Automated Optical Inspection

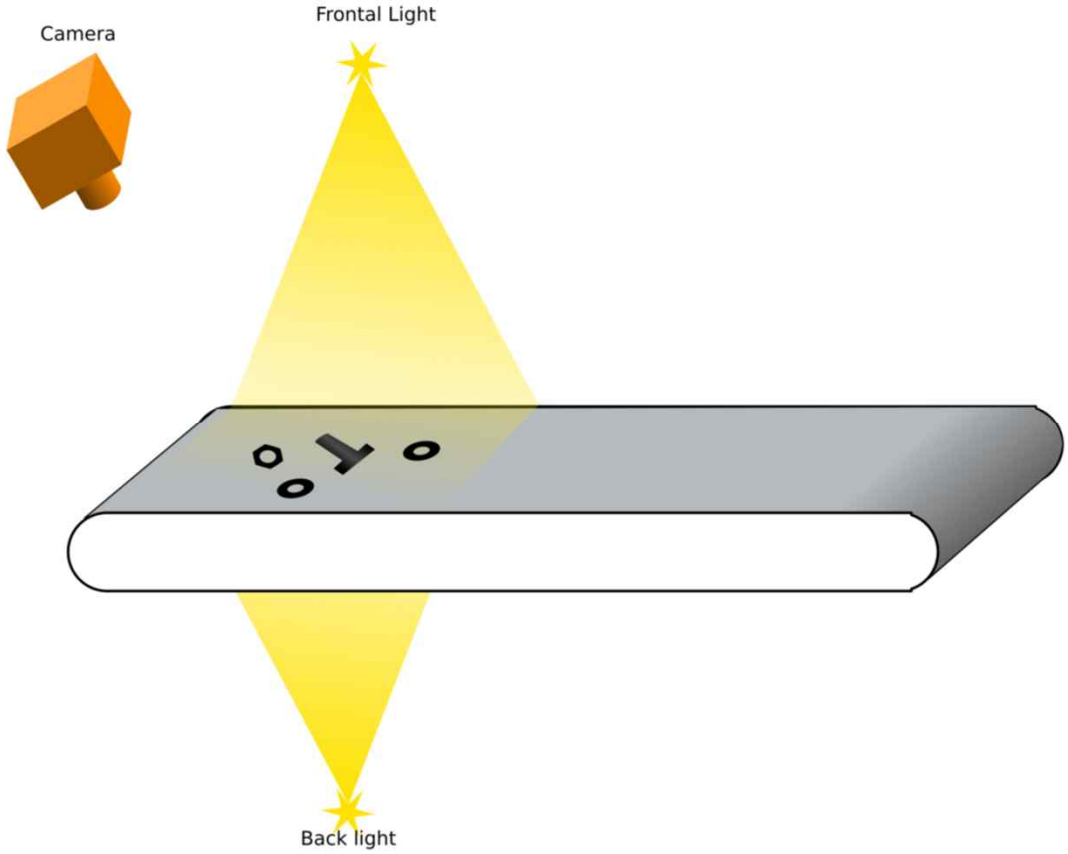
Object Segmentation and Detection

Learn OpenCV4 by Building Projects (2nd ed.)

https://*github.com/PacktPublishing/Learn-OpenCV-4-By-Building-Projects-Second-Edition/tree/master/Chapter_05*

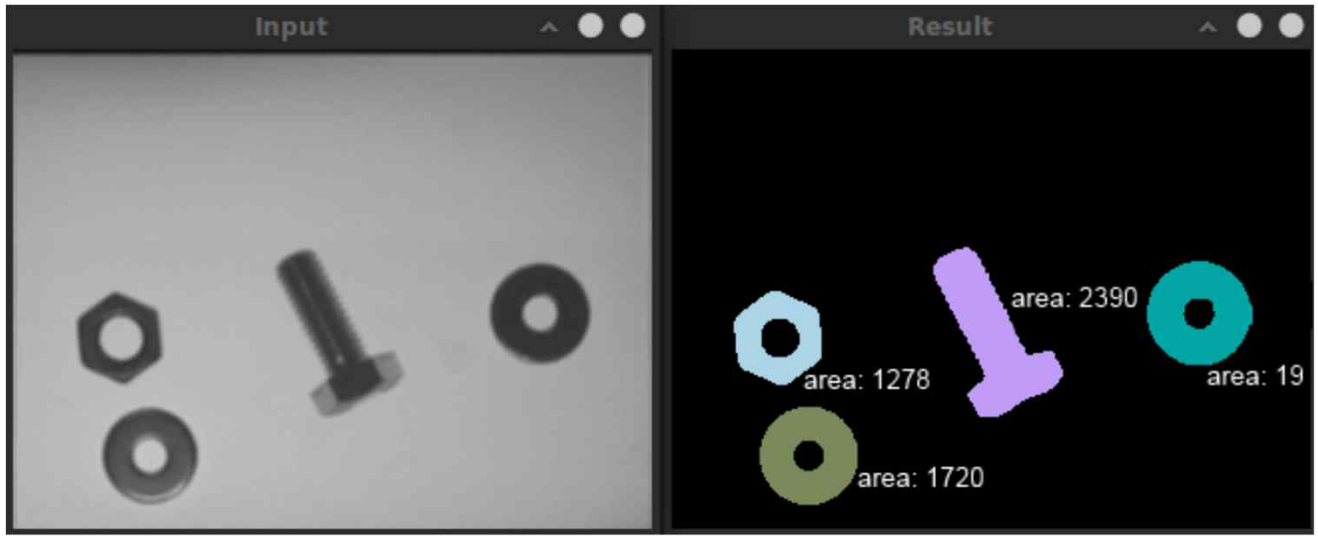
AOI

- Automated optical inspection



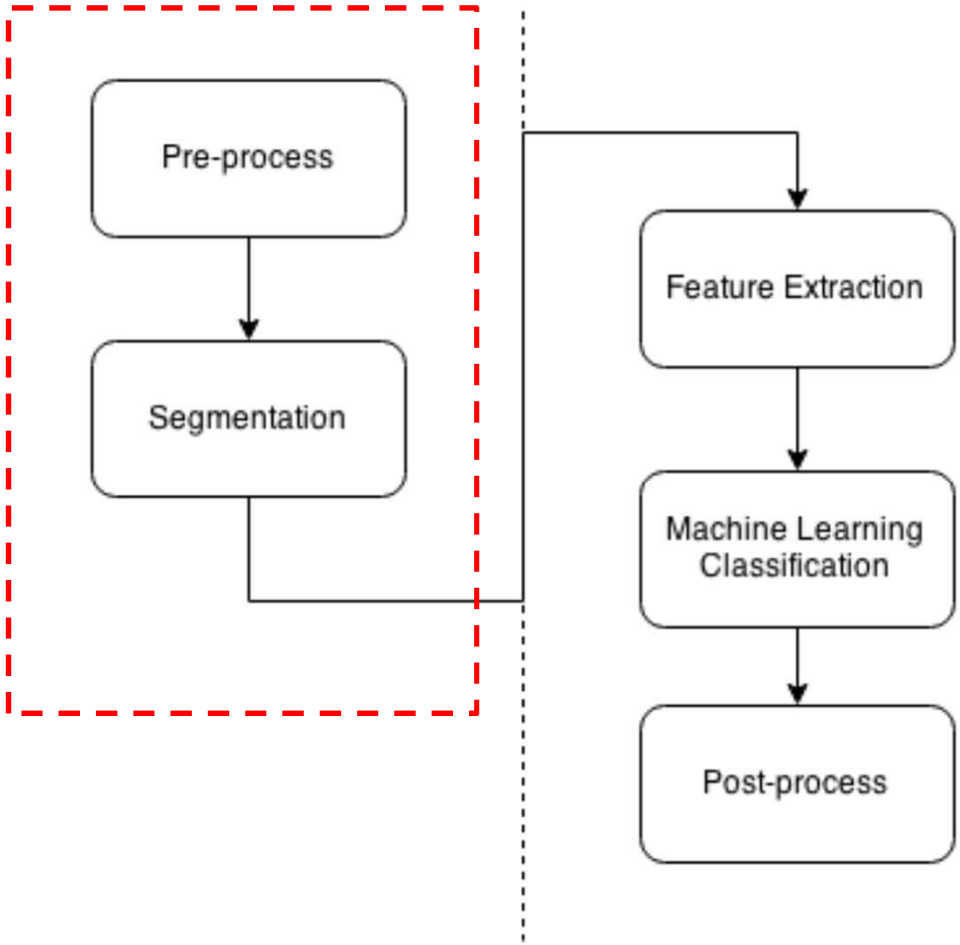
AOI

- Input & Output



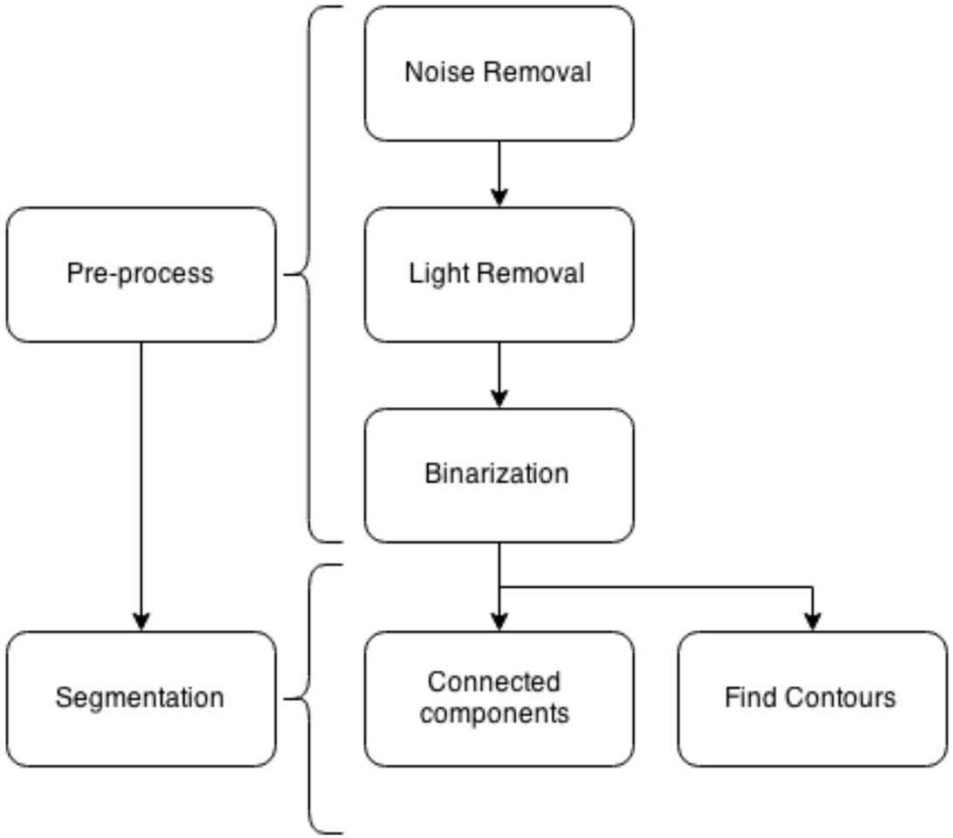
AOI

- Algorithm



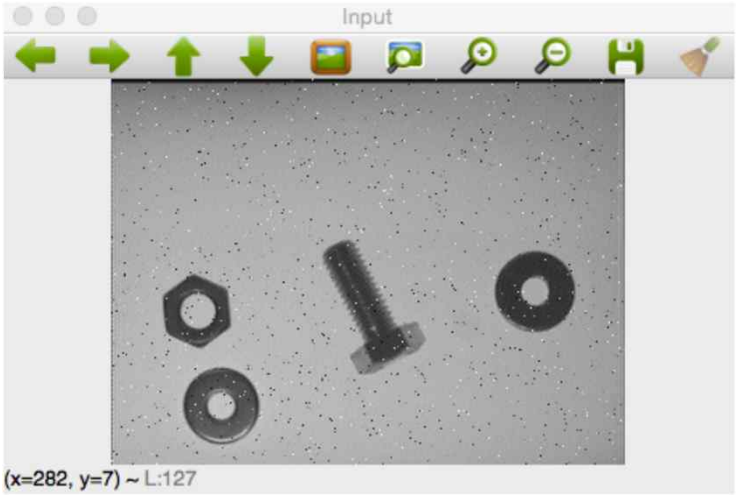
AOI

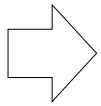
- Pre-process & Segmentation

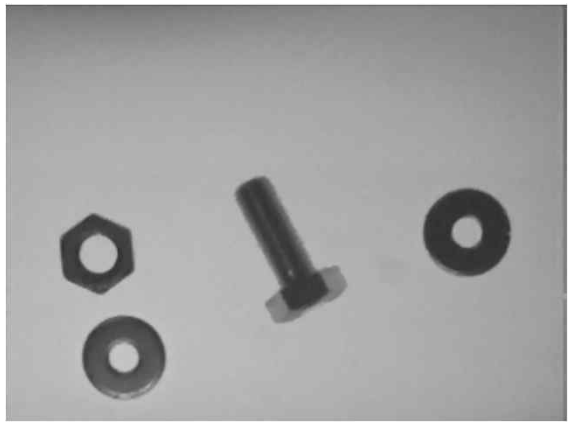


Noise Removal

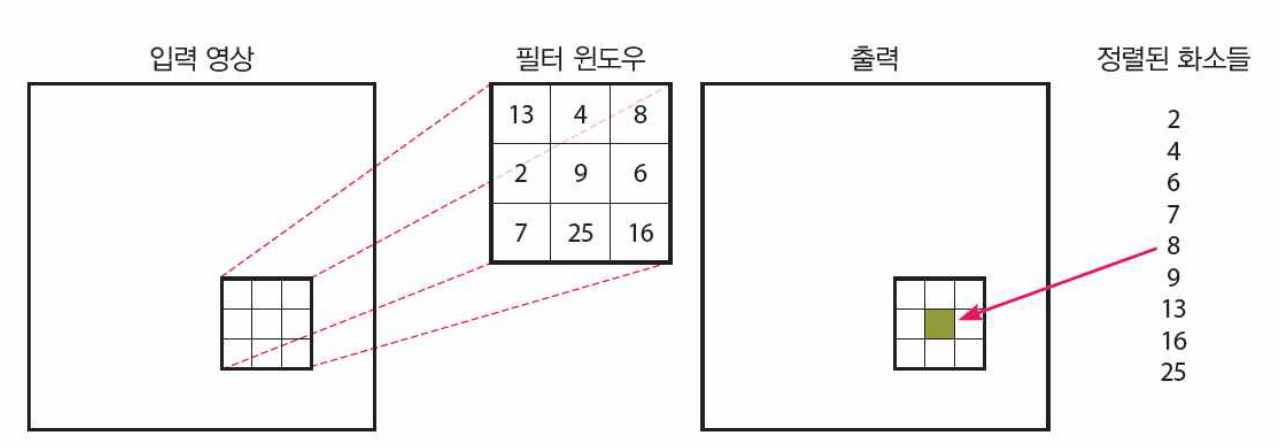
- Median filtering







Salt or pepper noise



Noise Removal

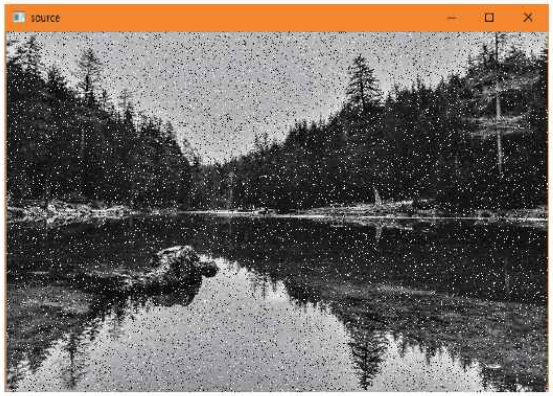
- OpenCV 함수

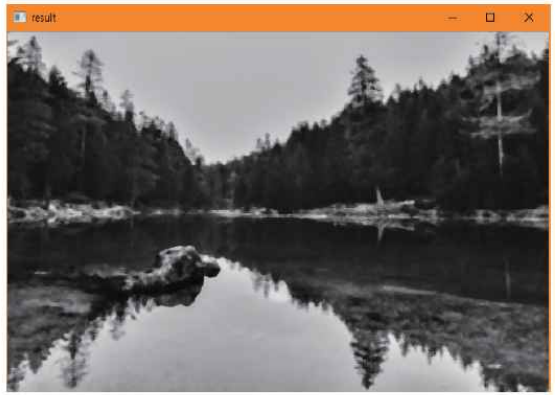
```
void medianBlur(InputArray src, OutputArray dst, int ksize);
```

매개 변수	설명
src	입력 영상으로서 1, 3, 4 채널이 가능하다. 영상 깊이는 CV_8U, CV_16U or CV_32F가 가능하다.
dst	출력 영상
ksize	윈도우의 크기. 홀수만 가능하다(예를 들어 3, 5, 7…)

Noise Removal

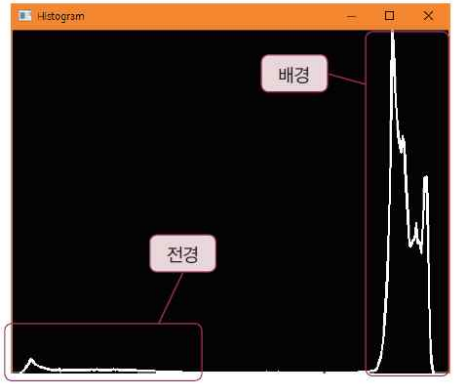
```
int main()
{
    Mat src = imread("d:/city1.jpg", IMREAD_GRAYSCALE);
    if (src.empty()) { return -1; }
    Mat dst;

    Mat noise_img = Mat::zeros(src.rows, src.cols, CV_8U);
    randu(noise_img, 0, 255);                // noise_img 의 모든 화소를 0 부터 255 까지의 난수로 채움

    Mat black_img = noise_img < 10;    // noise_img 의 화소값이 10 보다 작으면 1이되는 black_img 생성
    Mat white_img = noise_img > 245;  // noise_img 의 화소값이 245 보다 크면 1이되는 white_img 생성

    Mat src1 = src.clone();
    src1.setTo(255, white_img);   // white_img 의 화소값이 1 이면 src1 화소값을 255 로 한다=> salt noise
    src1.setTo(0, black_img);     // black_img 의 화소값이 1 이면 src1 화소값을 0 으로 한다=> pepper noise
    medianBlur(src1, dst, 5);
    imshow("source", src1);
    imshow("result", dst);

    waitKey(0);
    return 0;
}
```



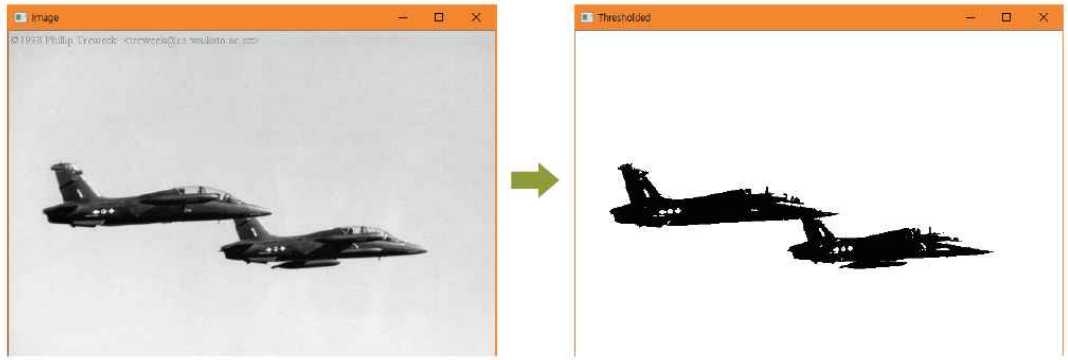


Light Removal

- Thresholding
  - Histogram 을 사용한 전경과 배경의 분리

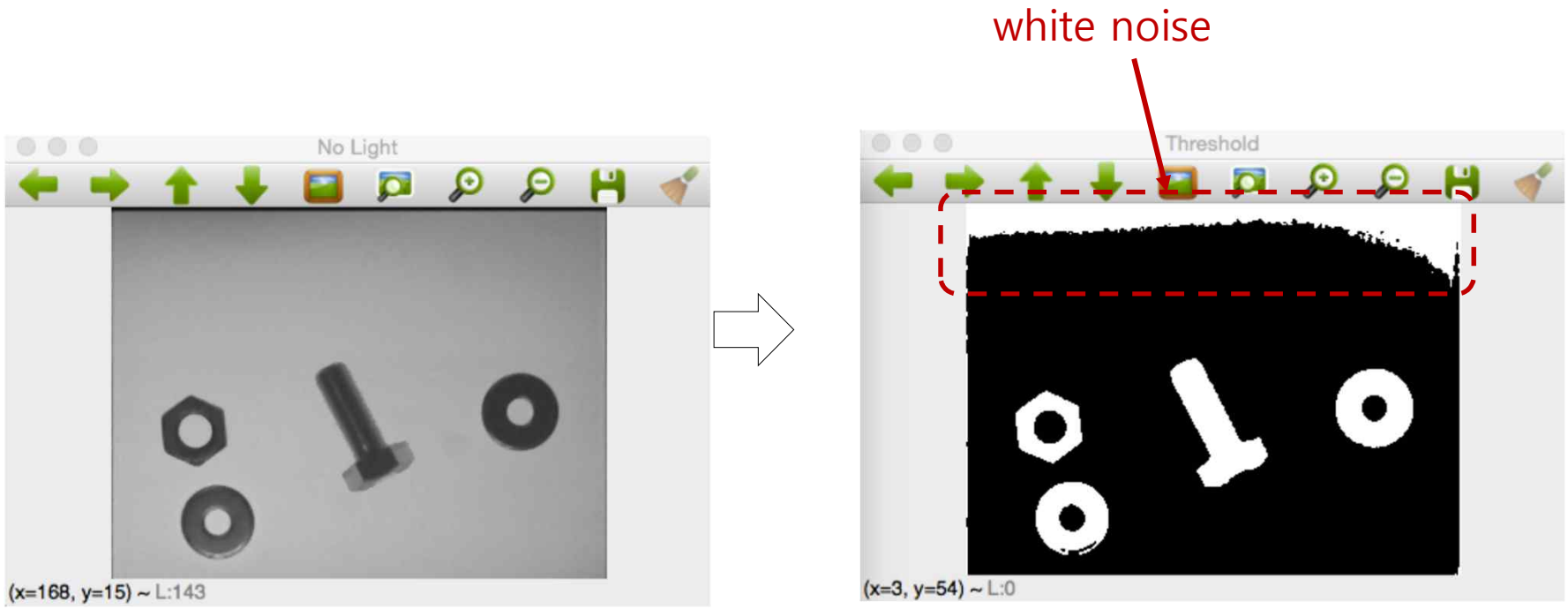



```
Mat src, threshold_image;
src = imread("d:/plane.jpg", IMREAD_GRAYSCALE);
threshold(src, threshold_image, 100, 255, THRESH_BINARY);
```



Light Removal

- Thresholding 의 어려움



Light Removal

- Light pattern
  - 동일한 light 환경에서 object 가 없는 영상
  - AOI 와 같이 동일한 환경에서 영상 획득이 가능한 경우, 별도 획득하여 사용

Light pattern

Light Removal

- Calculate light pattern
  - Light pattern 영상이 없는 경우 원 영상으로 부터 생성

```
Mat calculateLightPattern(Mat img)
{
   Mat pattern;
   // Basic and effective way to calculate the light pattern from one image
   blur(img, pattern, Size(img.cols/3,img.cols/3));
   return pattern;
}
```

Averaging or smoothing

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

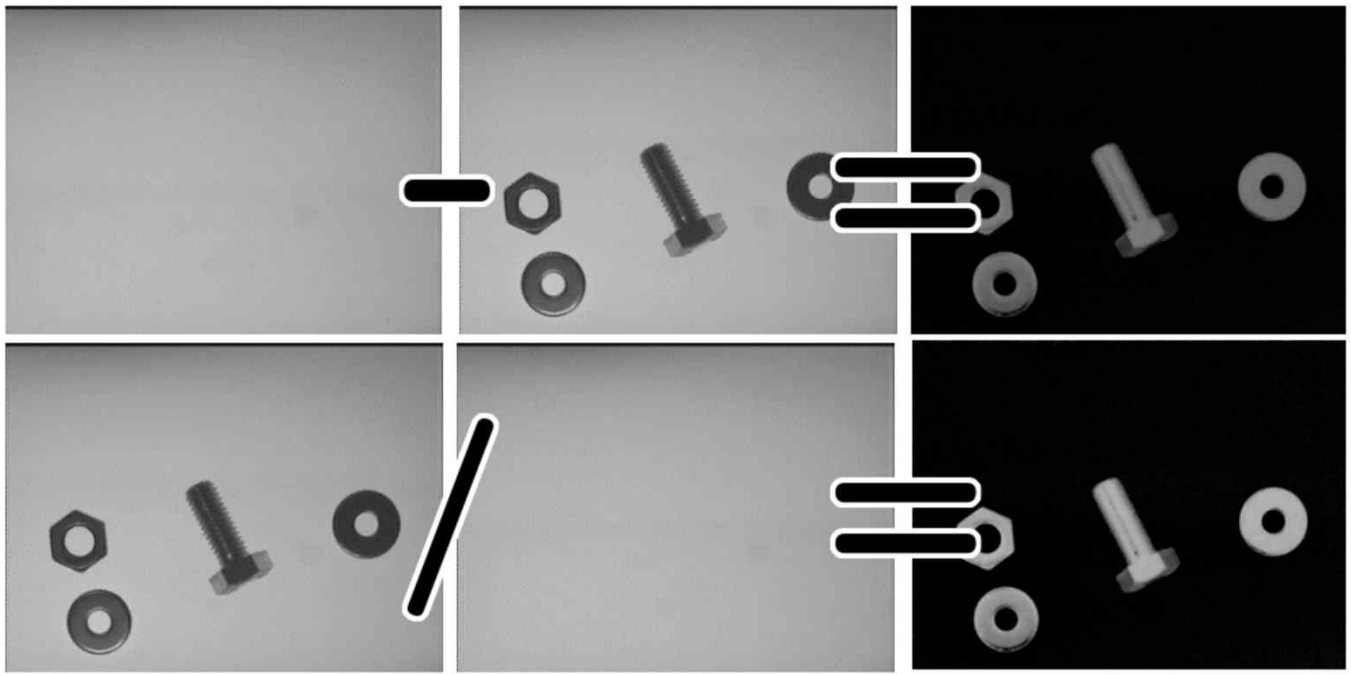
계산된 light pattern　　light pattern (GT)

- 계산 pattern 과 GT pattern 차이 있으나, 후속 처리에 큰 영향 없음

Light Removal

- Light removal algorithm
  1) R = L – I
  2) R = 255 * (1- (I/L))

  I : image, L : light pattern



Light Removal

```
Mat removeLight(Mat img, Mat pattern, int method)
{
    Mat aux;
    // if method is normalization
    if(method==1)
    {
        // Require change our image to 32 float for division
        Mat img32, pattern32;
        img.convertTo(img32, CV_32F);
        pattern.convertTo(pattern32, CV_32F);
        // Divide the imabe by the pattern
        aux= 1-(img32/pattern32);
        // Convert 8 bits format
        aux.convertTo(aux, CV_8U, 255);
    }else{
        aux= pattern-img;
    }
        //equalizeHist( aux, aux );
    return aux;
}
```

Binarization

- Thresholding

```
Mat img_thr;
if(method_light!=2){
    threshold(img_no_light, img_thr, 30, 255, THRESH_BINARY);
}else{
    threshold(img_no_light, img_thr, 140, 255, THRESH_BINARY_INV);
}
```

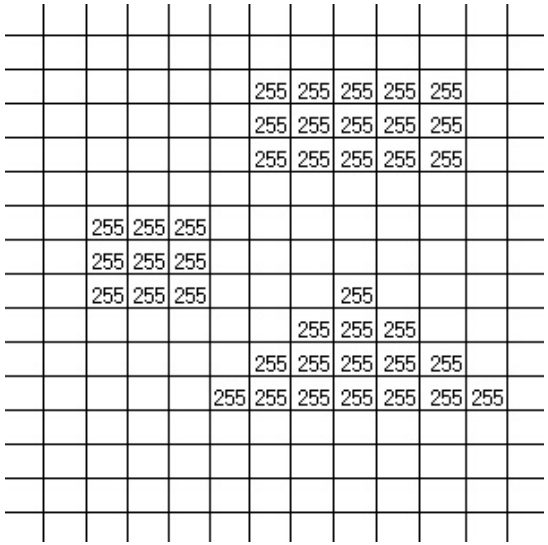
`threshold(src, dst, thresh, maxval, type)`

매개 변수	설명
src	입력 영상
dst	출력 영상
thresh	임계값
maxval	화소값이 임계값을 넘으면 부여되는 값
type	이진화 타입. 5개 중에서 하나이다. • THRESH_BINARY • THRESH_BINARY_INV • THRESH_TRUNC • THRESH_TOZERO • THRESH_TOZERO_INV

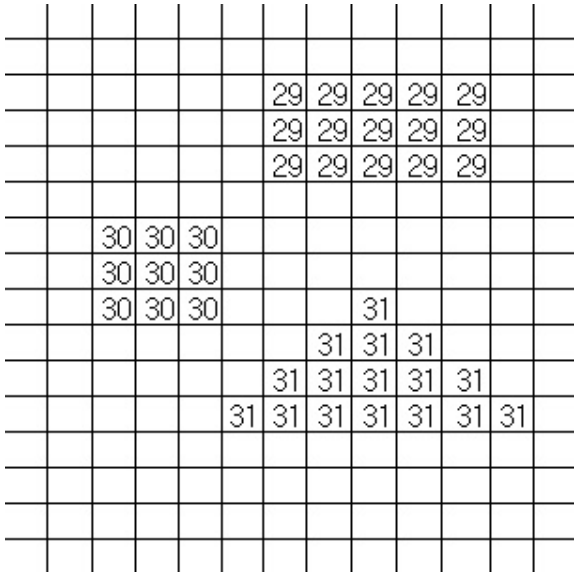
Connected Components

- Labeling
  - 인접하여 연결되어 있는 모든 화소에 동일한 번호(label) 을 붙이고, 다른 연결 성분에는 또 다른 번호를 붙이는 작업

(이진화 영상)



(레이블링된 영상)

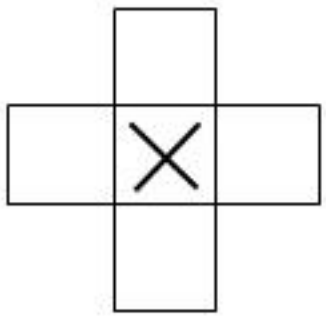


Connected Components

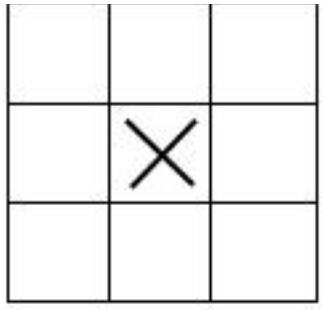
- OpenCV 함수

```
int connectedComponentsWithStats(InputArray image, OutputArray labels,
      OutputArray stats, OutputArray centroids, int connectivity=8,
      int ltype=CV_32S)
```

매개 함수	설명
image	입력 영상
labels	레이블 영상
connectivity	8-연결성이나 4-연결성
Itype	출력 영상의 레이블 타입 CV_32S 또는 CV_16U
statsv	각 레이블에 대한 통계 자료
반환값	레이블의 개수



(a) 4-neighbors



(b) 8-neighbors

Connected Components

```
void ConnectedComponentsStats(Mat img)
{
    // Use connected components with stats
    Mat labels, stats, centroids;
    auto num_objects= connectedComponentsWithStats(img, labels, stats, centroids);
    // Check the number of objects detected
    if(num_objects < 2 ){
      cout << "No objects detected" << endl;
      return;
    }else{
      cout << "Number of objects detected: " << num_objects - 1 << endl;
    }
    // Create output image coloring the objects and show area
    Mat output= Mat::zeros(img.rows, img.cols, CV_8UC3);
    RNG rng( 0xFFFFFFFF );
    for(auto i=1; i<num_objects; i++){
      cout << "Object "<< i << " with pos: " << centroids.at<Point2d>(i) << " with area "
           << stats.at<int>(i, CC_STAT_AREA) << endl;
      Mat mask= labels==i;
      output.setTo(randomColor(rng), mask);
      // draw text with area
      stringstream ss;
      ss << "area: " << stats.at<int>(i, CC_STAT_AREA);

      putText(output, ss.str(), centroids.at<Point2d>(i), FONT_HERSHEY_SIMPLEX, 0.4, Scalar(255,255,255));
    }
    imshow("Result", output);
    miw->addImage("Result", output);
}
```

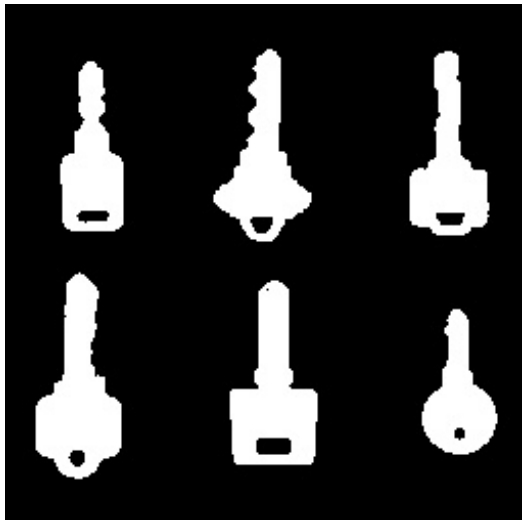
Connected Components

- Result

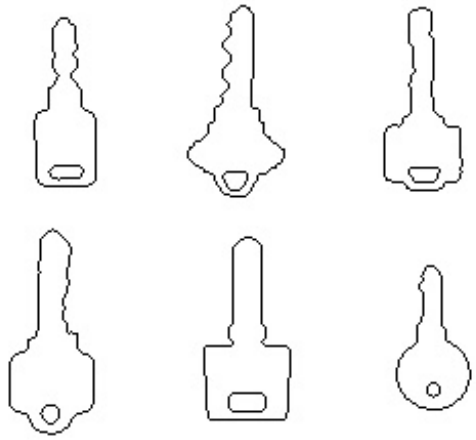
Find Contours

- Contouring or Edge Following
  - 이진화된 영상 또는 레이블링된 영상에서 영역의 경계를 추적하여, 경계 픽셀의 순서화된 정보 (chain code) 추출

(이진화 영상)

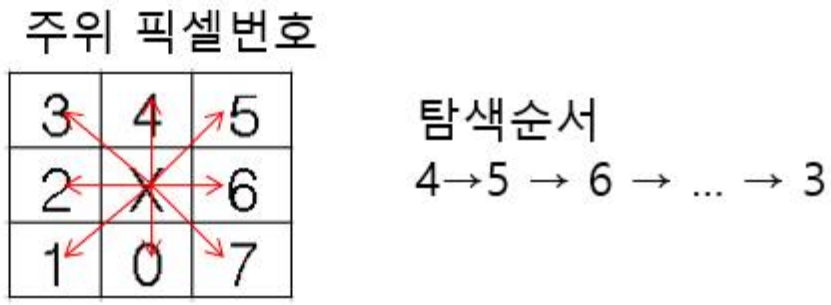


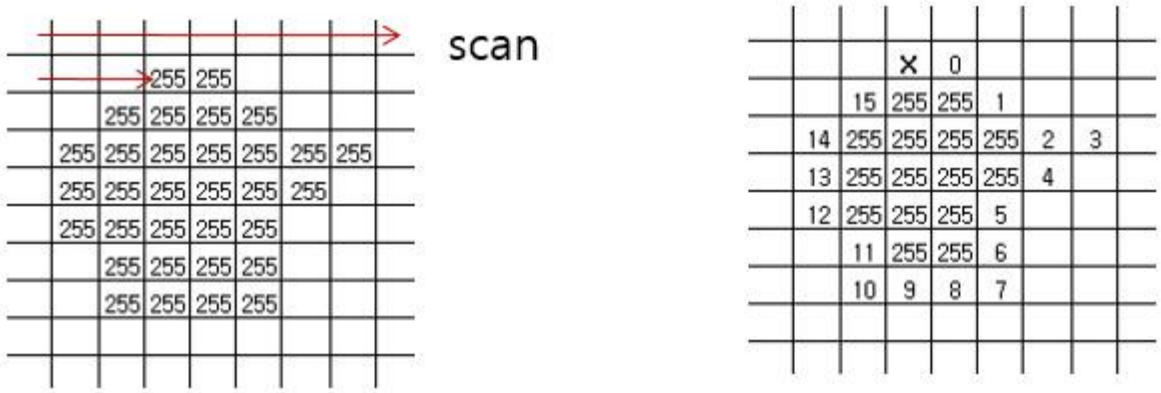
(경계 영상)



Find Contours

- Edge following algorithm





Find Contours

- OpenCV 함수

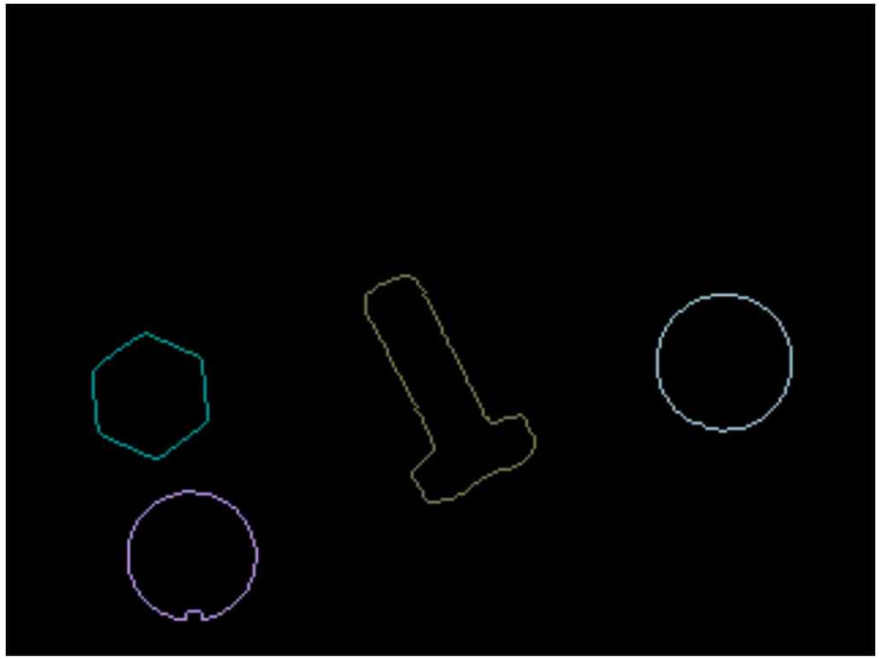
```
void cv::findContours ( InputOutputArray     image,
                        OutputArrayOfArrays  contours,
                        int                  mode,
                        int                  method,
                        Point                offset = Point()
                      )
```

Find Contours

```
void FindContoursBasic(Mat img)
{
   vector<vector<Point> > contours;
   findContours(img, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

   Mat output= Mat::zeros(img.rows,img.cols, CV_8UC3);
   // Check the number of objects detected
   if(contours.size() == 0 ){
     cout << "No objects detected" << endl;
     return;
   }else{
     cout << "Number of objects detected: " << contours.size() << endl;
   }

   RNG rng( 0xFFFFFFFF ); // random number generation
   for(auto i=0; i<contours.size(); i++)
     drawContours(output, contours, i, randomColor(rng));

   imshow("Result", output);
   miw->addImage("Result", output);
}
```

Find Contours

- Result



Test Program

```
int main( int argc, const char** argv )
{
CommandLineParser parser(argc, argv, keys);
parser.about("Chapter 5. PhotoTool v1.0.0");
//If requires help show
if (parser.has("help"))
{
parser.printMessage();
return 0;
}

String img_file= parser.get<String>(0);
String light_pattern_file= parser.get<String>(1);
auto method_light= parser.get<int>("lightMethod");
auto method_seg= parser.get<int>("segMethod");

// Check if params are correctly parsed in his variables
if (!parser.check())
{
parser.printErrors();
return 0;
}

// Load image to process
Mat img= imread(img_file, 0);
if(img.data==NULL){
cout << "Error loading image "<< img_file << endl;
return 0;
}
```

```
// Create the Multiple Image Window
miw= make_shared<MultipleImageWindow>("Main window", 3, 2,
        WINDOW_AUTOSIZE);

// Remove noise
Mat img_noise, img_box_smooth;
medianBlur(img, img_noise, 3);
blur(img, img_box_smooth, Size(3,3));

// Load image to process
Mat light_pattern= imread(light_pattern_file, 0);
if(light_pattern.data==NULL){
// Calculate light pattern
light_pattern= calculateLightPattern(img_noise);
}
medianBlur(light_pattern, light_pattern, 3);

//Apply the light pattern
Mat img_no_light;
img_noise.copyTo(img_no_light);
if(method_light!=2){
img_no_light= removeLight(img_noise, light_pattern, method_light);
}

// Binarize image for segment
Mat img_thr;
if(method_light!=2){
threshold(img_no_light, img_thr, 30, 255, THRESH_BINARY);
}else{
threshold(img_no_light, img_thr, 140, 255, THRESH_BINARY_INV);
}
```

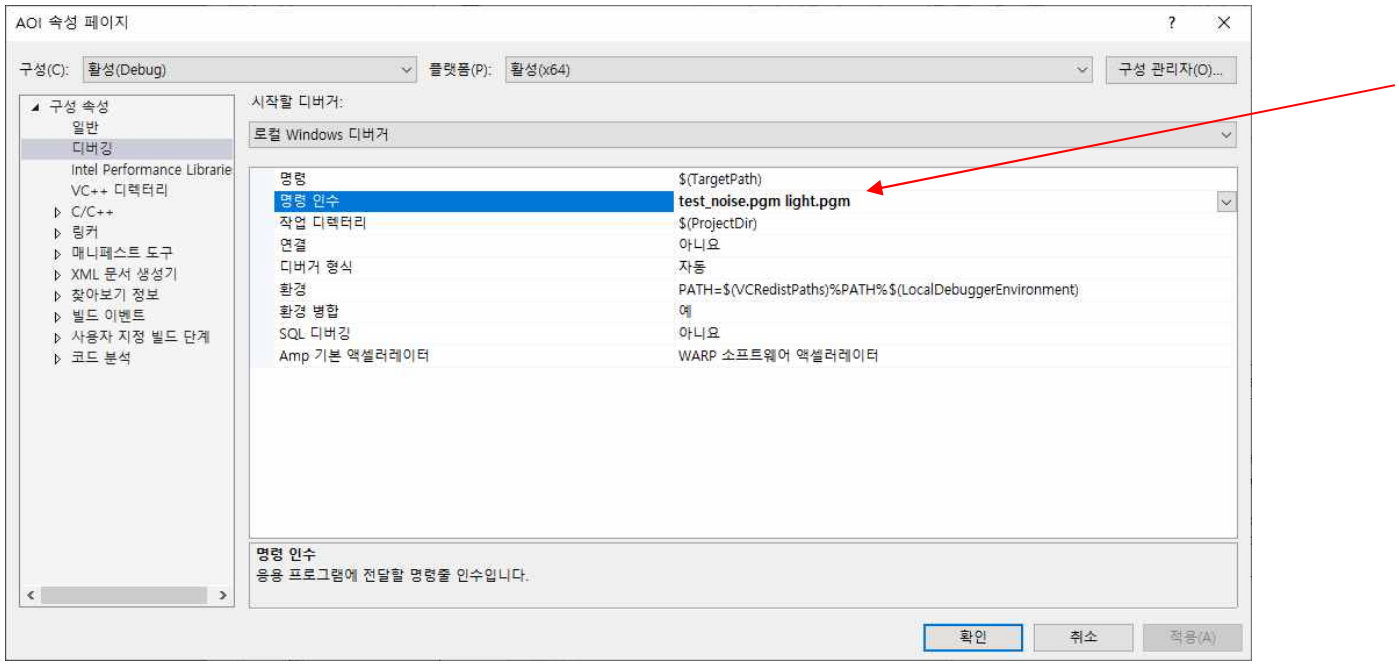
Test Program

```
// Show images
miw->addImage("Input", img);
miw->addImage("Input without noise", img_noise);
//miw->addImage("Input without noise with box smooth", img_box_smooth);
miw->addImage("Light Pattern", light_pattern);
//imshow("Light pattern", light_pattern);
//imshow("No Light", img_no_light);
miw->addImage("No Light", img_no_light);
miw->addImage("Threshold", img_thr);

switch(method_seg){
case 1:
ConnectedComponents(img_thr);
break;
case 2:
ConnectedComponentsStats(img_thr);
break;
case 3:
FindContoursBasic(img_thr);
break;
}

miw->render();
waitKey(0);
return 0;

}
```

Test Program

- Debugging

```
// OpenCV command line parser functions
// Keys accecpted by command line parser
const char* keys =
{
"{help h usage ? | | print this message}"
"{@image || Image to process}"
"{@lightPattern || Image light pattern to apply to image input}"
"{lightMethod | 1 | Method to remove backgroun light, 0 differenec, 1 div, 2 no light removal' }"
"{segMethod | 3 | Method to segment: 1 connected Components, 2 connectec components with stats, 3 find Contours }"
};
```



Test Program

- Result