LAB: Grayscale Image Segmentation

Date: 2024-Mar-30

Author: Taegeon Han 21900793

Introduction

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

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

- Bolt M5
- Bolt M6
- Hex Nut M5
- Hex Nut M6
- Rect Nut M5

2. Preparation

Software Installation

- OpenCV 4.9.0 with Visual Studio 2022

Dataset

This is a picture of the bolts and nuts.

Dataset link: Download the test image

Algorithm

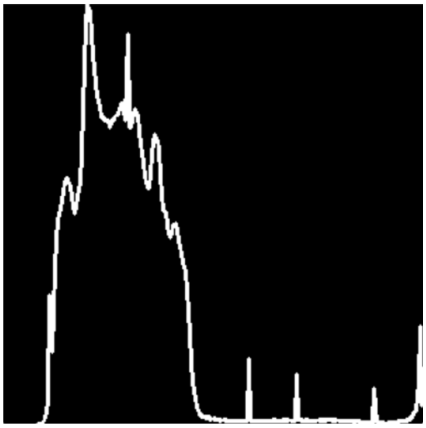
1. Overview

Objects are identified and classified from one image to derive results.

2. Procedure

Load Image & Histogram

The picture on the left is a picture to be analyzed, and the right is a histogram picture of it. The input image is analyzed with a histogram to understand the distribution of intensity values. As seen in the histogram on the right picture, the bright component of objects can be segmented from mostly dark backgrounds.



Apply Filters

Filtering was performed to classify and detect objects in the image. This step is important to improve the quality of the image and to detect the object well.

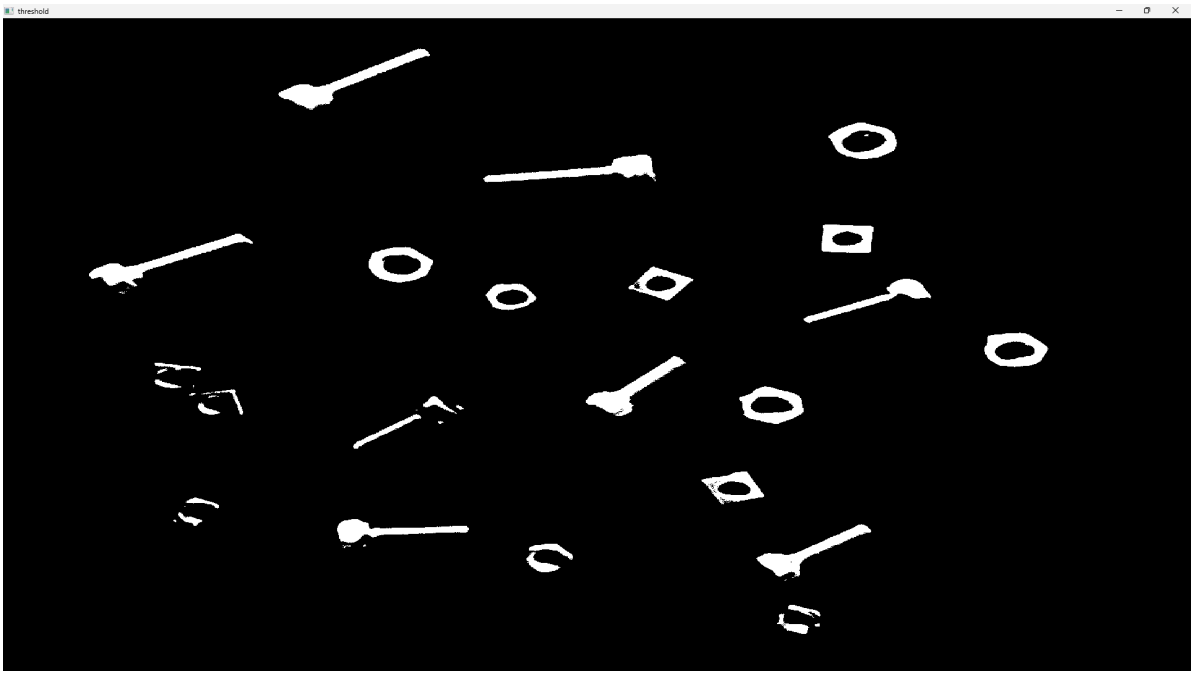
Gaussian Blur & Laplacian

Before Threshold, we aimed to diminish noise and make edge clearly. Gaussianblur reduces noise and makes the image smooth. Gaussianblur enables smooth deformation without affecting the edges of bolts and nuts too much. The larger kernel size, the blur effect spreads out more widely, making the image look much smoother. In images with gradation changes, such as shadows and highlights, it transitions smoothly throughout the image. Therefore, the kernel size is set to be large at 19.

It emphasizes the boundaries within the blurred image through the Laplacian filter. In other words, it makes the outline of the object clearer. In order to minimize noise and preserve the edge of the image during the Laplacian filter process, the kernel size was set to 3.

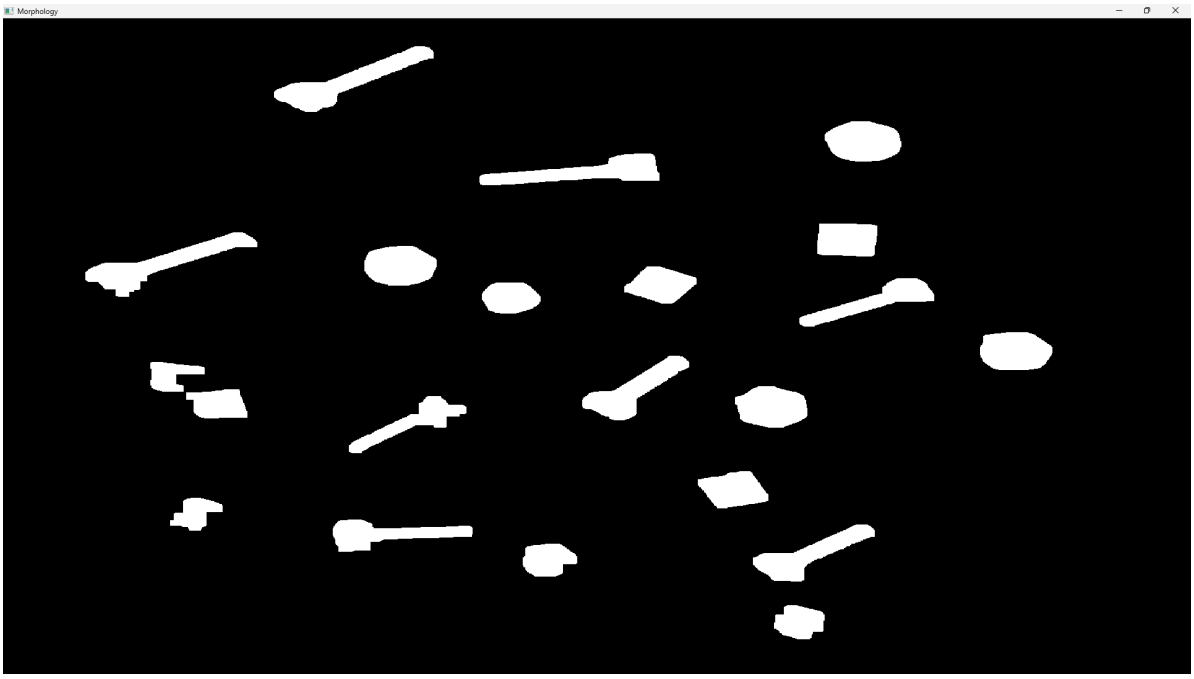
Thresholding

As can be seen from the left part of the image, the threshold value was set high to separate the attached square nut. The attached square nut showed better separation from the darker background. Therefore, the threshold value was set to 183. In addition, in order to separate the background and the object well, the type was set to Threshold_binary.



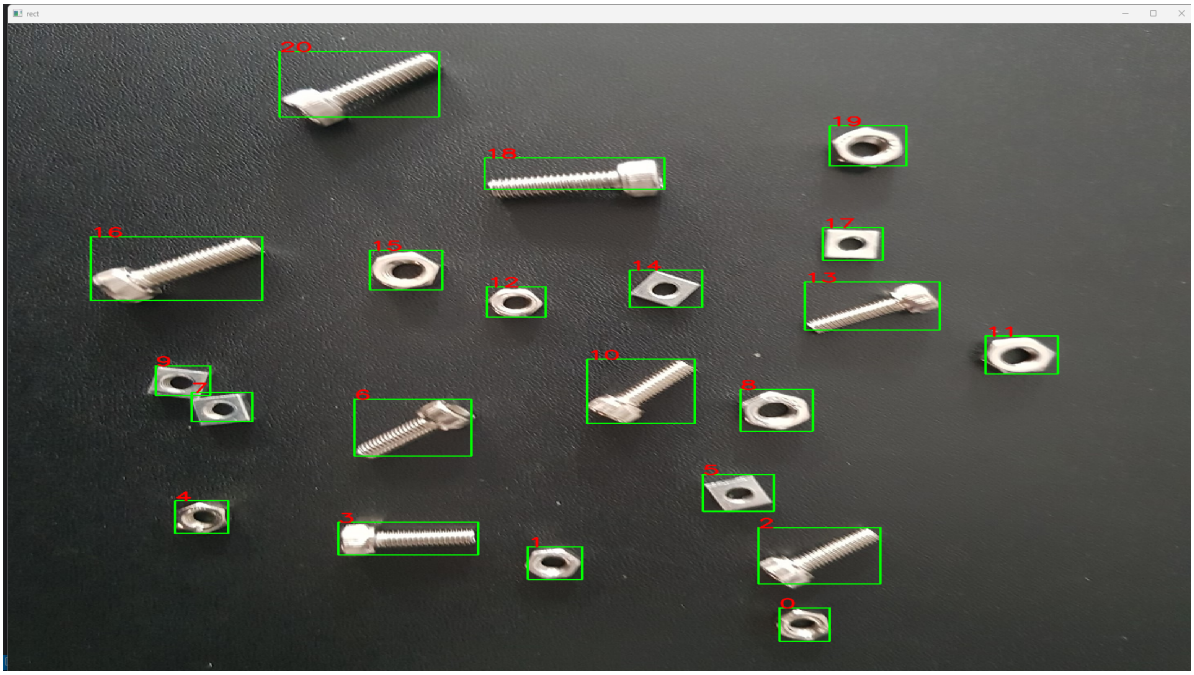
Apply morphological operations

The structural elements were divided into five 3, 5, 7, 9, and 11, and used to classify the objects as desired. After that, dilate and erode were repeated to fill the hole of the object, and white dots appearing in the background were removed. In particular, morphology played an important role in distinguishing between two attached square nuts.



Detecting and Classifying objects contours

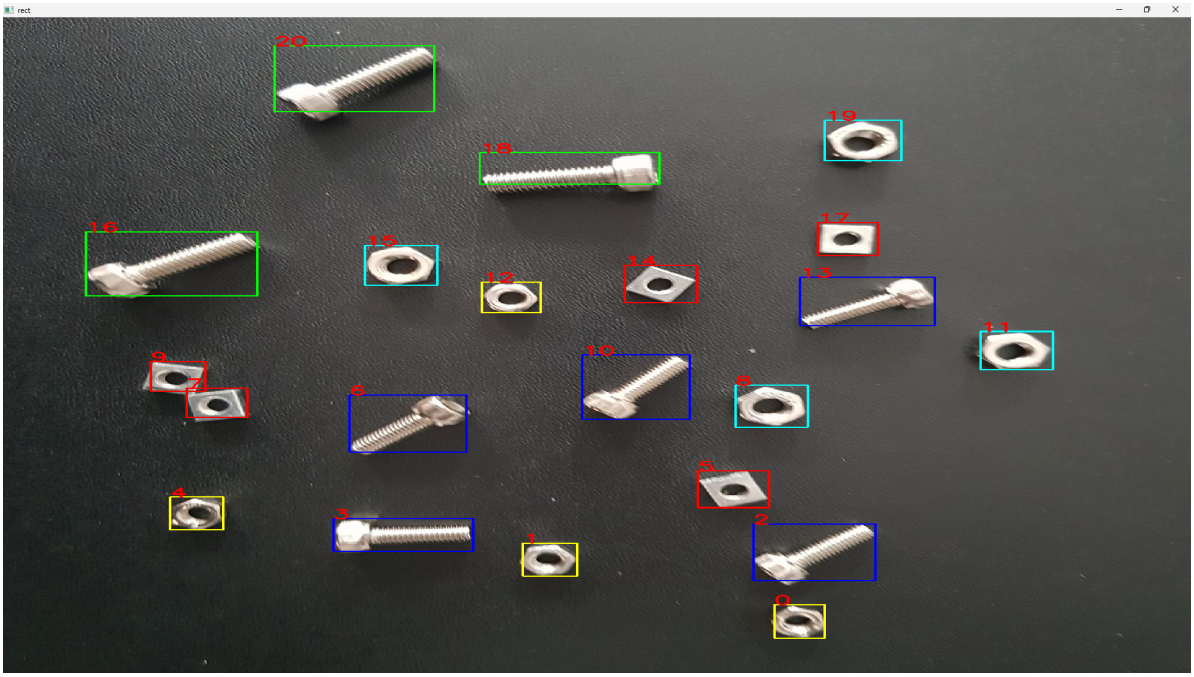
Contours are used to detect and outline various shapes within an image. Objects were recognized as rectangles, and numbers were assigned for each square.

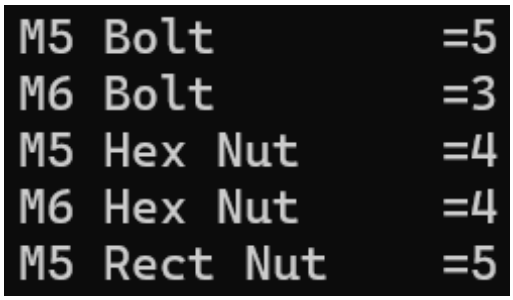


Result and Discussion

1. Final Result

The result of the mechanical part segmentation is displayed with the contour box. Also, the output of the number of nuts and bolts is displayed. You can see the picture below. It was classified by length. Also, the colors are marked differently according to the type of object.





2. Discussion

The table below shows the accuracy of object recognition.

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

Through the procedure, all objects could be separated. At first, it was tried to be classified according to the area of the object, but it failed, so types were classified by color according to length, and as a result of printing, 100% accuracy was shown. 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

The image was imported and object recognition was enabled through filtering, morphology, and contours. The goal was to count the number of nuts and boulders. Although it took time to separate because there was an attached object, it eventually allowed it to be detected separately, and the result was successful with 100% accuracy as shown in Discussion. Count the number of nuts & bolts of each size for a smart factory automation

Troubleshooting

In the image, it was challenging to distinguish the region where the square nut was connected. However, this separation was achievable by fine-tuning the threshold value, kernel size, and morphology. Given that blurring was initially applied to generate the outcome, the blur value had a substantial influence on all subsequent processing steps. Despite successfully separating the two nuts, there were instances where the bolt's body vanished, leaving only the head. Therefore, it is important to control the values used for object recognition.

Appendix

```
/*
 * DLIP_Tutorial_Contour.cpp
 * author: Taegeon Han
 * Date: 2024-03-30
 * DLIP_LAB1_Grayscale Image Segmentation
 * Accurate recognition of objects in an image using a variety of methods
 */


#include <iostream>
#include <opencv2/opencv.hpp>
#include <math.h>

using namespace std;
using namespace cv;

// dst: binary image
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;


Mat src, src_color, graycolor, dst, dst2, dst_morph, result_laplcaian;

// Threshold
int threshold_type = 0;
int threshold_value = 183;
int const max_value = 225;
int const max_type = 0;
```

```
int const max_BINARY_value = 255;
// Contours
int countBN[5] = {0};

void img_read(void);
void Threshold_Demo(int, void*);
void laplac(Mat& m1, Mat& m2, Mat& m3, int kernel_size);
void plotHist(Mat src, string plotname, int width, int height);
void morph(Mat& m1, Mat& m2, int element_size);
void Contours(int counts[]);
void contours_parents(const vector<Scalar>& colors);


int main() {

    img_read();

    /* Filter */
    int i = 19;
    GaussianBlur(src, dst, cv::Size(i, i), 0, 0);
    namedWindow("gaussian", WINDOW_NORMAL);
    imshow("gaussian", dst);

    laplac(dst, dst2, result_laplcaian, 3);


    /* Threshold */
    threshold(result_laplcaian, dst, threshold_value, max_BINARY_value,
max_type);
    namedWindow("threshold", WINDOW_NORMAL);
    imshow("threshold", dst);

    /* Morphology */
    int element_size = 3;
    morph(dst, dst, element_size);

    /* Contours */
    Contours(countBN);

    /* Print number of objects */
    printf(" M5 Bolt \t=%d \n", countBN[1]);
    printf(" M6 Bolt \t=%d \n", countBN[0]);
    printf(" M5 Hex Nut \t=%d \n", countBN[2]);
    printf(" M6 Hex Nut \t=%d \n", countBN[4]);
    printf(" M5 Rect Nut \t=%d \n", countBN[3]);

    /* Histogram */
    cvtColor(src_color, graycolor, COLOR_BGR2GRAY);
    plotHist(graycolor, "Rect Hist", 300, 300);


    waitKey(0);

    return 0;
}


void img_read(void) {
```

```
    src_color = imread("Lab_GrayScale_TestImage.jpg", IMREAD_COLOR);   // read
image file

    // Convert the image to Gray
    cvtColor(src_color, src, COLOR_BGR2GRAY);

    namedWindow("Original", WINDOW_NORMAL);
    imshow("Original", src_color);

    if (src.empty())
    {
        cout << "Error: Couldn't open the image." << endl;
        waitKey(0);
    }
}

void laplac(Mat& m1, Mat& m2, Mat& m3, int kernel_size) {
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;

    cv::Laplacian(m1, m2, ddepth, kernel_size, scale, delta,
cv::BORDER_DEFAULT);
    m1.convertTo(m1, CV_16S);
    m3 = m1 - m2;
    m3.convertTo(m3, CV_8U);
    namedWindow("Laplacian", WINDOW_NORMAL);
    imshow("Laplacian", m3);
}

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);   // Calcultate
the histogram of image

    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(0));
    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(255), 2, 8, 0);
    }   // line : Draw values of histogram to stick in plot

    imshow(plotname, histImage);
}
```

```
void morph(Mat& m1, Mat& m2, int n) {
    int element_shape = MORPH_RECT;      // MORPH_RECT, MORPH_ELIPSE,
MORPH_CROSS
    n = 3;
    Mat element = getStructuringElement(element_shape, Size(n, n));
    Mat element1 = getStructuringElement(element_shape, Size(n + 2, n + 2));
    Mat element2 = getStructuringElement(element_shape, Size(n + 4, n + 4));
    Mat element3 = getStructuringElement(element_shape, Size(n + 6, n + 6));
    Mat element4 = getStructuringElement(element_shape, Size(n + 8, n + 8));


    dilate(dst, dst, element2);
    erode(dst, dst, element3);

    for (int i = 0; i < 4; i++) {
        dilate(dst, dst, element3);
    }

    erode(dst, dst, element3);
    for (int i = 0; i < 2; i++) {
        morphologyEx(dst, dst, MORPH_CLOSE, element4);
    }

    //dilate(dst, dst, element1);
    for (int i = 0; i < 3; i++) {
        erode(dst, dst, element4);
        dilate(dst, dst, element2);
    }


    namedWindow("Morphology", WINDOW_NORMAL);
    imshow("Morphology", dst);
}

void Contours(int counts[]) {
    /// Find contours
    findContours(dst, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE);
    vector<Scalar> colors(contours.size()); // 각 컨투어에 할당된 색상을 저장할 벡터

    // Draw all contours excluding holes
    Mat drawing(dst.size(), CV_8U, Scalar(255));
    drawContours(drawing, contours, -1, Scalar(0), FILLED, 8, hierarchy);

    cout << " Number of coins are =" << contours.size() << endl;


    for (int i = 0; i < contours.size(); i++)
    {
        printf(" * Contour[%d] -  Area OpenCV: %.2f - Length: %.2f \n", i,
contourArea(contours[i]), arcLength(contours[i], true));

        //double area = contourArea(contours[i]);
        double leng = arcLength(contours[i], true);

        int fontFace = FONT_HERSHEY_SIMPLEX;
        double fontScale = 1;
        int thickness = 2;
```

```
        Scalar color;

        bool shouldLabel = false;

        Rect boundRect = boundingRect(contours[i]);


        // 길이에 따른 사각형 색 분류
        if (leng >= 500) {
            color = Scalar(0, 255, 0);
            counts[0]++;
        }
        else if (leng >= 400 && leng < 500)
        {
            color = Scalar(255, 0, 0);
            counts[1]++;
        }
        else if (leng >= 300 && leng < 400)
        {
            color = Scalar(255, 255, 0);
            counts[2]++;
        }
        else if (leng < 300 && leng > 260)
        {
            color = Scalar(0, 0, 255);
            counts[3]++;
        }
        else {
            color = Scalar(0, 255, 255);
            counts[4]++;
        }

        colors[i] = color;

        rectangle(src_color, boundRect.tl(), boundRect.br(), color, 2);

        // 사각형 안에 숫자를 추가합니다.
        string label = to_string(i);
        Point textOrg(boundRect.x, boundRect.y);
        putText(src_color, label, textOrg, FONT_HERSHEY_SIMPLEX, fontScale,
Scalar(0, 0, 255), thickness, LINE_AA);
    }

    //contours_parents(colors);

    namedWindow("rect", WINDOW_NORMAL);
    imshow("rect", src_color);
}


/* 컨투어 안에 컨투어 색상 같게 하기 */
void contours_parents(const vector<Scalar>& colors) {
    // colors 벡터를 사용하여 모든 중첩된 사각형에 대한 색상을 조정합니다.
    for (int i = 0; i < contours.size(); i++) {
        double area = contourArea(contours[i]);
        // 컨투어의 부모가 있는지 확인합니다.
        int parentIdx = hierarchy[i][3];
        if (parentIdx != -1) {   // 부모 윤곽선이 있는 경우에만 실행
```

```
            // 컨투어의 부모에 이미 색상이 할당되어 있는지 확인하고 할당된 색상을 사용합니
다.
            Scalar parentColor = colors[parentIdx];
            Rect boundRect = boundingRect(contours[i]);
            if (area > 300 && area < 1000)
                rectangle(src_color, boundRect.tl(), boundRect.br(),
parentColor, 2);
        }
    }
}
```