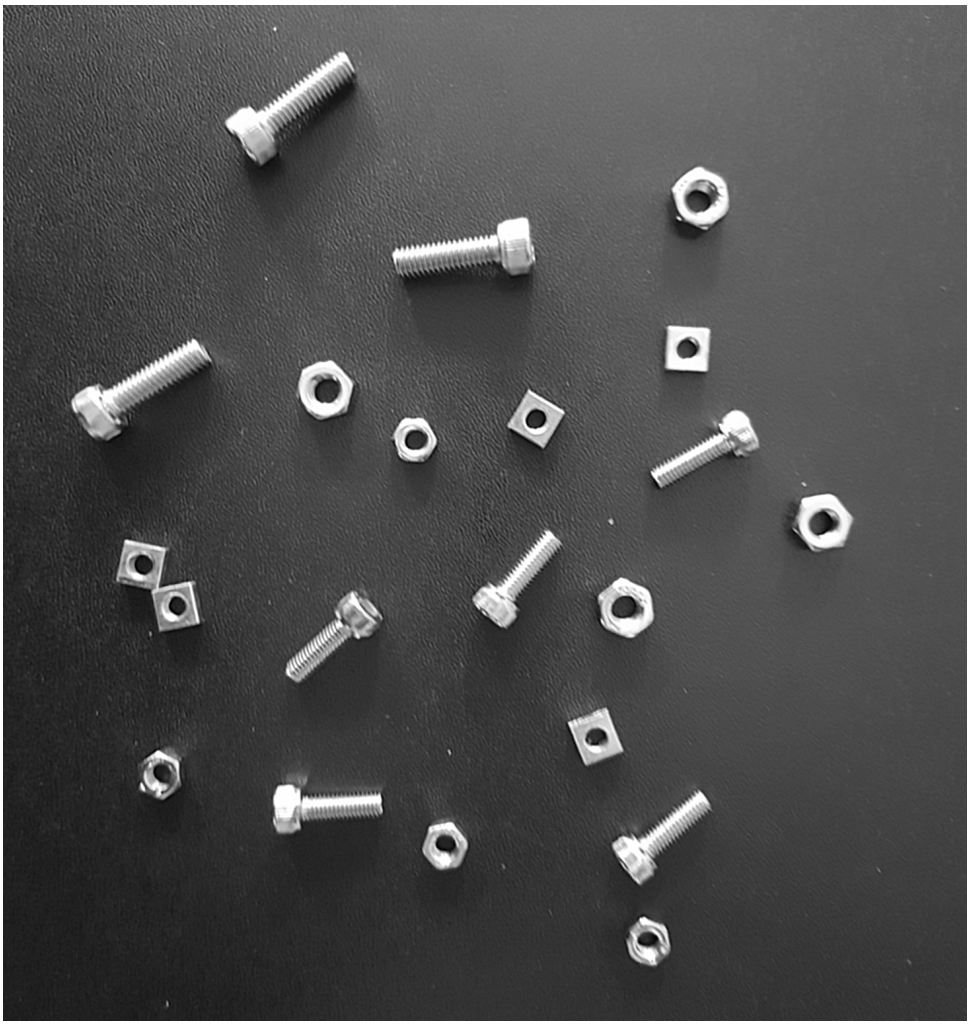


The figure above is a histogram of intensity of the original image. I do not choose to equalize histogram as it was harder to classify the components as the contrast at the right upper corner and the bottom left corner is different.

Filtering

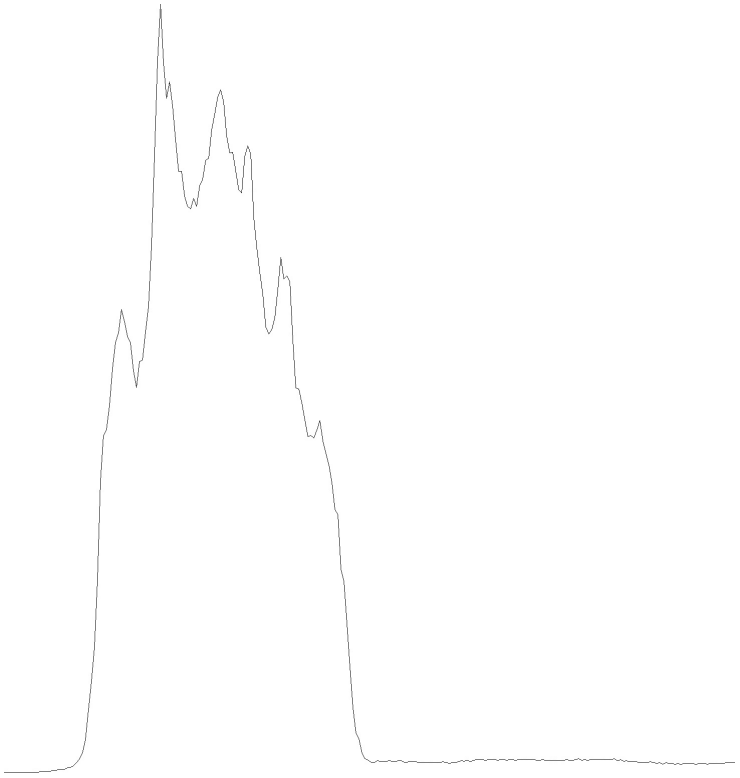
There are three layers of filters that the image goes through before Thresholding and Morphology. First filter is a median filter that filters out salt and pepper noises in the image. Then the extra smoothing filter(Gaussian) is used to filter extra noises that follows Gaussian noises. Then to clarify the edges of classes and the background, Laplacian filter is introduced. The weight 0.7 is applied before subtraction so that not many noises go into the output image.

After all, filtered image is as below:



Still there are some salt and pepper-ish noises in the image but the size of the noise was too big so not any filter could remove with keeping the edges sharp enough.

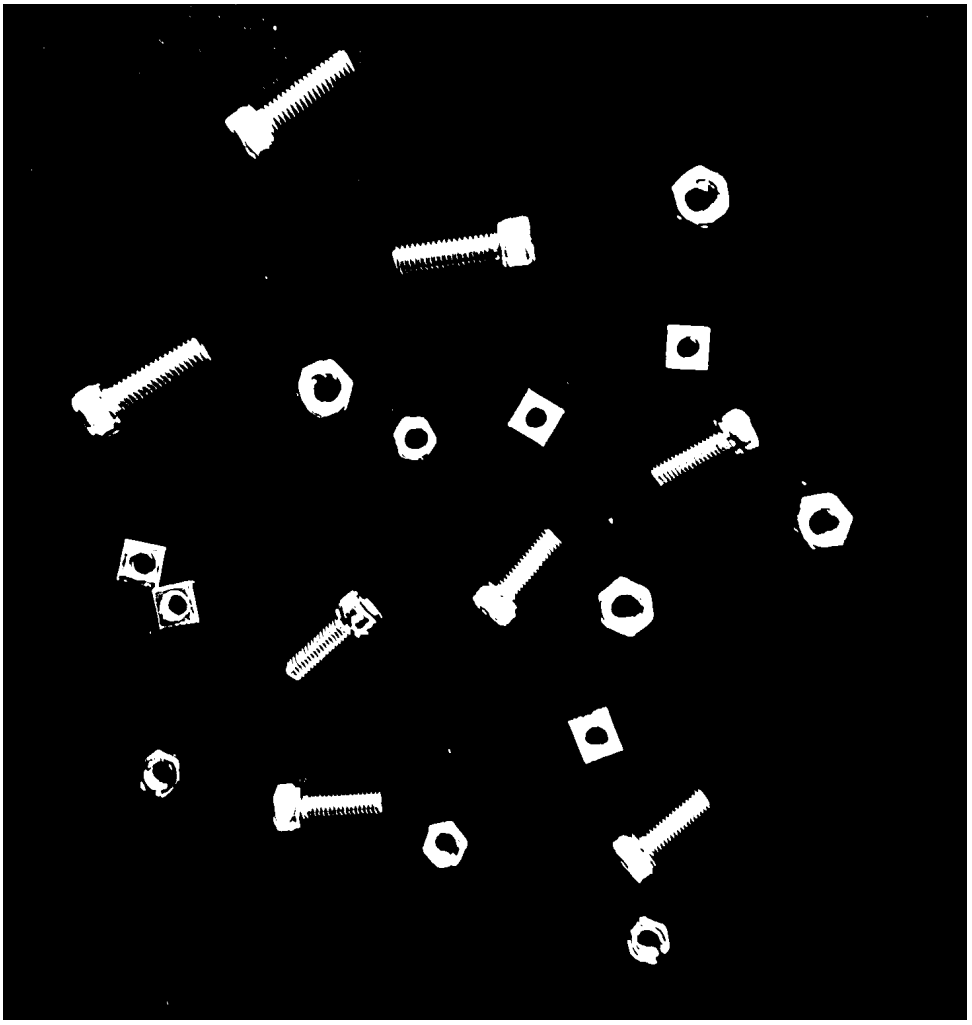
After filtering the histogram looks like this. The peaks become more distinctive with higher values compared to original histogram.



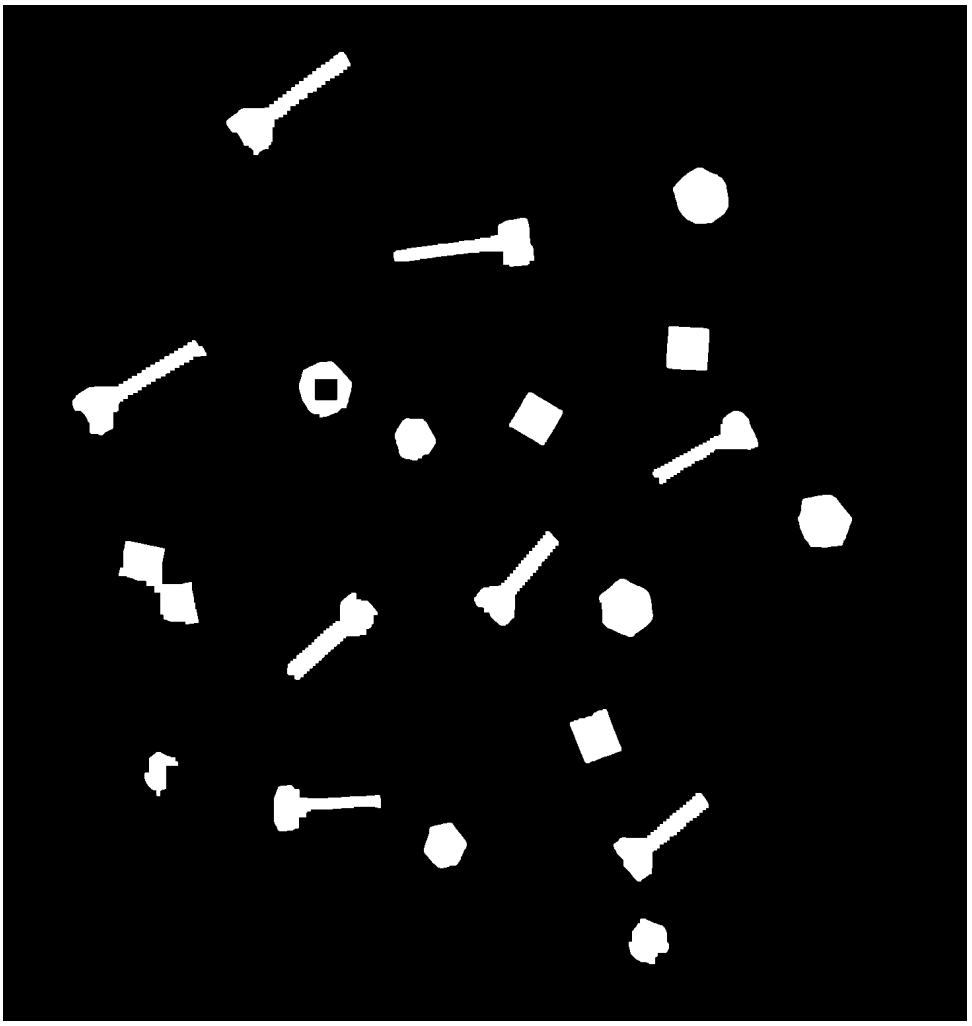
Thresholding and Morphology

The threshold value is considered in a quantitative manner. The threshold technique is Otsu method where the threshold value moves along the boundary of maximum to minimum threshold value which is known as the optimum global threshold value.

After **Thresholding** the image looks like below. There are still some noises in the picture but keeping the edges clear.



Afterwards, morphology has taken place. Removing the noises were done by erosion. Filling the hole was done by dilation. Smoothing was done by closing, opening(filtering out noise and smooth the contour lines). Dilation was taken too much the nuts do not have holes anymore.



I contoured the edges of the figure and detected each classes by their contour arc lengths. I removed the contoured inner rectangles using bitwise operations. Example code is the below:

```
    std::vector<bool> ignore_contour(contours.size(),false);
    cv::Rect FalseDetect[contourSize];
    /* Make contour rectangles */
    for(int idx = 0; idx<contourSize; idx++)    FalseDetect[idx] =
cv::boundingRect(contours[idx]);

    /* Find any rectangles that are inside another rectangles*/
    for(int rectA = 0 ; rectA<contourSize; rectA++){
        for(int rectB = rectA+1 ;rectB<contourSize; rectB++){
            if((FalseDetect[rectA]&FalseDetect[rectB]) == FalseDetect[rectA])
ignore_contour[rectA] = true;
            else if((FalseDetect[rectA]&FalseDetect[rectB])== FalseDetect[rectB])
ignore_contour[rectB] = true;
        }
    }
```

I skipped the ignore indices as follows:

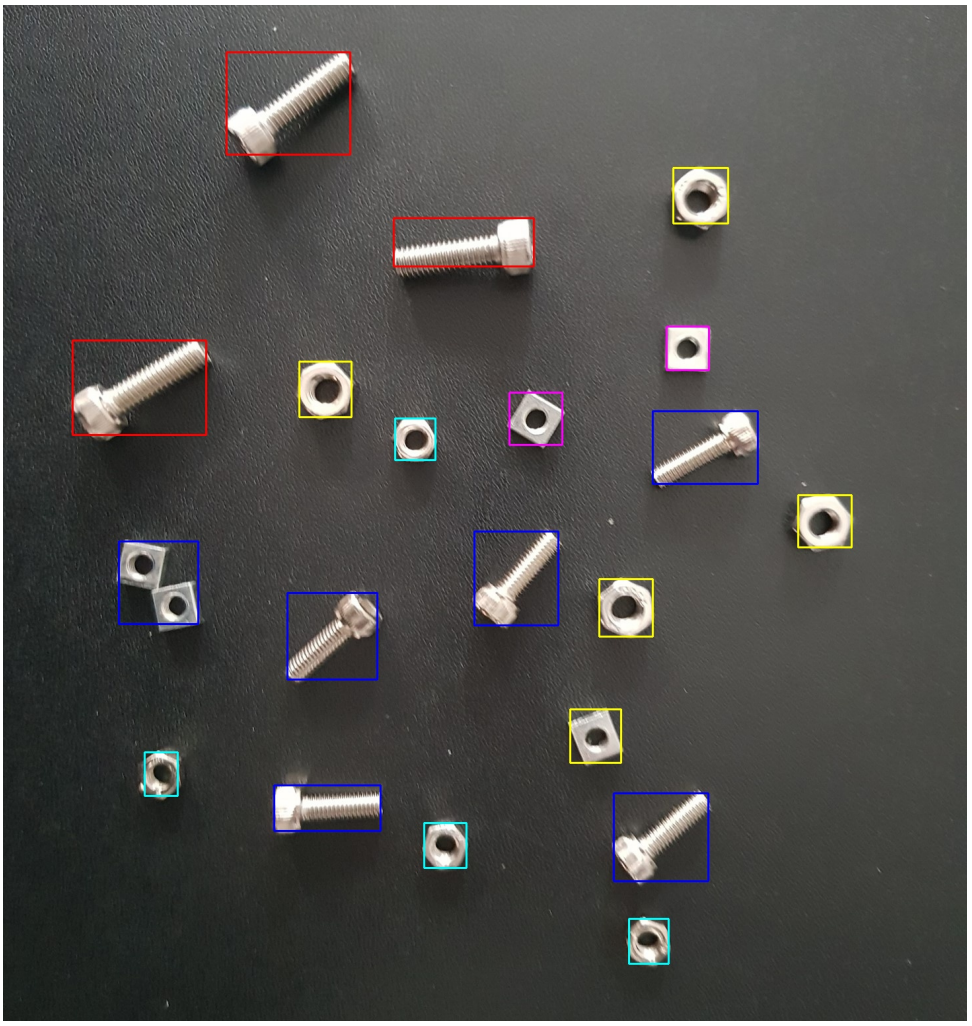
```
    /* Segment each rectangle depending on their size */
    for(int idx = 0; idx< contourSize; idx++){
        if(!ignore_contour[idx]){
            arc_length = cv::arcLength(contours[idx], true);
            cv::Rect bounding_rect = FalseDetect[idx];
            if(arc_length >= 500 && arc_length <= 700){
```

```
            Component_cnt[M6BOLT] ++;
            color = RED;
        }
        else if(arc_length >= 420 && arc_length < 500){
            Component_cnt[M5BOLT] ++;
            color = BLUE;
        }
        else if(arc_length >= 325 && arc_length < 400){
            False_cnt ++;
            color = GREEN;
        }
        else if(arc_length >= 262 && arc_length < 325){
            Component_cnt[M5NUT] ++;
            color = CYAN;
        }
        else if(arc_length >= 240 && arc_length < 262){
            Component_cnt[M5RECT] ++;
            color = PINK;
        }
        else{
            Component_cnt[M6NUT] ++;
            color = YELLOW;
        }
```

Result and Discussion

1. Final Result

The final result of the detection is the below. I was unable to distinguish in between the rect nuts at the left.



2. Discussion

The table below shows the accuracy of the detection.

Items	True	Estimated	Accuracy
M5 Bolt	5	6	80%
M6 Bolt	3	3	100%
M5 Hex Nut	4	4	100%
M5 Rect Nut	5	2	40%
M6 Hex Nut	4	5	80%

Since this project objective is to obtain a detection accuracy of 80% for each item, the proposed algorithm has achieved the project goal successfully with 80% of mean accuracy.

Conclusion

This project is done to figure out appropriate classic image processing techniques for correct image segmentation. Some of the followings are used to segment each bolts and nuts such as **Spatial Filtering**, **Morphology(Opening and Closing)**,**Contouring**, and **Thresholding**. The analyzing is done by histogram.

Appendix

`main.c`

```
/** @brief LAB 1 Grayscale Image Segmentation
* @author Jin Kwak/21900031
* @Created 2024/03/22
* @Modified 2024/04/01
*/

#include "DLIP.hpp"
#include <sstream>

#define MAX_BIN_VAL         (int)(255)
#define THRESH_VAL          (double)(180)
#define THRESH_VAL2         (double)(95)
#define KERNEL_SIZE3        (int)(3)
#define KERNEL_SIZE5        (int)(5)
#define KERNEL_SIZE7        (int)(7)
#define GaussianVar         (double)(1.5)
#define BLUE                (cv::Scalar(255,0,0))
#define RED                 (cv::Scalar(0,0,255))
#define GREEN               (cv::Scalar(0,255,0))
#define CYAN                (cv::Scalar(0,255,255))
#define PINK                (cv::Scalar(255,0,255))
#define YELLOW              (cv::Scalar(255,255,0))

/*Define Variables*/
cv::Mat src,src_gray,src_th;
cv::Mat dst_laplace;
cv::Mat dst_masked;
cv::Mat dst_morph;
cv::Mat dst_color;
// Contour Variables
std:: vector<std::vector<cv::Point>>  contours;
std::vector<cv::Vec4i> hierarchy;
cv::Scalar color;
int Component_cnt[5] = {0,};
int False_cnt = 0;
enum Components{
    M6BOLT = 0,
    M5BOLT = 1,
    M6NUT  = 2,
    M5NUT  = 3,
    M5RECT = 4
};

void func_Filter(void);
void func_Threshold(void);
void func_Morphology(void);
void func_Contour(void);

int main(){

    src = cv::imread("Lab_GrayScale_TestImage.jpg",cv::IMREAD_COLOR);
```

```
    cvtColor(src, src_gray, cv::COLOR_BGR2GRAY);

    func_Filter();

    func_Threshold();

    func_Morphology();

    func_Contour();

    image("Final",src_th);
    cv::waitKey(0);//Pause the program
    return 0;
}

void func_Filter(void){
    cv::medianBlur(src_gray,src_gray, KERNEL_SIZE5);

    cv::GaussianBlur(src_gray,src_gray,
                     cv::Size(KERNEL_SIZE7,KERNEL_SIZE7)
                     ,GaussianVar, GaussianVar);

    cv::Laplacian(src_gray,dst_laplace,CV_16S,
                  KERNEL_SIZE3,1,0,cv::BORDER_DEFAULT);

    dst_laplace. convertTo(dst_laplace,CV_16S);
    src_gray -= (0.7*dst_laplace);
    showGrayImgHist(src_gray);
}

void func_Threshold(void){
    cv::threshold(src_gray,src_th,THRESH_VAL,MAX_BIN_VAL,cv::THRESH_OTSU);
    cv::threshold(src_th,src_th,THRESH_VAL2,MAX_BIN_VAL,cv::THRESH_BINARY);
}

void func_Morphology(void){
    cv::dilate(src_th,src_th,cv::Mat::ones(KERNEL_SIZE3,KERNEL_SIZE3,CV_8UC1));
    cv::erode(src_th,src_th,cv::Mat::ones(KERNEL_SIZE5,KERNEL_SIZE5,CV_8UC1));
    cv::morphologyEx(src_th,src_th,cv::MORPH_OPEN,
                     cv::Mat::ones(KERNEL_SIZE3,KERNEL_SIZE3,
                                   CV_8UC1),cv::Point(-1,-1),2);
    cv::morphologyEx(src_th,src_th,cv::MORPH_CLOSE,
                     cv::Mat::ones(KERNEL_SIZE7,KERNEL_SIZE7,CV_8UC1),
                     cv::Point(-1,-1),5);
    dst_morph = src_th;
    image("Threshold and morphology",dst_morph);
}

void func_Contour(void){
    std::vector<cv::Scalar> colors(contours.size());
    cv::Mat contour(dst_morph.size(),CV_8U, cv::Scalar(255));
    cv::findContours(dst_morph,contours,
                     hierarchy,cv::RETR_CCOMP,cv::CHAIN_APPROX_SIMPLE);
    std::vector<bool> ignore_contour(contours.size(),false);
    cv::drawContours(contour,contours,
                     -1,cv::Scalar (0),cv::FILLED);
```

```
    int contourSize = contours.size();          //Number of Contours
    double arc_length =  0;                     //Length of the contour box
    cv::Rect FalseDetect[contourSize];
    /* Make contour rectangles */
    for(int idx = 0; idx<contourSize; idx++)    FalseDetect[idx] =
cv::boundingRect(contours[idx]);

    /* Find any rectangles that are inside another rectangles*/
    for(int rectA = 0 ; rectA<contourSize; rectA++){
        for(int rectB = rectA+1 ;rectB<contourSize; rectB++){
            if((FalseDetect[rectA]&FalseDetect[rectB]) == FalseDetect[rectA])
ignore_contour[rectA] = true;
            else if((FalseDetect[rectA]&FalseDetect[rectB])== FalseDetect[rectB])
ignore_contour[rectB] = true;
        }
    }

    /* Print out the number of contours*/
    std::cout<<"The number of detected Industrial components: "
<<contourSize<<std::endl;

    /* Segment each rectangle depending on their size */
    for(int idx = 0; idx< contourSize; idx++){
        if(!ignore_contour[idx]){
            arc_length = cv::arcLength(contours[idx], true);
            cv::Rect bounding_rect = FalseDetect[idx];
            if(arc_length >= 500 && arc_length <= 700){
                Component_cnt[M6BOLT] ++;
                color = RED;
            }
            else if(arc_length >= 420 && arc_length < 500){
                Component_cnt[M5BOLT] ++;
                color = BLUE;
            }
            else if(arc_length >= 325 && arc_length < 400){
                False_cnt ++;
                color = GREEN;
            }
            else if(arc_length >= 262 && arc_length < 325){
                Component_cnt[M5NUT] ++;
                color = CYAN;
            }
            else if(arc_length >= 240 && arc_length < 262){
                Component_cnt[M5RECT] ++;
                color = PINK;
            }
            else{
                Component_cnt[M6NUT] ++;
                color = YELLOW;
            }

            cv::rectangle(src, bounding_rect, color, 2, cv::LINE_8, 0);
        }
    }
    //Print the results
    std::cout<<"The number of M6 Bolt: "<<Component_cnt[M6BOLT]<<std::endl;
```

```
    std::cout<<"The number of M5 Bolt: "<<Component_cnt[M5BOLT]<<std::endl;
    std::cout<<"The number of M5 HEX NUT: "<<Component_cnt[M5NUT]<<std::endl;
    std::cout<<"The number of M5 RECT NUT: "<<Component_cnt[M5RECT]<<std::endl;
    std::cout<<"The number of M6 HEX NUT: "<<Component_cnt[M6NUT]<<std::endl;
    std::cout<<"The number of False Counts: "<<False_cnt<<std::endl;
    cv::imshow("Contoured",src);
}
```

From source `DLIP.c`, I have defined various functions that uses OpenCV functions with comfortability. Some functions that I used in the LAB are as below:

image()

Displays an image with a name. `namedWindow()` and `imshow()`

```
void image(cv::String str, cv::Mat mat);
```

> Parameters:
>
> - str: Window Name
> - mat: Image source

showGrayImgHist()

Show histogram of the source image in a white background

```
void showGrayImgHist(cv::Mat src);
```

> Parameters:
>
> - src: Source Image

Troubleshooting

- One method to distinguish each classes is to compare corners or holes of nuts. However, my algorithm has low resolution and some of the holes are filled so I could not use this method.
- If clear morphology is taken place, it can be appropriate to use convolution(correlation) filter and find out which pixel points have the highest value of the convolution filter.
- The one should be able to appropriately use thresholding and morphology. The light intensity was different at the top right and bottom left. Another solution to the image could be adaptive thresholding or may be