영상
기말프로젝트

오예진

프로그램 목적

물체 모양(shape) 영상에서 서로 다른 모양들의 **개수**를 파악한다.

물체 모양(shape) 영상에서 서로 다른 모양들의 **위치**를 파악한다.

세부 요구사항

각 object들을 구분 가능하도록 영상을 개선한다.(노이즈 제거 및 명암 대조비 개선)

object 영역들을 구분하여 각 object들의 중심점과 boundary를 종류별로 서로 다른 색으로 표시한다.

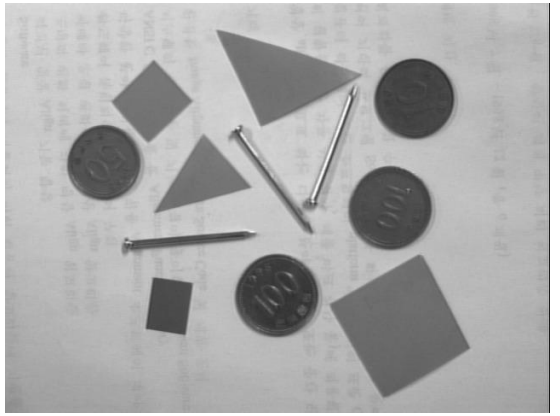
주어진 영상들을 모두 처리할 수 있어야 한다.

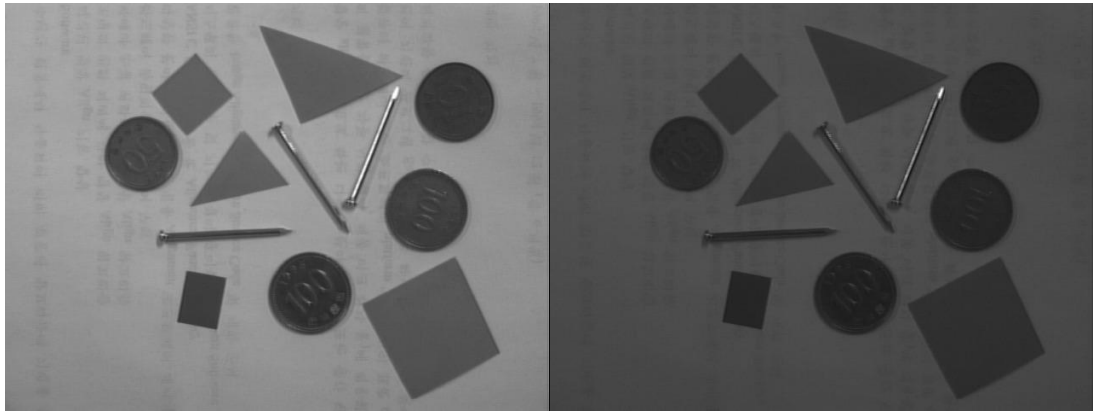
영상 이외의 다른 정보(조명 조건, 노이즈 유형, 동전 배열 상태 등)를 별도로 입력하여서는 안된다.

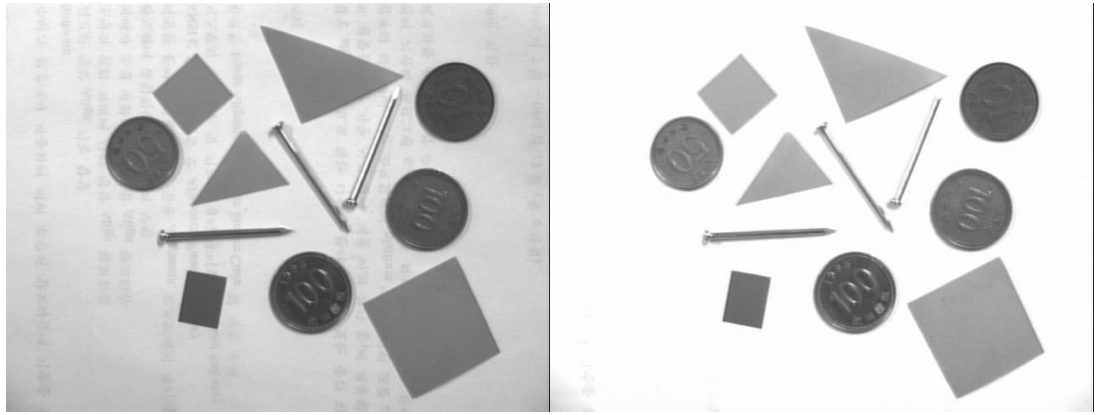
입력영상

입력영상인 shape.zip의 image분석

다섯개의 입력영상에서 솔트페퍼노이즈같은 일반적인 noise는 발견되지 않았다.
대신 영상의 명암대조비가 모두 차이가 있고 대체적으로 blur noise가 존재하므로 이를 해결하는 전처리과정을 거쳐야 한다.







소프트웨어설계
-전처리과정1

영상 전체가 blur 처리 되어있으므로 이에 대한 noise 개선

이미지의 blur noise를 개선하기 위해 샤프닝을 한다. 사용할 mask는 라플라시안 mask이며 왼쪽과 같은데, 이 mask를 사용하면 edge를 찾아낼 수 있다. 엣지만을 추출하지 않고 blur noise가 개선된 이미지 전체 결과값을 얻기 위해 배열의 중앙값에 1씩 더한다. 이렇게 만들어진 mask는 오른쪽과 같다.

위의 mask를 사용하여 blur noise가 존재하는 영상을 개선할 수 있다.

−1	−1	−1
−1	8	−1
−1	−1	−1

-1	-1	-1
-1	9	-1
-1	-1	-1

소프트웨어설계
-전처리과정2

각 입력영상의 모두 다른 명암 대조비 개선

1. histogram 함수

histogram함수를 만들어서, 입력 영상의 명암값을 계산한다.밝은 명암값을 저장하는 변수와 어두운 명암값을 저장하는 변수를 만들어서 두개를 비교한 후,

전체적으로 어두운 영상인지 밝은 영상인지 확인한다.

소프트웨어설계
-전처리과정2

각 입력영상의 모두 다른 명암 대조비 개선

2. 감마보정 함수

비선형 연산에 해당하는 감마 수정의 식은 아래와 같다.

$$f_{out}(j,i) = (L-1) \times (\hat{f}(j,i))^{\gamma} \quad \text{이때} \quad \hat{f}(j,i) = \frac{f(j,i)}{(L-1)}$$

gamma의 값이 작을수록 이미지가 밝아지고, gamma의 값이 클수록 이미지가 어두워진다.

그렇기 때문에 code에서 histogram함수의 return 값에 따라 gamma의 값은 유동적이고, 그에 따라 이미지 결과값도 다르다.

histogram 함수의 return값을 통해 영상이 전체적으로 어둡다고 판단되면 감마수정함수를 통해 영상을 밝게 하고,

영상이 전체적으로 밝다고 판단되면 감마수정함수를 통해 영상을 어둡게 한다.

소프트웨어설계
-모양파악

1. thresholding을 통한 이진화 이미지로의 변환.

입력영상에는 모양 검출에 필요하지 않은 배경등이 존재하므로, 임계값을 기준으로 단순한 이진화된 영상으로 변환하여야 한다. 아래의 간단한 식을 참고한다.

$$b(j,i) = \begin{cases} 1, & f(j,i) \geq T \\ 0, & f(j,i) < T \end{cases}$$

2. close 연산 수행

삼각형, 사각형, 원 등과 달리 못의 경우에는 close연산을 수행하여야 한다.

$$S_t = \{\mathbf{s} + \mathbf{t} \mid \mathbf{s} \in S\}$$

팽창 : $f \oplus S = \bigcup_{\mathbf{x} \in f} S_{\mathbf{x}}$

침식 : $f \ominus S = \{\mathbf{x} \mid \mathbf{x} + \mathbf{s} \in f, \forall \mathbf{s} \in S\}$

열기 : $f \circ S = (f \ominus S) \oplus S$

닫기 : $f \cdot S = (f \oplus S) \ominus S$

소프트웨어설계
-모양파악

3. contour list인 contours 찾은 후 근사화 및 윤곽선 표시

입력 영상을 grayscale로 읽은 후 연산을 모두 거쳐 모양을 찾아 낸다. 이렇게 찾아낸 모양에 맞는 윤곽선을 원본 영상에 표시하여 result_img로 띄운다.

소프트웨어설계
-모양검출

1. 모양의 개수 파악

우선 못의 경우에는 다른 모양들과 달리 명확한 특징이 없기 때문에 꼭짓점이 3개, 4개 등으로 나타난다.따라서 이의 경우에는 꼭짓점의 개수가 아닌 근사화한 contour의 면적을 통해 판단한다.

삼각형의 경우에는 꼭짓점이 3개, 사각형의 경우에는 꼭짓점이 4개, 원의 경우에는 꼭짓점 5개 이상으로 판단한다.판별된 각 모양의 윤곽선은 다른 색으로 표현하여 차이를 두고, 각 모양이 몇개 나왔는지 알 수 있도록 변수를 선언한다.

소프트웨어설계
-모양검출

2. 모양의 위치 파악

구한 contour와 외접하는 직사각형을 구하는 opencv의 내장함수를 이용한다.이 사각형의 중심좌표를 구해 result_img에 표시하여 모양의 위치를 시각적으로 나타내고,

변수의 값을 출력하여 정확한 위치 또한 알 수 있게 한다.

시스템구현
-blur noise 개선

source code는 다음과 같다. 이미지 테두리에서는 계산을 할 수 없기 때문에, 마스크 크기/2만큼 테두리 영역을 무시하고 계산하고, mask_size/2가 이를 의미한다.

```
void blur_noise(Mat &original_img) {

    Mat blured_img = original_img.clone();

    int mask[3][3] = { {-1,-1,-1},{-1,9,-1},{-1,-1,-1} };    //라플라시안 +1

    long int sum;
    int mask_size = 3;          //mask[3][3]

    for (int row = 0 + mask_size / 2; row < original_img.rows - mask_size / 2; row++){
        for (int col = 0 + mask_size / 2; col < original_img.cols - mask_size / 2; col++){
            sum = 0;
            for (int i = -1 * mask_size / 2; i <= mask_size / 2; i++){
                for (int j = -1 * mask_size / 2; j <= mask_size / 2; j++){
                    sum += original_img.at<uchar>(row + i, col + j) * mask[mask_size / 2 + i][mask_size / 2 + j];
                }
            }

            if (sum > 255)          //명암은 255를 넘지 않게 한다.
                sum = 255;
            if (sum < 0)
                sum = 0;

            blured_img.at<uchar>(row, col) = sum;
        }
    }
```

시스템구현
-영상 밝기 조절

1. histogram 함수

histogram함수를 통해 입력 영상의 명암값을 계산한다. 변수 count_histo_l는 영상의 밝은 부분, count_histo_d는 영상의 어두운 부분을 나타내는 변수이다.

계산을 통해 count_histo_l의 값이 count_histo_d보다 크면 영상이 전체적으로 밝다는 의미이기 때문에 return num1을하고, 그 반대의 경우라면 return num2를 한다.

```
int histogram(Mat &original_img) {

    float histogram[256] = { 0 };        //히스토그램
    double count_histo_l = 0, count_histo_d = 0;      // count_histo_l:밝음,  count_histo_d:어두움
    int num1 = 1, num2 = 2;

    for (int row = 0; row < original_img.rows; row++) {
        for (int col = 0; col < original_img.cols; col++) {
            histogram[original_img.at<uchar>(row, col)]++;
        }
    }

    for (int i = 0; i < 256; i++) {
        if (i < 128)
            count_histo_d += histogram[i];
        else
            count_histo_l += histogram[i];
    }

    if (count_histo_l < count_histo_d)
        return num1;       //어두운 영상
    else
        return num2;       //밝은 영상

}
```

시스템구현
-영상 밝기 조절

2. 감마수정 함수

우선 감마수정의 코드는 다음과 같다. 이는 앞선 과제에서 구현한 gamma_transformation과 같다.

```
Mat gamma_transformation(Mat &original_img, float gamma) {
    Mat gamma_img = original_img.clone();

    float c = 255;

    for (int row = 0; row < original_img.rows; row++) {
        for (int col = 0; col < original_img.cols; col++) {
            gamma_img.at<uchar>(row, col) = c * pow(original_img.at<uchar>(row, col) / c, gamma);
        }
    }
    return gamma_img;
}
```

시스템구현
-영상 밝기 조절

2. 감마수정 함수

영상의 대조비 개선은 아래의 코드를 통해 계산한다.return 값이 1, 즉 입력 영상이 어두운 영상이라면 gamma_transformation의 gamma변수를 0.5로 하여 영상을 밝게 하고, return 값이 2, 즉 반대의 경우라면 gamma변수를 1.7로 하여 영상을 어둡게한다.

이를 통해 입력 영상의 전체적인 명암 대조비 개선을 할 수 있다.

```
int histo;
histo = histogram(original_img);
if (histo == 1)          //어두운 영상이면
    original_img = gamma_transformation(original_img, 0.5);
else                     //밝은 영상이면
    original_img = gamma_transformation(original_img, 1.7);
imshow("gamma_img", original_img);
```

시스템구현
-모양 파악

1. 이진화 이미지로 변환

입력 영상을 이진화 영상으로 변환할 때에는 thresholding을 사용한다.

시스템구현
-모양 파악

2. close 연산 수행
못 검출을 위해 close연산을 수행해야 한다. 사용한 함수는 다음과 같다.

morphologyEx()
erosion와 dilation을 이용해서 이미지를 향상시키는 함수이다.
CV_MOP_CLOSE가 closing연산을 의미한다.

getstructuringElement()
우리가 만들고자 하는 마스크를 만들어준다
MORPH_ELLIPSE는 마스크를 장축과 단축이 존재하는 타원 모양으로 만들어주는 것을 의미한다.

시스템구현
-모양 파악

3. contours 찾은 후 근사화

우선 contour란 동일한 색 또는 동일한 색상 강도(intensity)를 가진 부분의 가장 자리 경계를 연결한 선이다. opencv에서 contour 찾기는 검정색 배경에서 흰색 물체를 찾는 것과 비슷하므로, contour를 찾기 위해 대상을 흰색으로, 배경을 검정색으로 변경한다. 즉, 입력영상에서 보다 정확한 이미지 contour를 확보하기 위해threshold된 binary image를 사용하였다.

findContours(a,b,c)

a : contour찾기를 할 소스 이미지를 의미한다.

b : contour 추출 모드를 의미한다. RETR_EXTERNAL는 이미지의 가장 바깥쪽의 contour만 추출함을 의미한다.

c : contour 근사 방법을 의미한다. CHAIN_APPROX_SIMPLE는 contour의 수평,수직,대각선 방향의 점은 모두 버리고 끝 점만 남겨두는 것을 의미한다.

즉, contours는 findContours()를 통해 찾은 contour의 list형 자료이고, [i]번째 contour는 contour[i]로 찾을 수 있다.

approxPolyDP(a,b,c) (선분 간략화)

꼭지점의 수를 줄이면서 외곽선을 그려 근사화하는 함수이다.

a는 구한 contour를 의미한다.

b는 근사의 정확도를 위한 값으로 이 epsilon값이 커질수록 근사화됨을 의미한다

c가 true이면 폐곡선이다.

시스템구현
-모양 파악

4. 기타 함수

contourArea() 면적 계산을 위한 함수이다.

arcLength(a,b)

contour 호의 길이를 구한다.

이때 b는 contour가 폐곡선인지 아닌지를 나타내는 boolean값으로 true이면 폐곡선을 의미한다.

line(a, b, c, d)

a 는 line을 그릴 image, b, c 는 각각 시작점, 끝점, d 는 line 의 색을 의미한다.

여기서는 Scalar(x,y,z)를 통해 원하는 값으로 설정하였다.

시스템구현
-모양 파악

```
//이진화 이미지로 변환
//thresholding
threshold(original_img, original_img, 100, 250, THRESH_BINARY_INV | THRESH_OTSU);     //오츄 알고리즘 사용

//close연산 수행(못 검출을 위함)
Mat to_close = getStructuringElement(MORPH_ELLIPSE, Size(3, 3));       //mask 사이즈는 (3,3)으로 지정
morphologyEx(original_img, original_img, CV_MOP_CLOSE, to_close);

//findContours함수를 사용하여 contour list인 contours 찾음
vector<vector<Point>> contours;
findContours(original_img, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

```
//찾아낸 contour를 근사화함
vector<Point> img_approx;
for (int i = 0; i < contours.size(); i++)
{
    //epsilon : 호의길이*0.02로 설정함
    approxPolyDP(Mat(contours[i]), img_approx, arcLength(Mat(contours[i]), true)*0.02, true);
    if (contourArea(Mat(img_approx)) > 300)  //면적이 일정크기 이상이어야 한다.
    {

        //Contour를 근사화한 직선을 그린다.
        for (int j = 0; j < img_approx.size(); j++) {
            if ((contourArea(Mat(img_approx))) < 1500) {       //면적이 1500보다 작으면 못으로 간주함
                line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(255, 255, 0), 3);
            }

            else if (img_approx.size() == 3) {                 //삼각형
                line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(0, 0, 255), 3);
            }

            else if (img_approx.size() == 4) {                 //사각형
                line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(255, 0, 255), 3);
            }

            else {                                             //꼭짓점이 4개보다 많으면 원으로 간주함
                line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(102, 255, 0), 3);
            }

        }
```

시스템구현
-모양 검출

1. 모양 개수 검출

우선 면적을 통해 못인지 판별한 후, 꼭짓점의 개수로 그 외의 모양을 판별한다.변수를 설정하여 정확히 어떤 모양이 몇개 나왔는지 판별할 수 있도록 하였다.

```
        //도형 판별
        //꼭짓점이 3개나 그 이상으로 나와도 면적이 1500보다 작으면 못으로 간주함
        if ((contourArea(Mat(img_approx))) < 1500)
            num_mot++;

        else if (img_approx.size() == 3)              //삼각형
            num_tri++;

        else if (img_approx.size() == 4)              //사각형
            num_rect++;

        else                                        //꼭짓점이 4개보다 많으면 원으로 간주함
            num_round++;


    }
}
cout << "num_of_mot : " << num_mot << endl;         //못의 개수
cout << "num_of_triangle : " << num_tri << endl;    //삼각형 개수
cout << "num_of_rectangle : " << num_rect << endl;  //사각형 개수
cout << "num_of_round : " << num_round << endl;     //원 개수
```

시스템구현
-모양 검출

2. 모양 위치 검출

모든 모양이 성공적으로 검출되었기 때문에, 각 모양의 중심위치를 계산할 수 있다.계산한 중심위치를 result_img에 시각적으로 표시하고, 정확한 중심 위치를 출력할 수 있다.

boundingRect()

opencv의 내장함수로, 구한 contour의 외접직사각형을 구하는 함수이다.
이 직사각형의 중심좌표를 구해 면적의 중심 좌표를 계산한다.

putText()

원하는 이미지에 원하는 string을 출력할 수 있는 함수이다.
검출하고자 하는 이미지 중 못의 경우는 면적이 좁아서, 윤곽선과 같은 색으로 putText()할 경우 잘 보이지 않아 색을 다르게 하였다. 다른 모양은 윤곽선과 같은 색으로 중심위치를 나타내었다.
중심 좌표를 표시하도록 수정한 코드는 아래와 같다.

시스템구현
-모양 검출

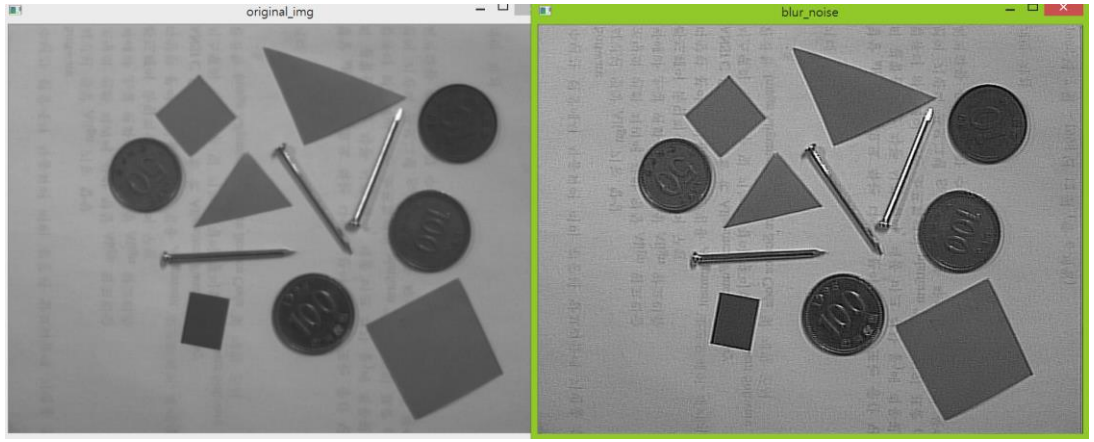
```
Rect out_rect = boundingRect(contours[i]);      //주어진 contour와 외접하는 직사각형
Point point_center(out_rect.x + (out_rect.width / 2), out_rect.y + (out_rect.height / 2));      //외접하는 직사각형의 중심점 point_center를 구함

//Contour를 근사화한 직선을 그린다.
for (int j = 0; j < img_approx.size(); j++) {
    if ((contourArea(Mat(img_approx))) < 1500) {       //면적이 1500보다 작으면 못으로 간주함
        line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(255, 255, 0), 3);
        putText(result_img, "c", point_center, FONT_HERSHEY_PLAIN, 0.5, Scalar(0, 0, 0), 3, 7);  //result_img에 중심좌표를 나타낸다
        //c = "center"
    }

    else if (img_approx.size() == 3) {                  //삼각형
        line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(0, 0, 255), 3);
        putText(result_img, "c", point_center, FONT_HERSHEY_PLAIN, 0.5, Scalar(0, 0, 255), 3, 7);   //result_img에 중심좌표를 나타낸다
    }

    else if (img_approx.size() == 4) {                  //사각형
        line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(255, 0, 255), 3);
        putText(result_img, "c", point_center, FONT_HERSHEY_PLAIN, 0.5, Scalar(255, 0, 255), 3, 7);    //result_img에 중심좌표를 나타낸다
    }

    else {                                              //꼭짓점이 4개보다 많으면 원으로 간주함
        line(result_img, img_approx[j], img_approx[(j + 1) % img_approx.size()], Scalar(102, 255, 0), 3);
        putText(result_img, "c", point_center, FONT_HERSHEY_PLAIN, 0.5, Scalar(102, 255, 0), 3, 7);    //result_img에 중심좌표를 나타낸다
    }

}
```

```
//각 모양의 중심점 출력
if ((contourArea(Mat(img_approx))) < 1500)
    cout << "mot_locate : " << point_center << endl;
else if (img_approx.size() == 3)
    cout << "triangle_locate : " << point_center << endl;
else if (img_approx.size() == 4)
    cout << "rectangle_locate : " << point_center << endl;
else
    cout << "round_locate : " << point_center << endl;
```

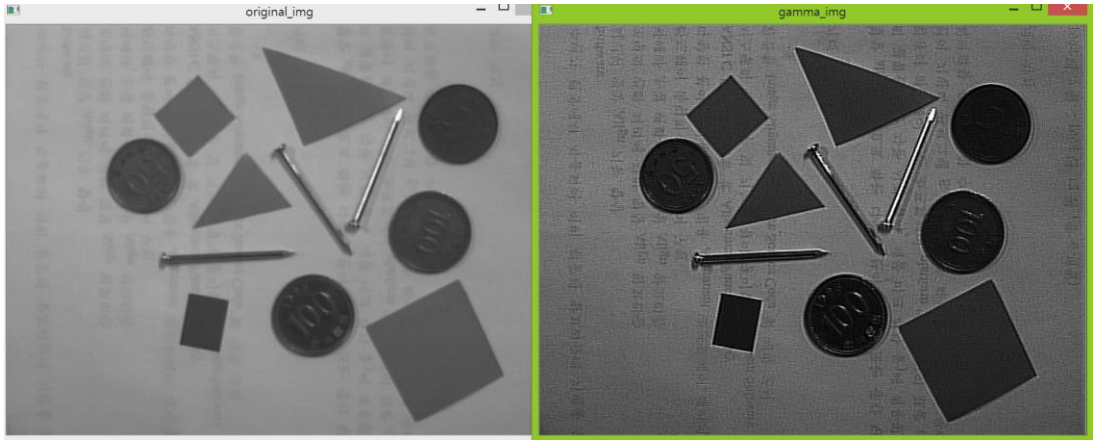
시스템 테스트 및 평가
-전처리 과정
shape1

blur noise가 개선되고, 영상의 명암 대조비가 개선된 모습을 확인 할 수 있다.

Blur noise 개선



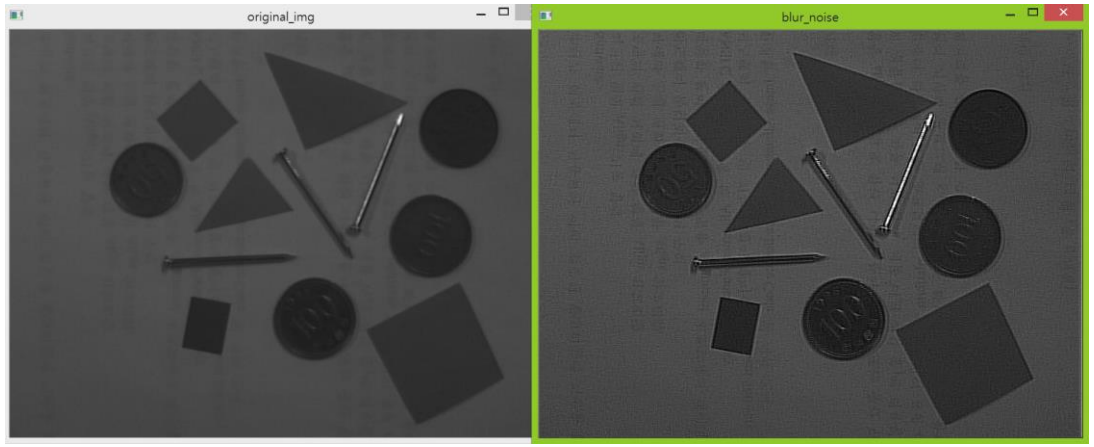
명암 대조비 개선



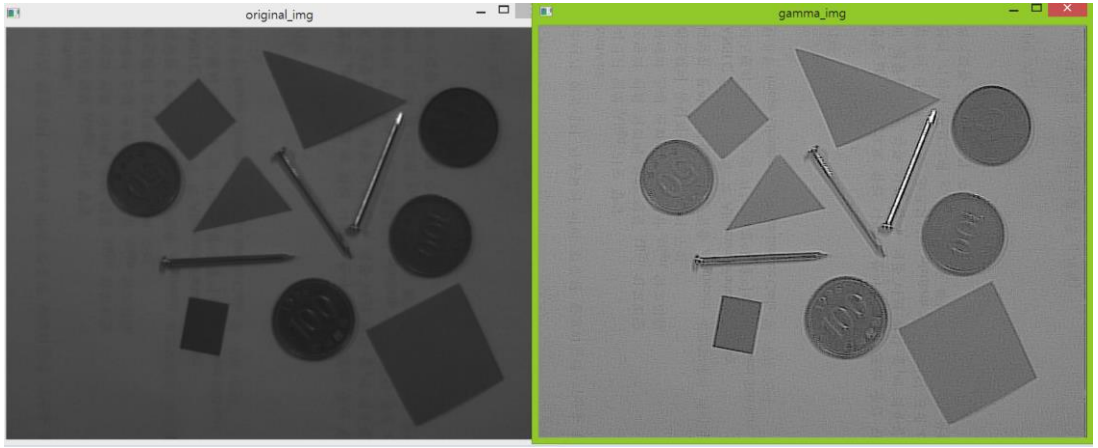
시스템 테스트 및 평가
-전처리 과정
shape2

blur noise가 개선되고, 영상의 명암 대조비가 개선된 모습을 확인 할 수 있다.

Blur noise 개선



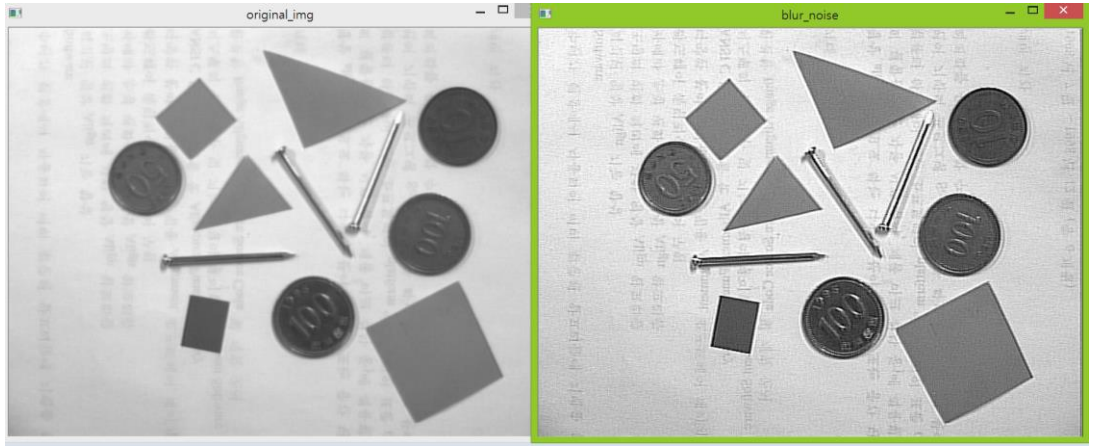
명암 대조비 개선



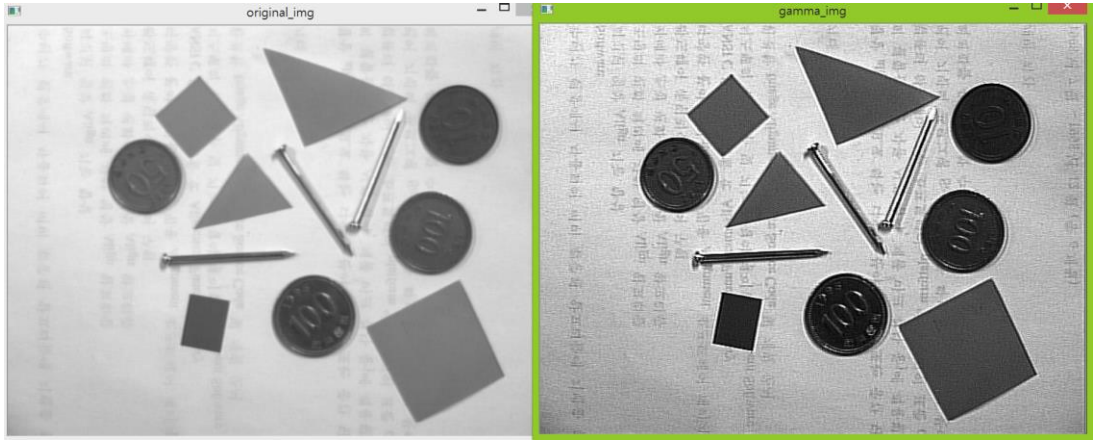
시스템 테스트 및 평가
-전처리 과정
shape3

blur noise가 개선되고, 영상의 명암 대조비가 개선된 모습을 확인 할 수 있다.

Blur noise 개선



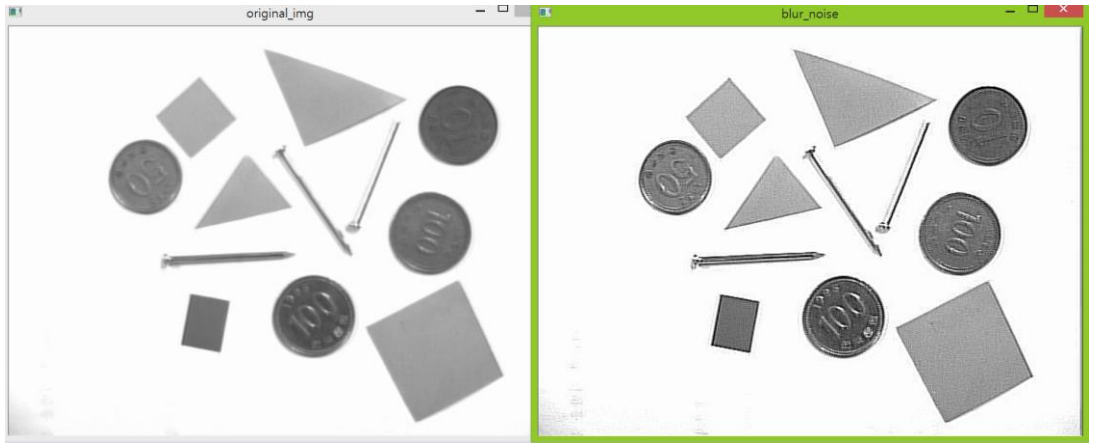
명암 대조비 개선



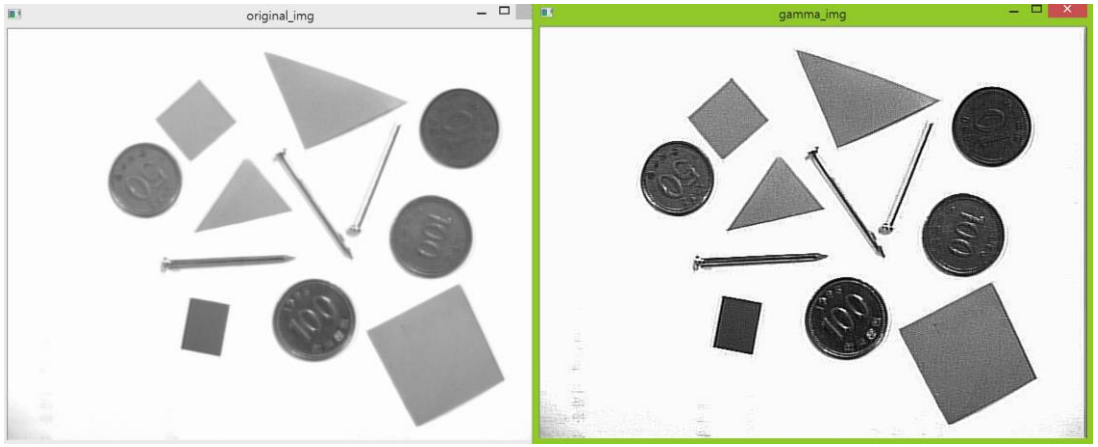
시스템 테스트 및 평가
-전처리 과정
shape4

blur noise가 개선되고, 영상의 명암 대조비가 개선된 모습을 확인 할 수 있다.

Blur noise 개선



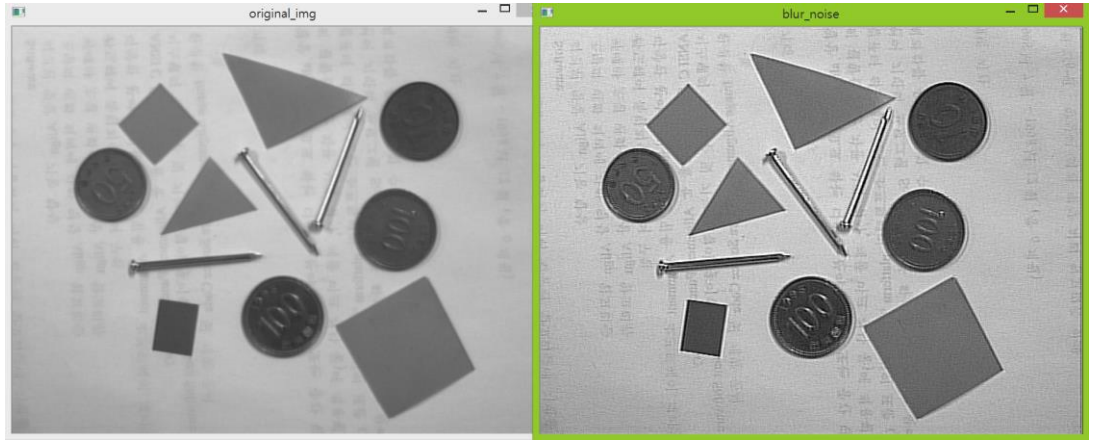
명암 대조비 개선



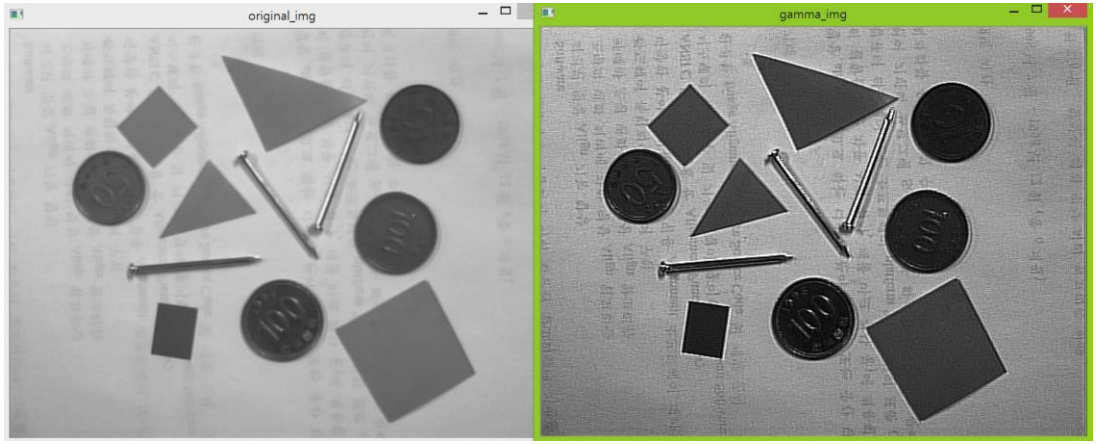
시스템 테스트 및 평가
-전처리 과정
shape5

blur noise가 개선되고, 영상의 명암 대조비가 개선된 모습을 확인 할 수 있다.

Blur noise 개선



명암 대조비 개선

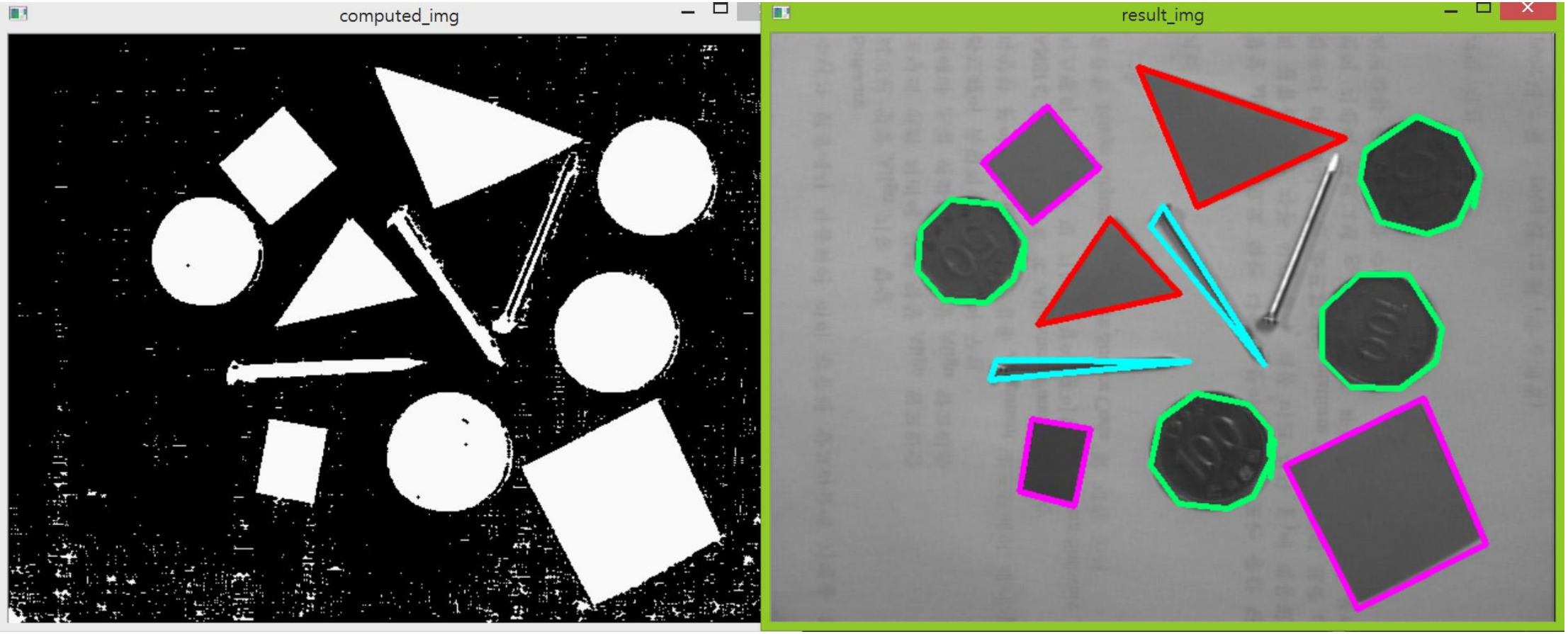


시스템 테스트 및 평가
-모양 파악 테스트 1

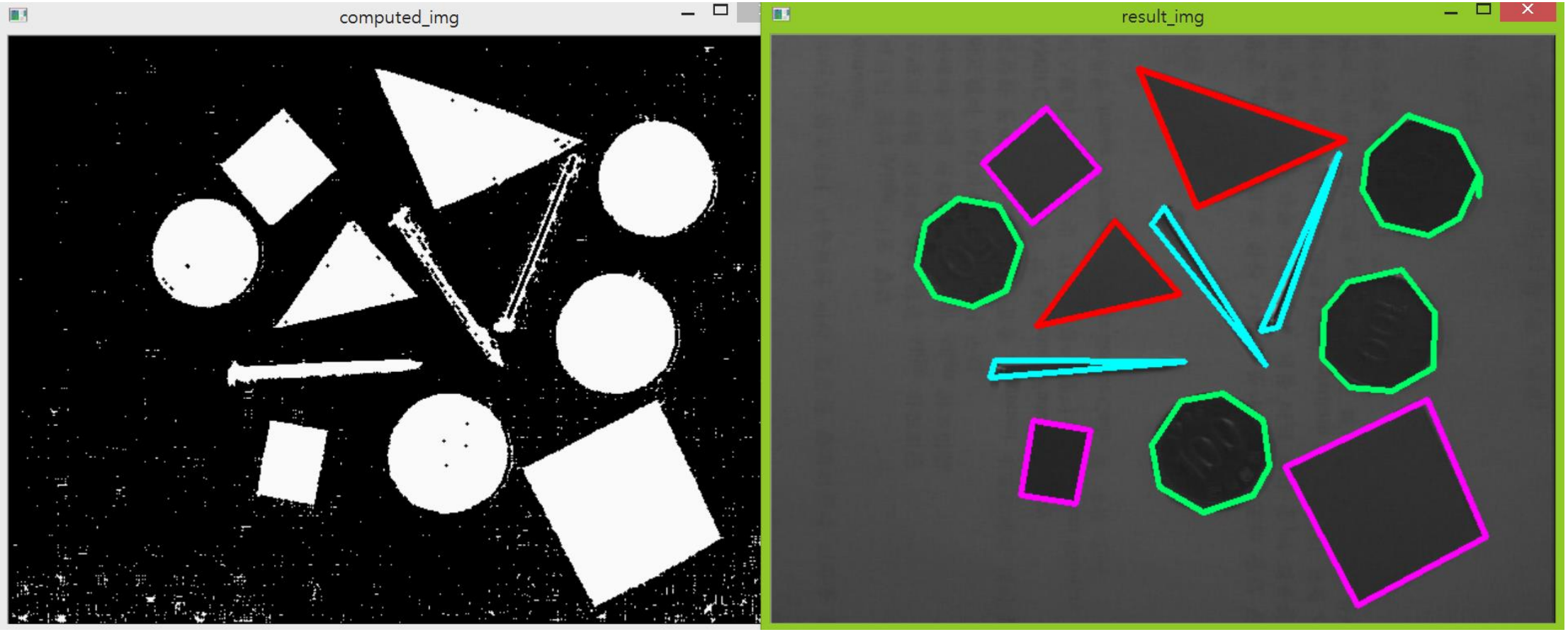
모든 입력영상에 약간의 blur noise가 있어서 이를 개선하였다.

이를 다시 이진화 영상으로 변환하고 모양을 검출하니, 오히려 blur noise개선으로 인해 제대로 검출되지 않았다.

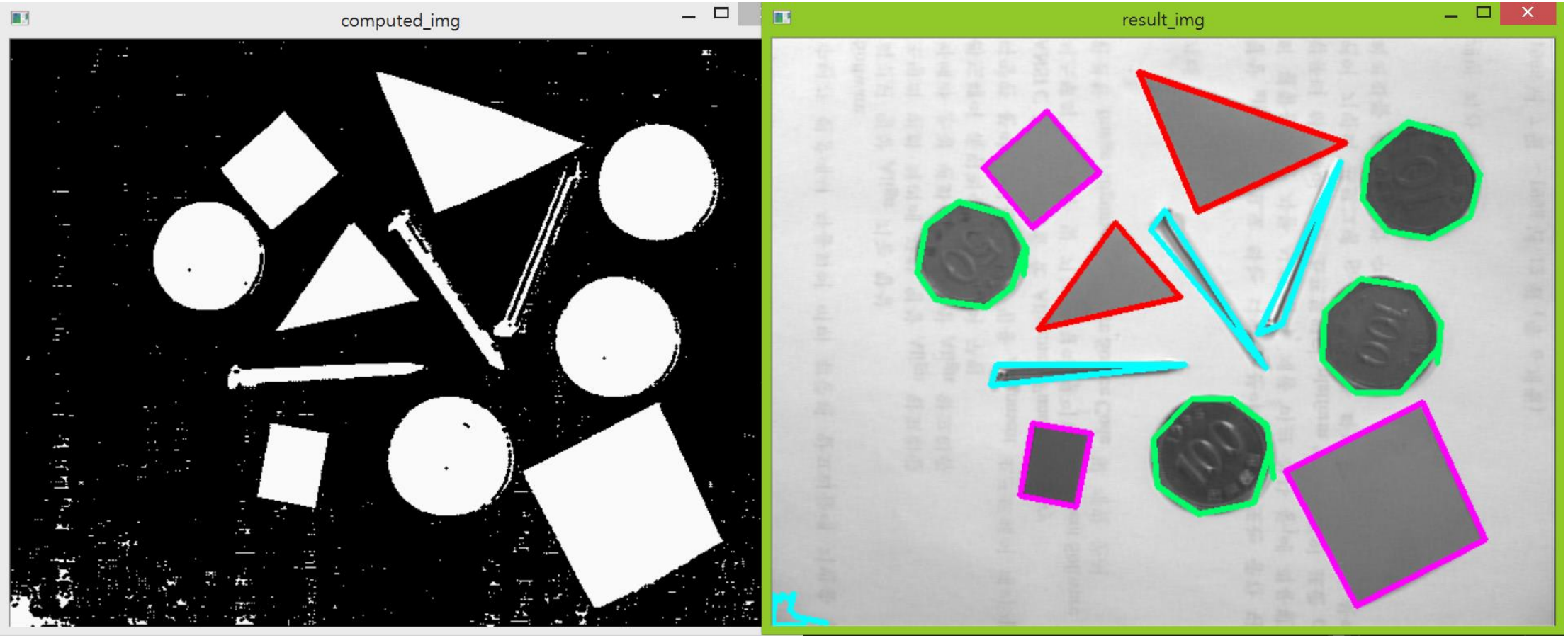
시스템 테스트 및 평가
-모양 파악 테스트 1
shape1



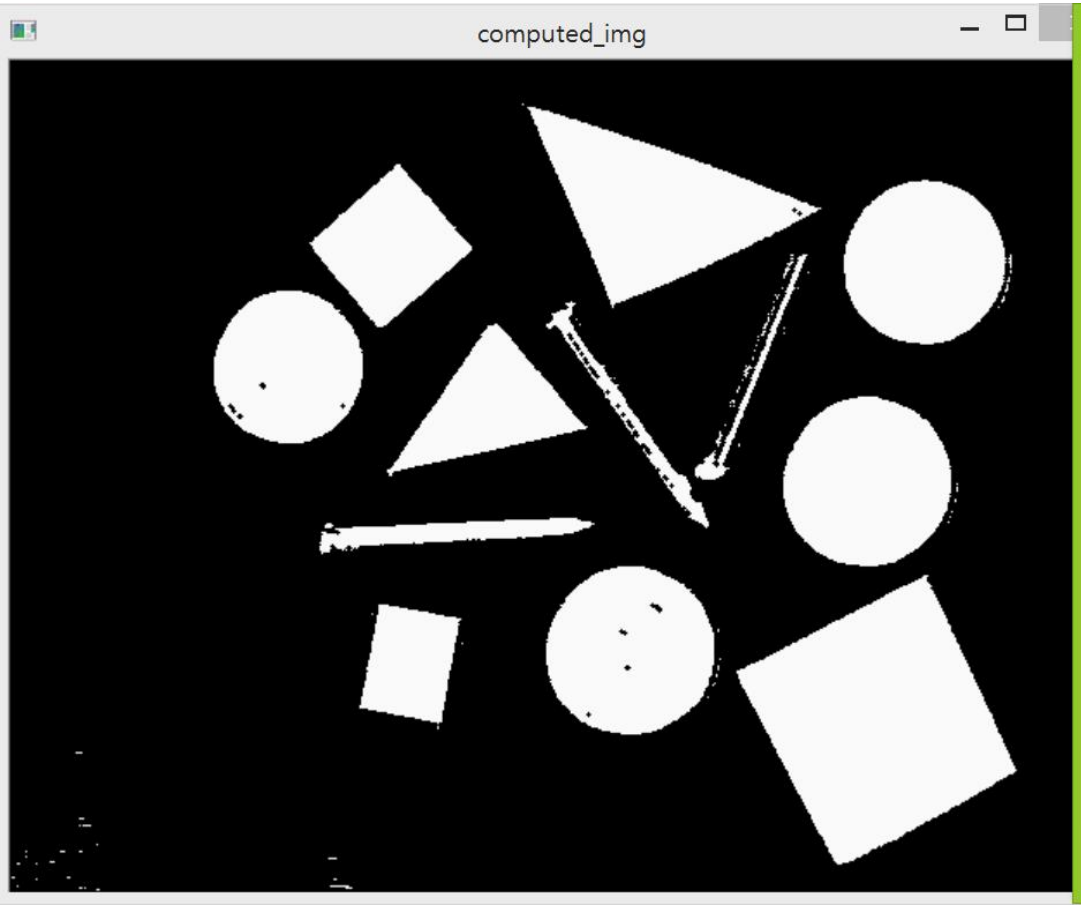
시스템 테스트 및 평가
-모양 파악 테스트 1
shape2

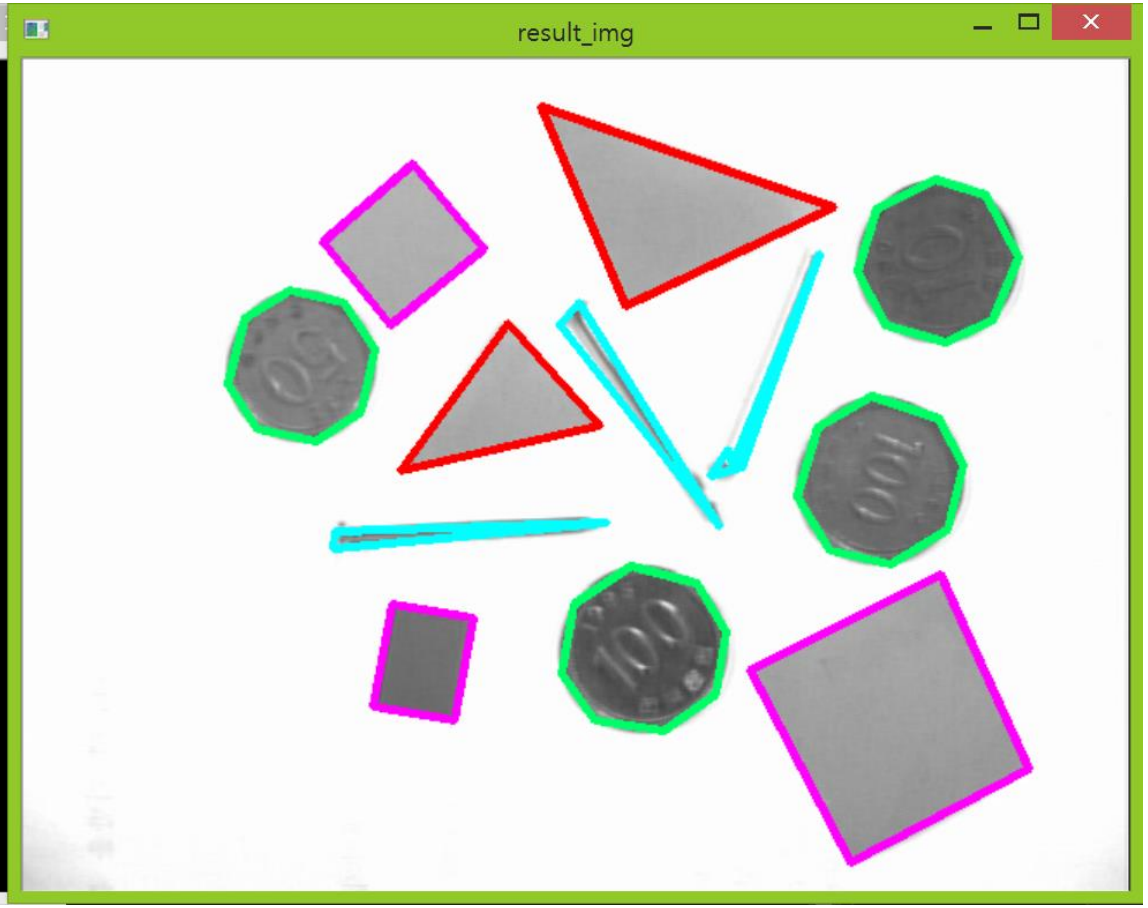


시스템 테스트 및 평가
-모양 파악 테스트 1
shape3

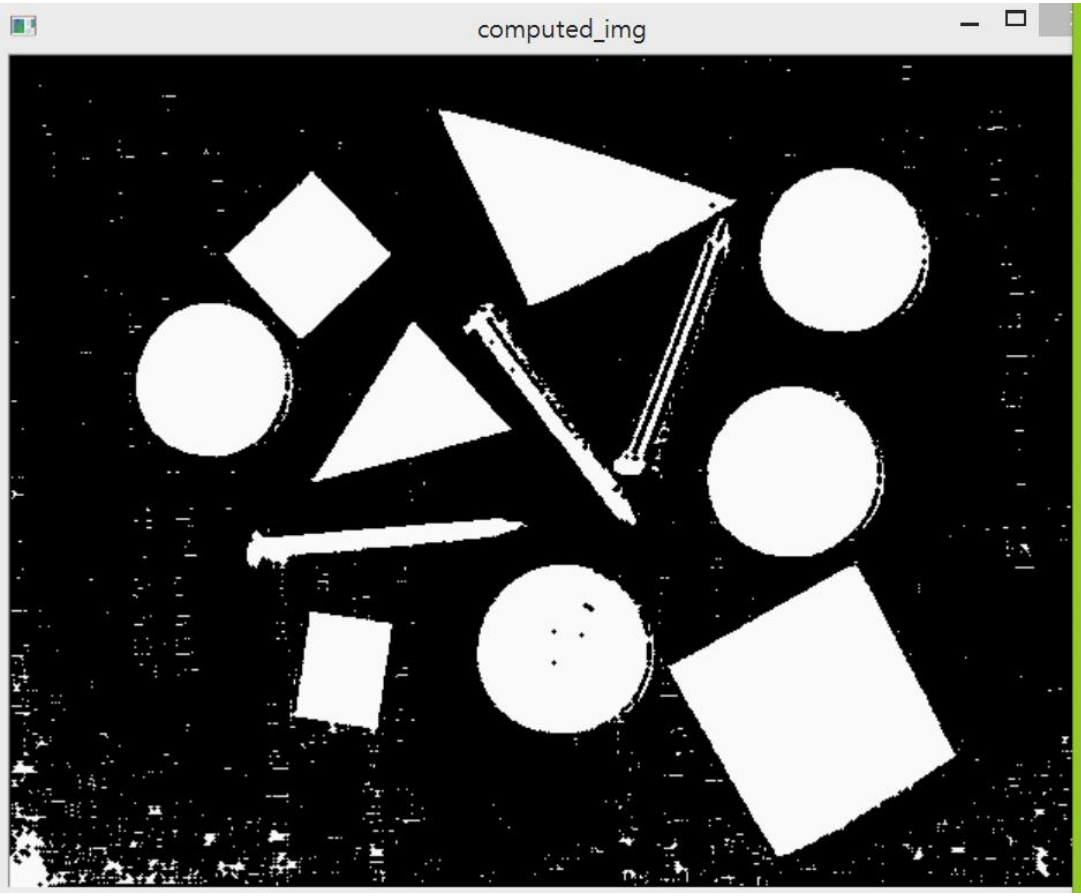


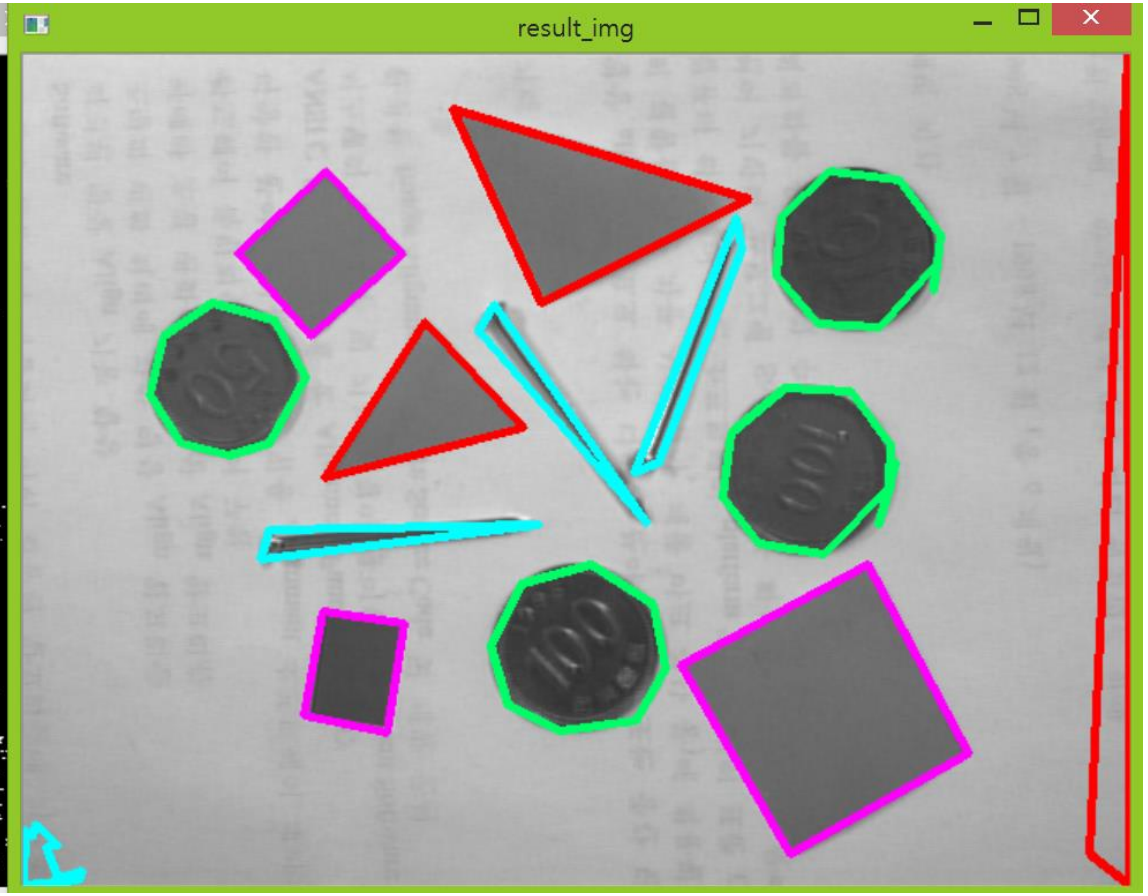
시스템 테스트 및 평가
-모양 파악 테스트 1
shape4





시스템 테스트 및 평가
-모양 파악 테스트 1
shape5





시스템 테스트 및 평가
-모양 파악 테스트 1

제대로 검출되는 영상도 있긴하지만, 일부 영상의 경우 잘못 인식하거나 아예 인식하지 못하는 부분도 있었다.

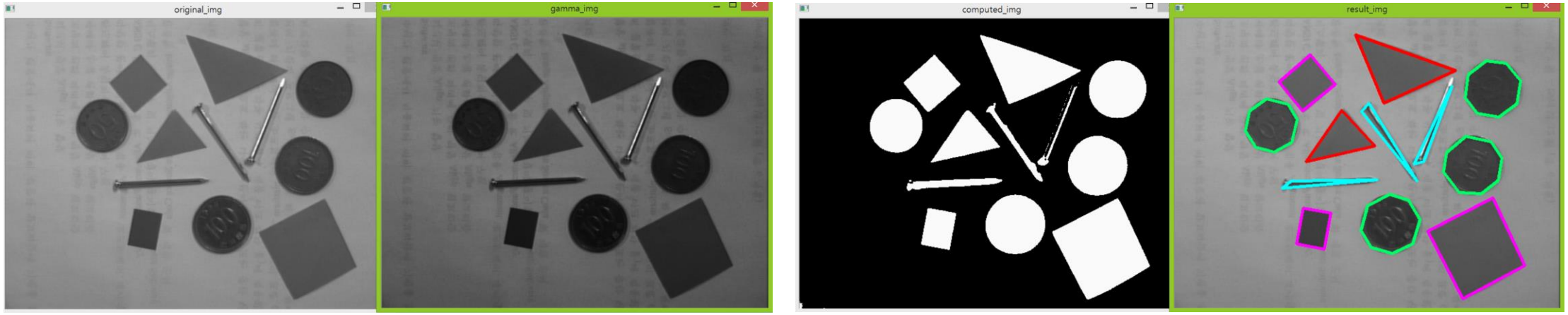
이를 해결하기 위해 blur noise를 제거하지 않고 다시 프로그램을 수행하였더니, 5개의 입력영상 모두 제대로 검출되었다.

시스템 테스트 및 평가
-모양 파악 테스트 2

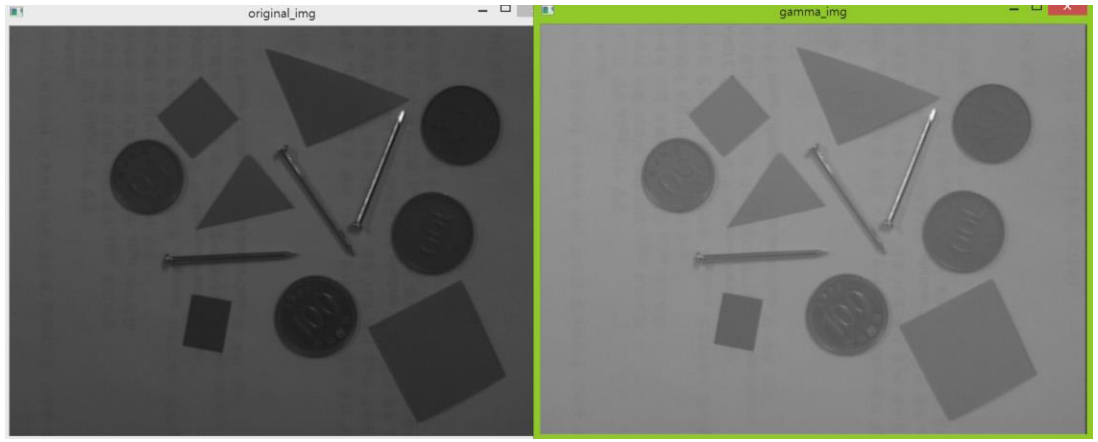
명암대조비만을 개선한 후 각 모양을 파악하고 윤곽선을 그림

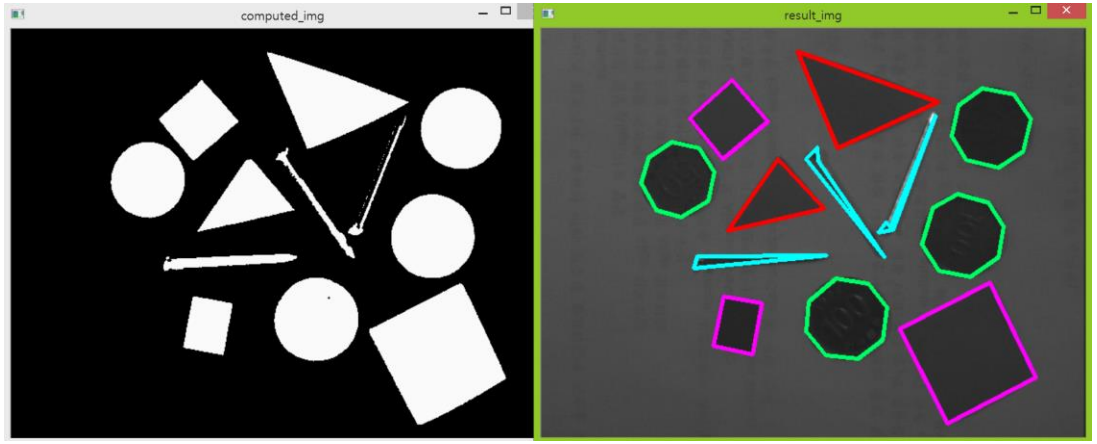
모든 영상의 blur noise를 개선하지 않고, gamma 수정만을 거친 후 모양 검출을 하면 5개의 입력 영상 모두 제대로 모양이 검출된다.

시스템 테스트 및 평가
-모양 파악 테스트 2
shape1

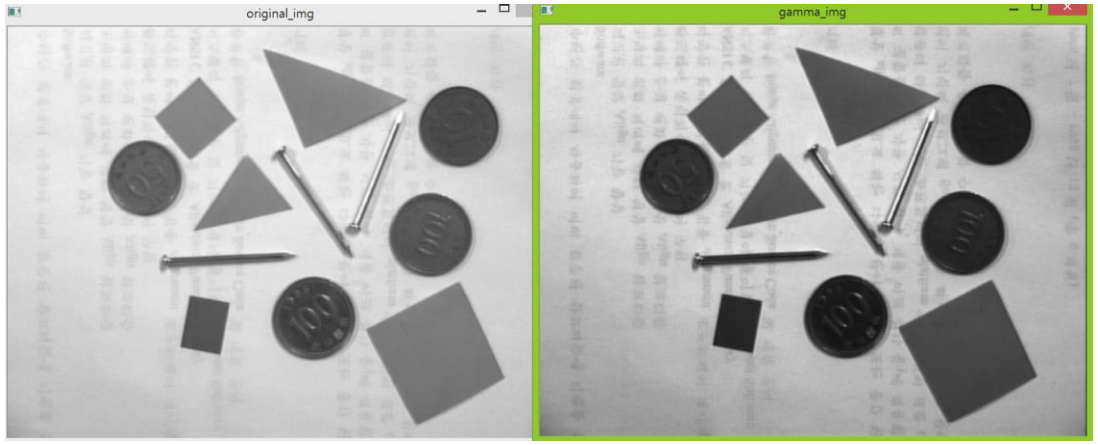


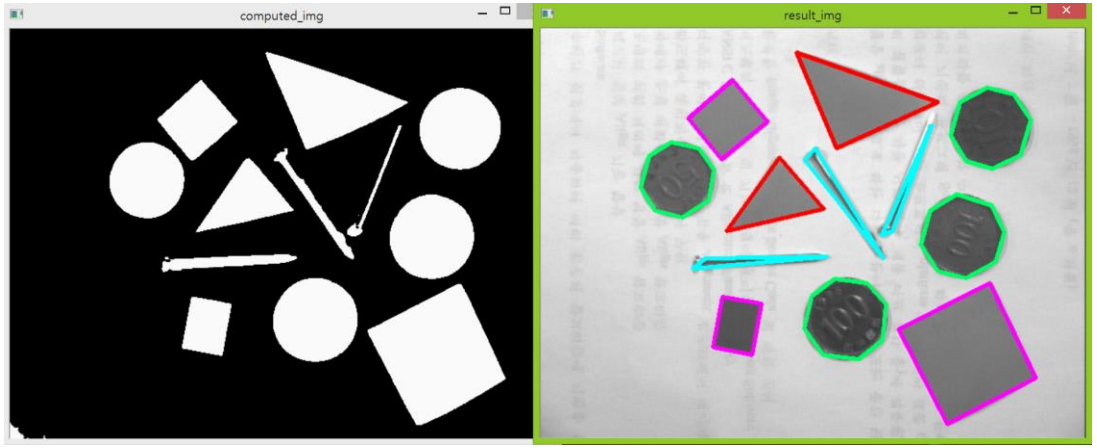
시스템 테스트 및 평가
-모양 파악 테스트 2
shape2



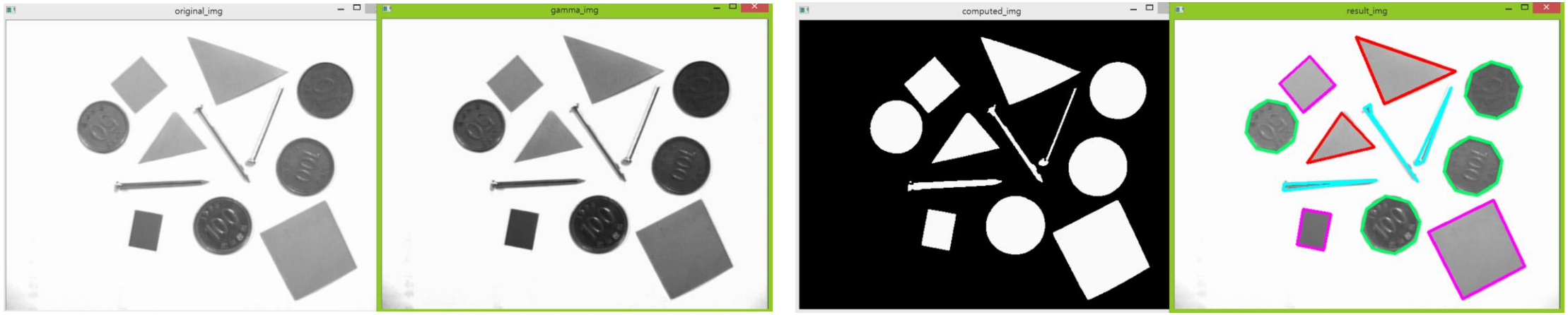


시스템 테스트 및 평가
-모양 파악 테스트 2
shape3

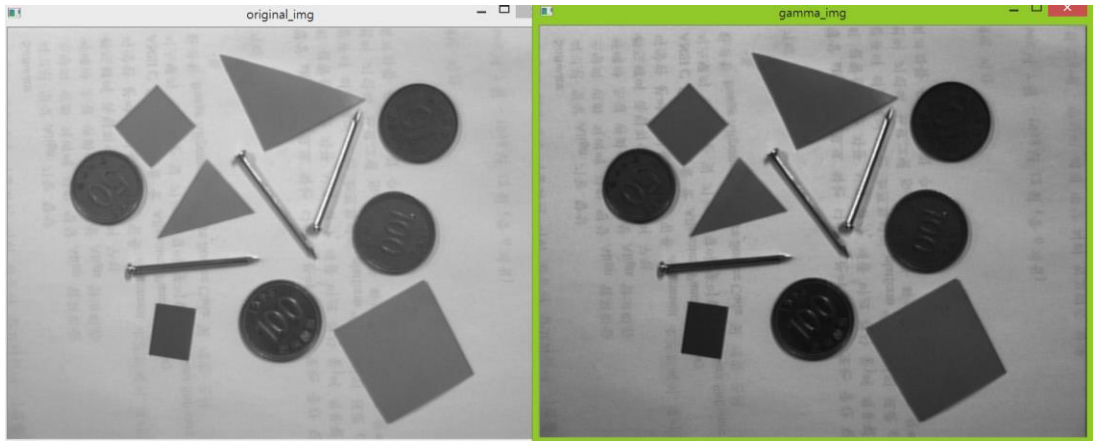


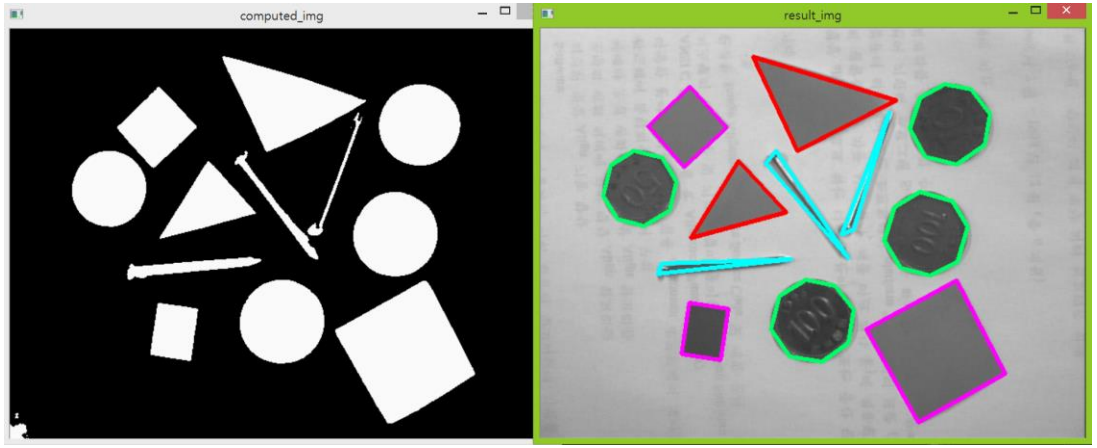


시스템 테스트 및 평가
-모양 파악 테스트 2
shape4



시스템 테스트 및 평가
-모양 파악 테스트 2
shape5



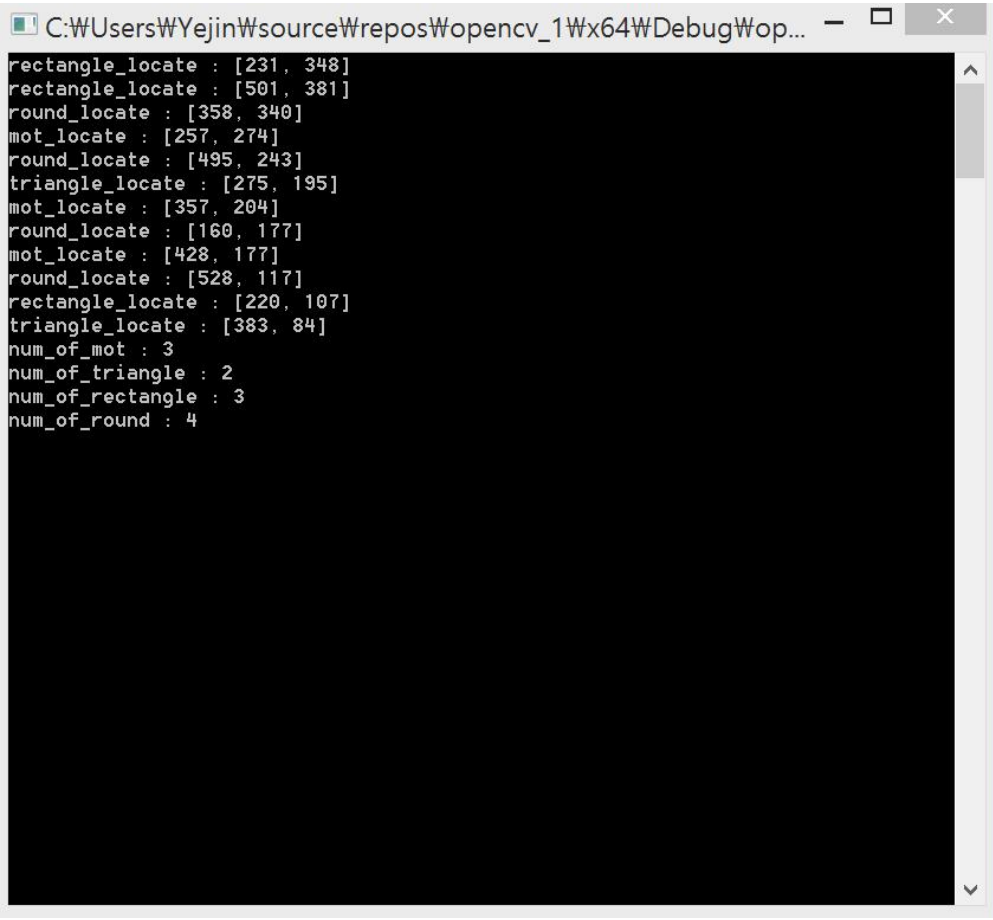


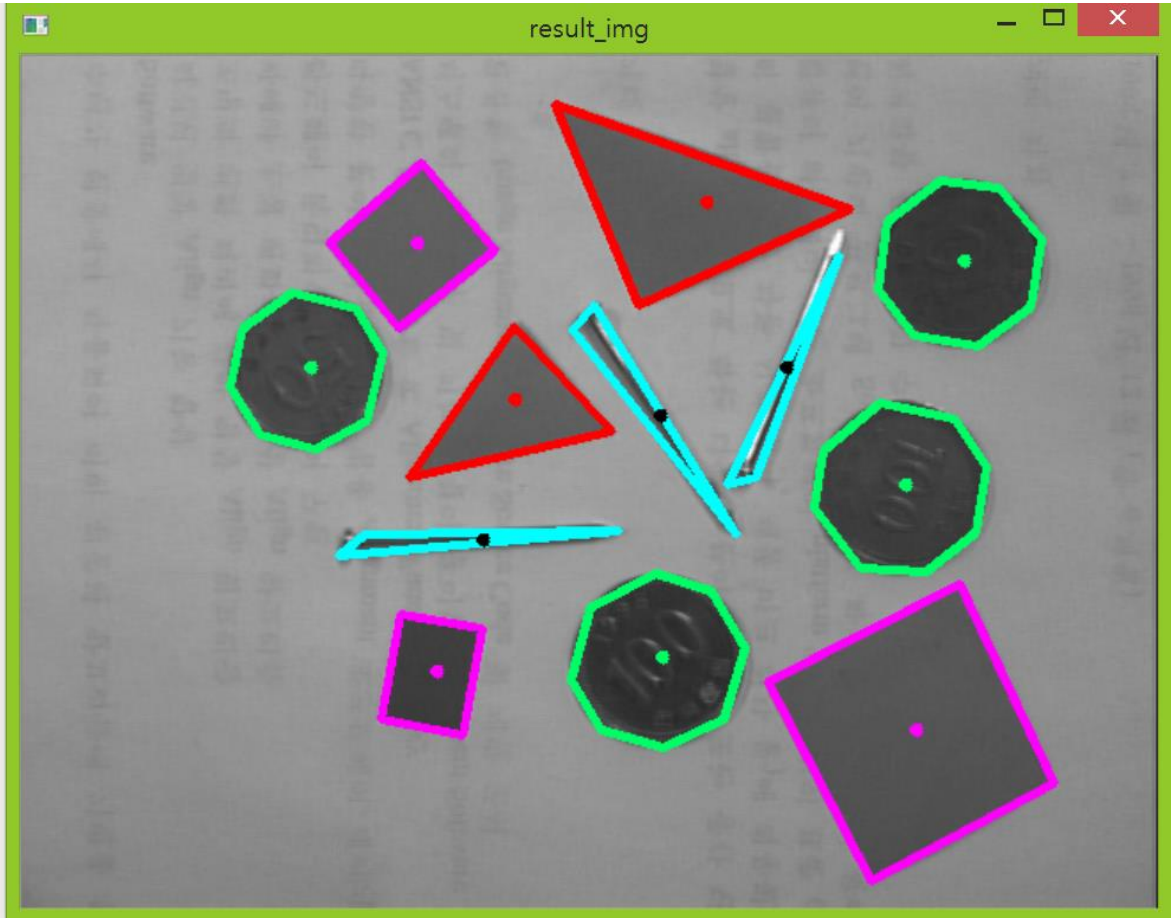
시스템 테스트 및 평가
-입력영상에 대한 결과

모양 검출을 끝낸 후 결과

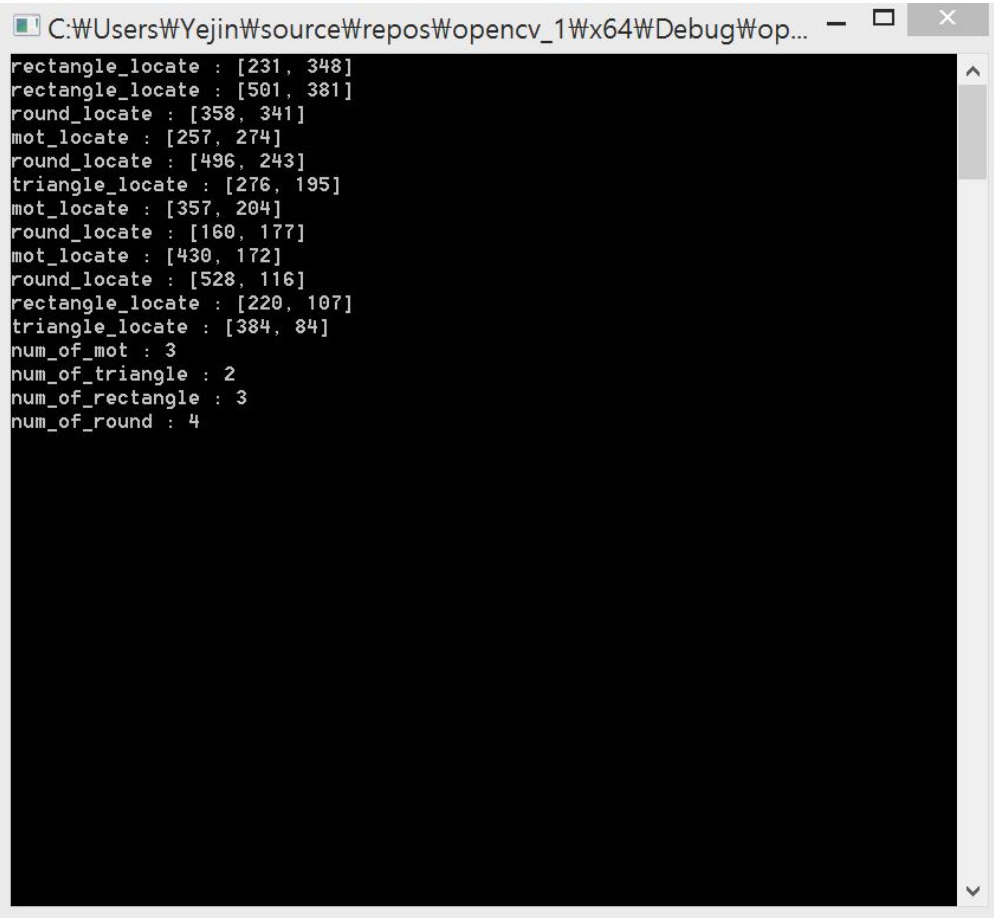
각 입력영상의 중심위치는 어디에 있으며 어떤 모양이 몇개 있는지 변수를 통해 확인할 수 있다.

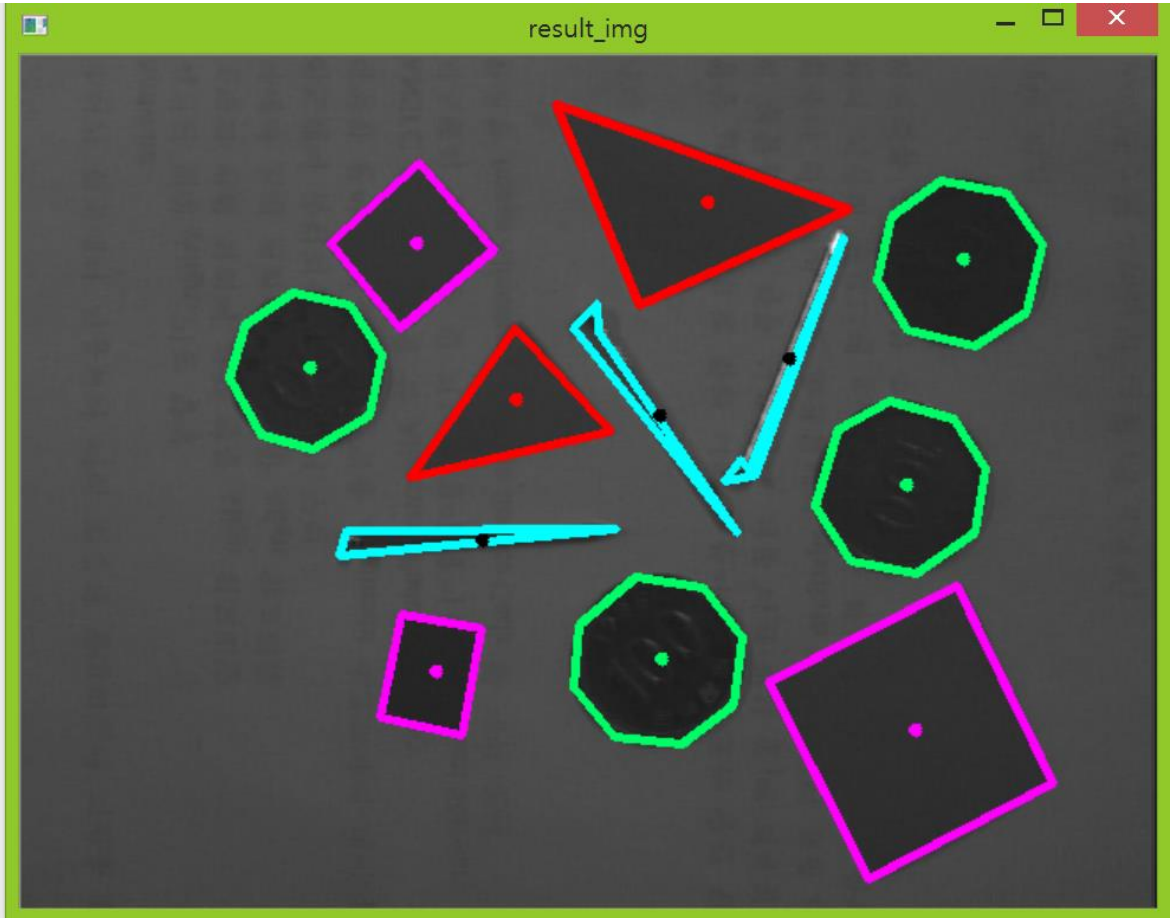
시스템 테스트 및 평가
-입력영상에 대한 결과
shape1



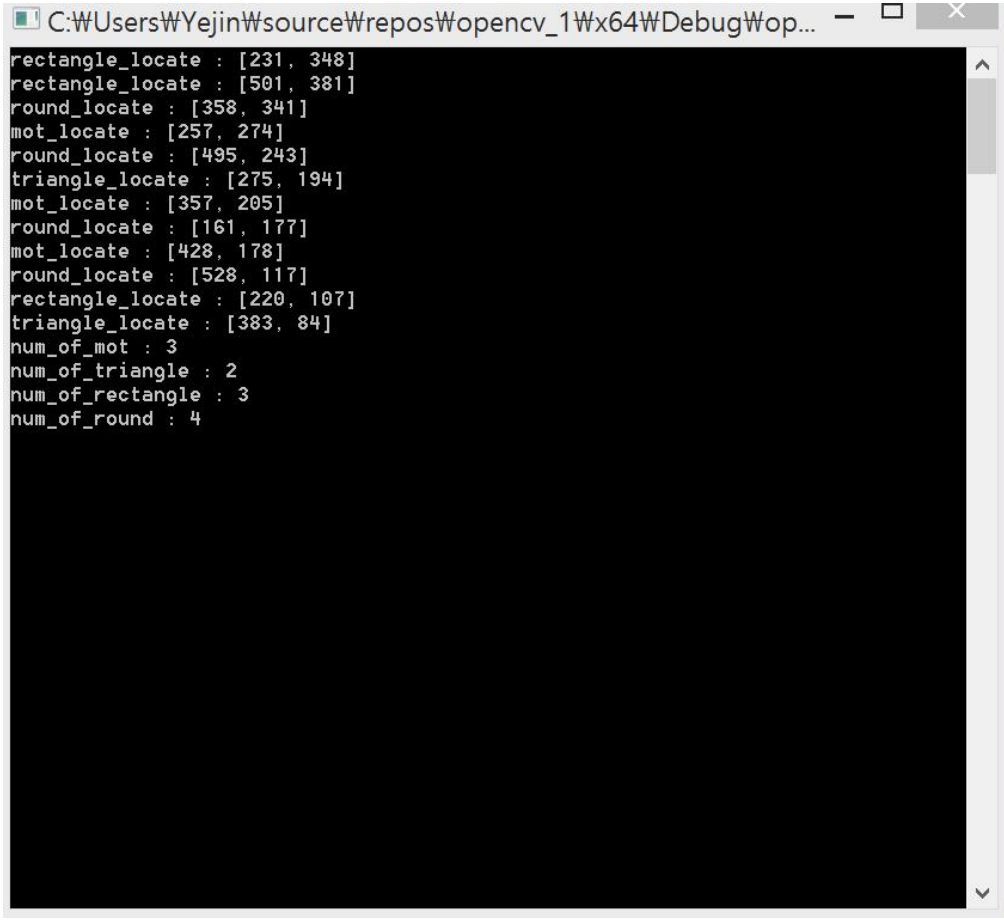


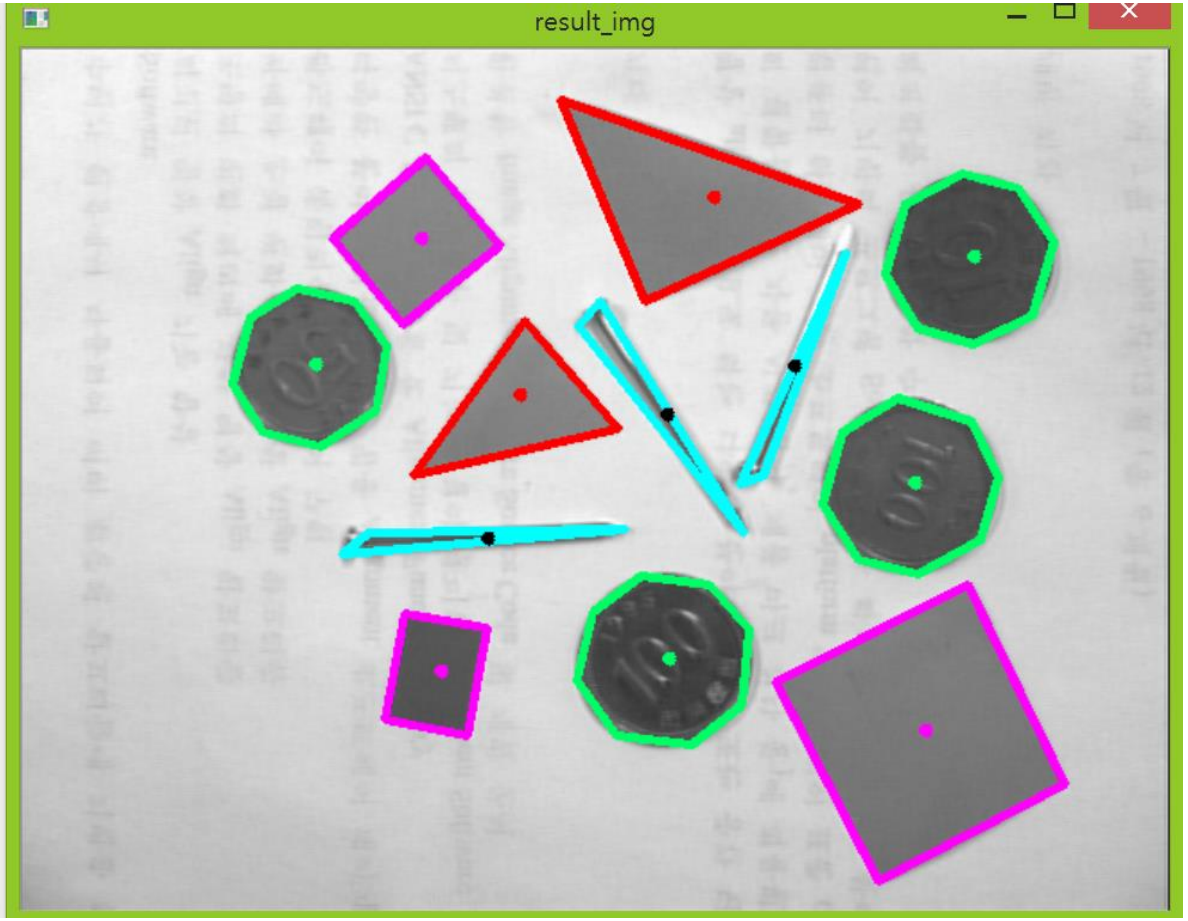
시스템 테스트 및 평가
-입력영상에 대한 결과
shape2



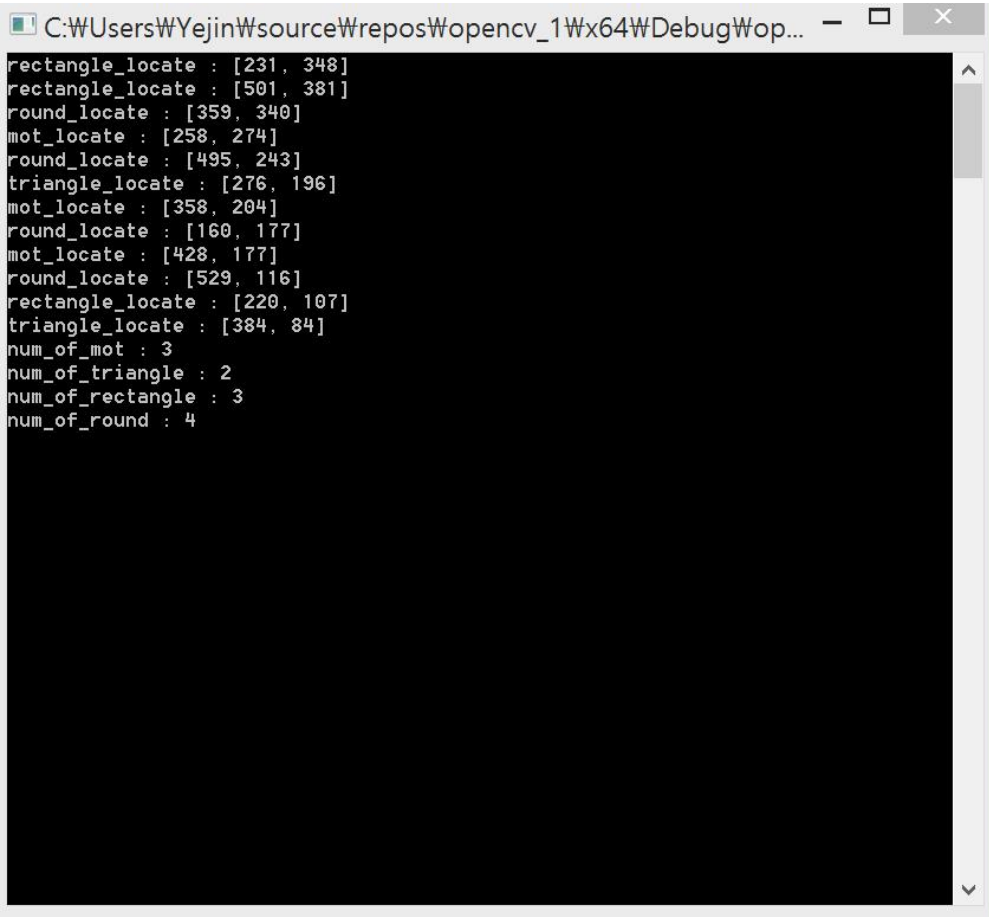


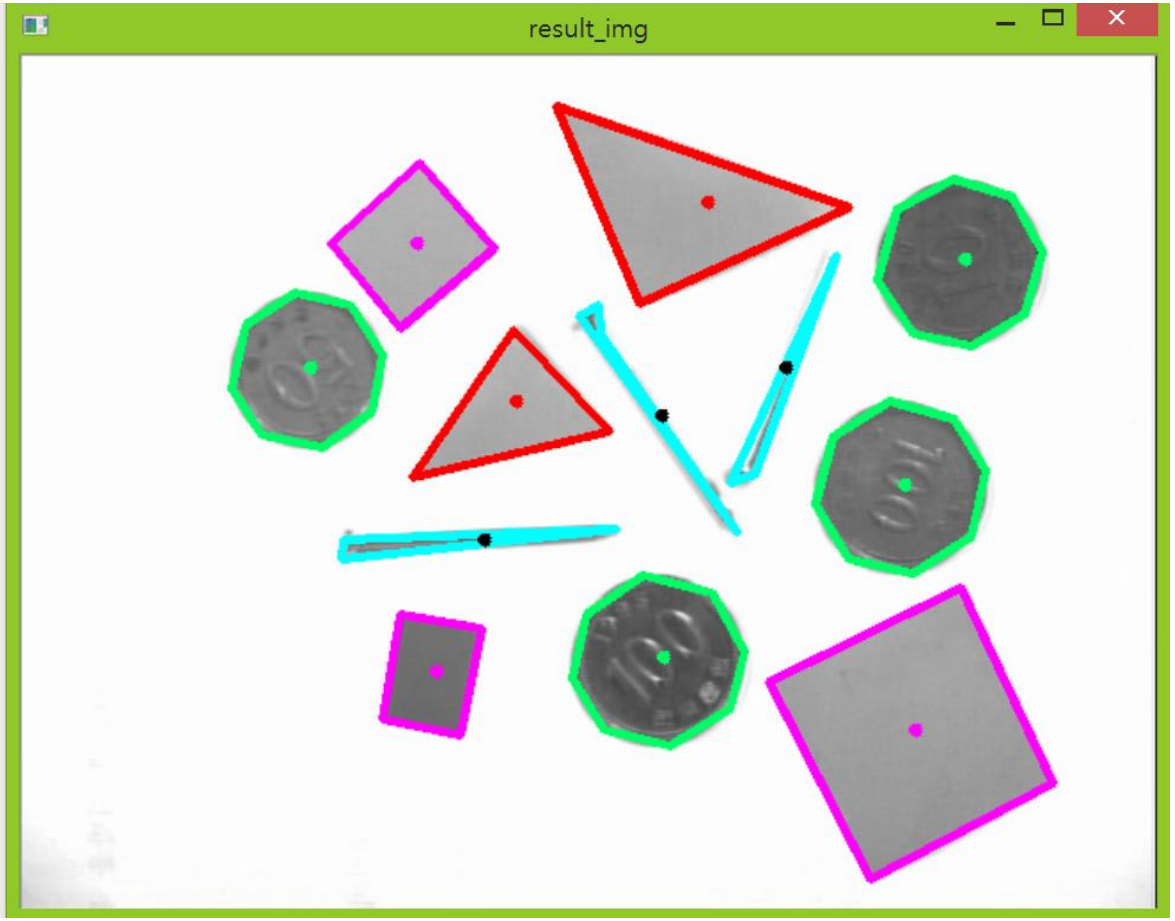
시스템 테스트 및 평가
-입력영상에 대한 결과
shape3



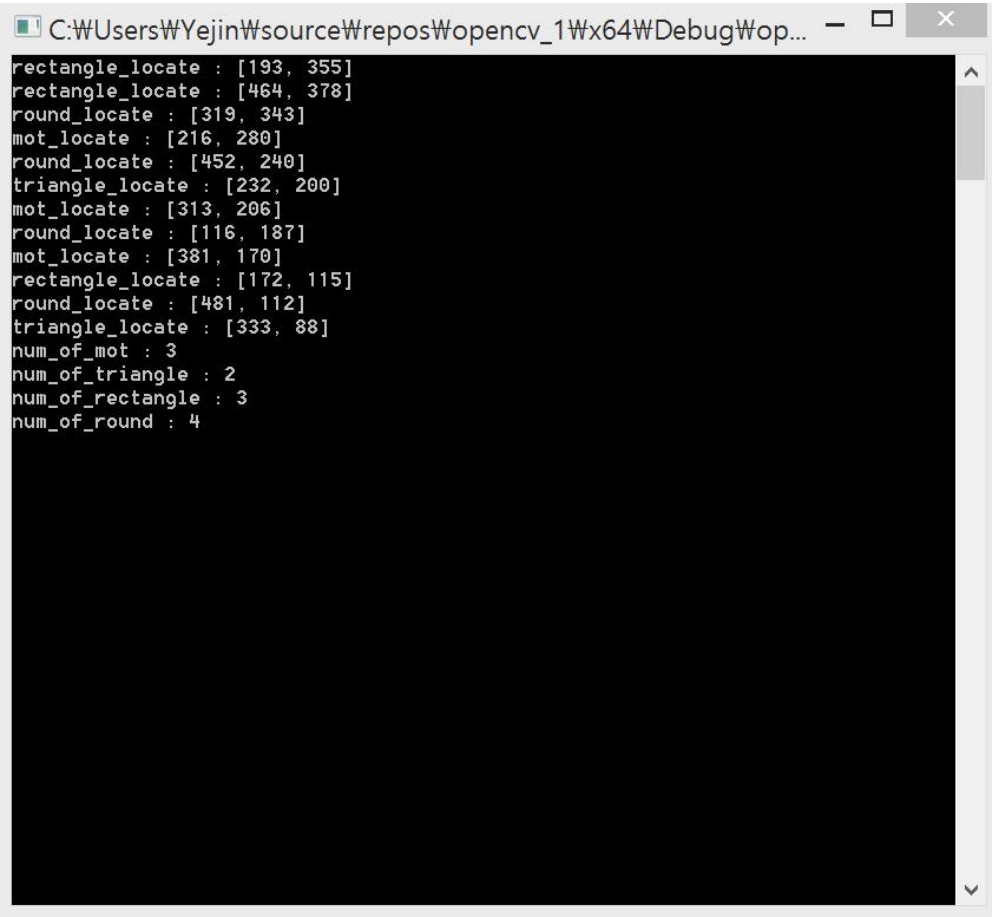


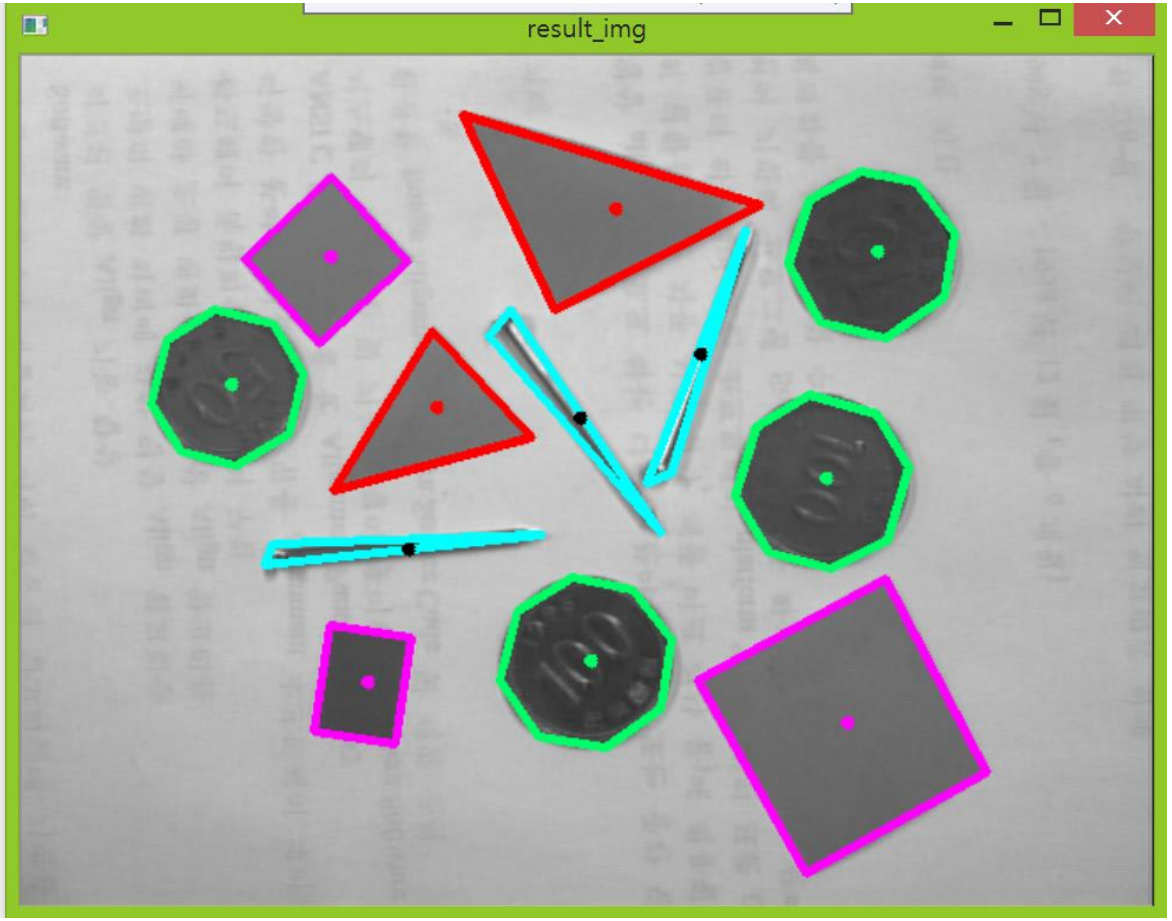
시스템 테스트 및 평가
-입력영상에 대한 결과
shape4





시스템 테스트 및 평가
-입력영상에 대한 결과
shape5





시스템 테스트 및 평가
-입력영상에 대한 결과

이를 통해 모든 입력영상에서 못의개수, 삼각형의 개수, 사각형의 개수, 원의 개수 등과 각 모양의 중심위치등을 성공적으로 검출해 냈음을 알 수 있다.

짧은 소견

이전에 과제로 했던 영상개선함수들을 응용해서 새로운 프로그램을 만들었다는 점이 의미깊다고 생각하고, 이번 과제를 통해 다양하고 편리한 opencv 내장함수들이 얼마나 많은지 조금이나마 체감할 수 있었다.

과제를 단계별로 진행하면서 코드를 수정할 일도 많이 생기고 시행착오도 많이 겪었지만, 뜻깊은 과정이었다고 느껴진다.