11장 영상 분할

(Image Segmentation)

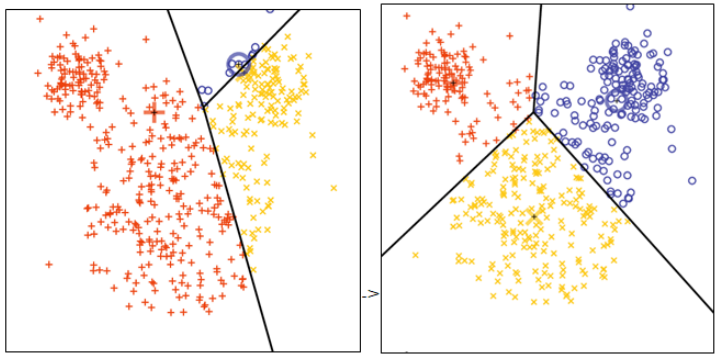
영상 분할

- Image segmentation
  - 영상 안의 화소를 의미 있는 영역(segment)으로 분할하는 것
  - Semantic segmentation

(cf) Image recognition: 영상 안의 사물/객체의 종류와 영역을 인지하는 것

영상 분할

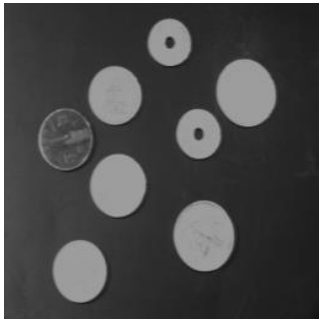
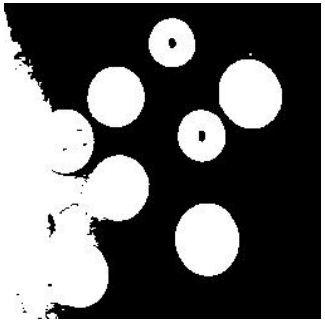
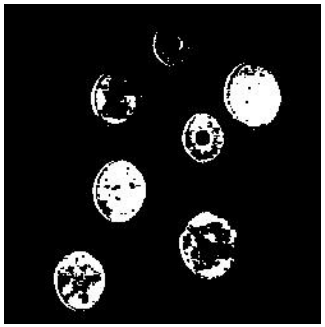
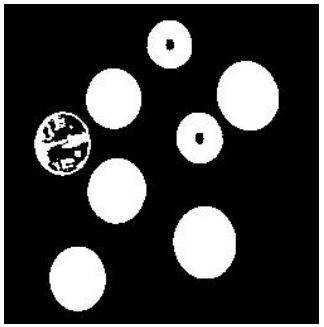
- 방법
  - 히스토그램을 이용하여 분할하는 방법
  - 클러스터링을 이용하여 분할하는 방법
  - 물체의 윤곽선(contour) 을 추출하여 분할하는 방법
  - 딥러닝을 이용하여 분할하는 방법

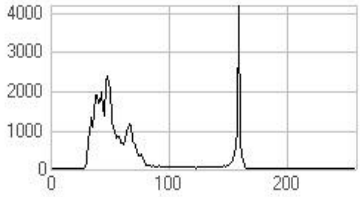


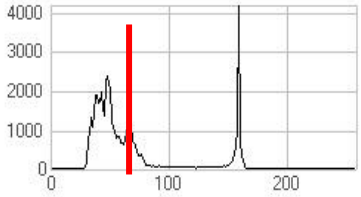
이진화

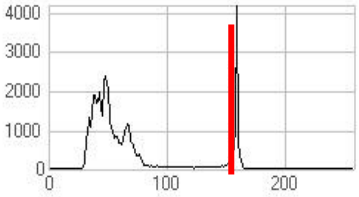
- 어떤 임계값 (threshold) 을 정하고 이 값을 기준으로 그레이스케일 영상을 이진 영상으로 만듦

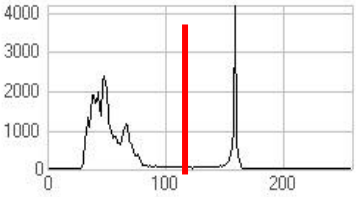
$$dst(x, y) \simeq \begin{cases} 1, & src(x, y) > T\text{인 경우} \\ 0, & src(x, y) \leq T\text{인 경우} \end{cases}$$









이진화

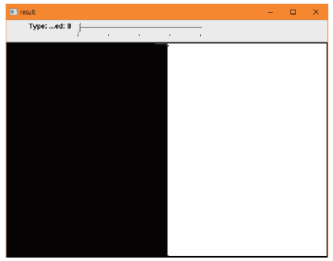
$$dst(x, y) = \begin{cases} maxVal & \text{if } src(x, y) > thresh \\ 0 & \text{otherwise} \end{cases}$$

- OpenCV 함수-threshold()

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

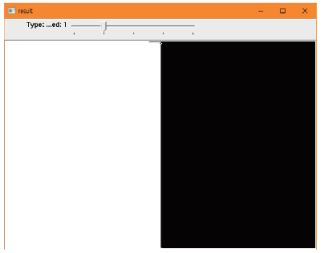
매개 변수	설명
src	입력 영상
dst	출력 영상
thresh	임계값
maxval	화소값이 임계값을 넘으면 부여되는 값
type	이진화 타입. 5개 중에서 하나이다. • THRESH_BINARY • THRESH_BINARY_INV • THRESH_TRUNC • THRESH_TOZERO • THRESH_TOZERO_INV

TRESH_BINARY



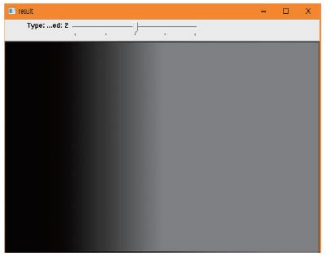
$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > thresh \\ maxVal & \text{otherwise} \end{cases}$$

TRESH_BINARY_INV



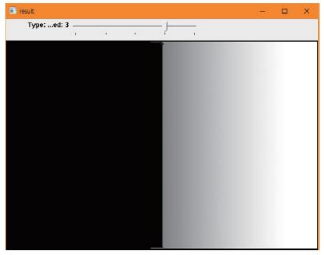
TRESH_TRUNC

$$dst(x, y) = \begin{cases} thresh & \text{if } src(x, y) > thresh \\ src(x, y) & \text{otherwise} \end{cases}$$



TRESH_TOZERO

$$dst(x, y) = \begin{cases} src(x, y) & \text{if } src(x, y) > thresh \\ 0 & \text{otherwise} \end{cases}$$



TRESH_TOZERO_INV

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > thresh \\ src(x, y) & \text{otherwise} \end{cases}$$

실습 1

```
int threshold_value = 128;
int threshold_type = 0;
const int max_value = 255;
const int max_binary_value = 255;
Mat src, src_gray, dst;

static void MyThreshold(int, void*)
{
    threshold(src, dst, threshold_value, max_binary_value, threshold_type);
    imshow("result", dst);
}

int main()
{
    src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    namedWindow("result", WINDOW_AUTOSIZE);
    createTrackbar("임계값", "result", &threshold_value, max_value, MyThreshold);
    MyThreshold(0, 0); // 초기화를 위하여 호출한다.
    waitKey();
    return 0;
}
```

실습 2

```
int threshold_value = 128;
int threshold_type = 0;
const int max_type = 4;
const int max_binary_value = 255;
Mat src, src_gray, dst;

static void MyThreshold(int, void*)
{
    threshold(src, dst, threshold_value, max_binary_value, threshold_type);
    imshow("result", dst);
}
int main()
{
    src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("src", src);
    namedWindow("result", WINDOW_AUTOSIZE);
    createTrackbar("Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n 3: To Zero \n
    4: To Zero Inverted", "result", &threshold_type, max_type, MyThreshold);

    MyThreshold(0, 0); // 초기화를 위하여 호출한다.
    waitKey();
    return 0;
}
```

실습 2


result
Type: ...4: 0
result
Type: ...ed: 1
result
Type: ...4: 2
result
Type: ...ed: 3

적응적 이진화

- 전역적 이치화
  - 프레임 전체 영역에 대하여 하나의 임계치로 이치화 수행
  - 영상 내 배경에 밝기의 불균형이 존재하는 경우 물체추출 어려움

- 적응적 이진화 (adaptive binarization)
  - 밝기 변동이 존재하는 영상의 각 부분에서, 밝기 변동을 반영하는 각각의 임계치를 결정

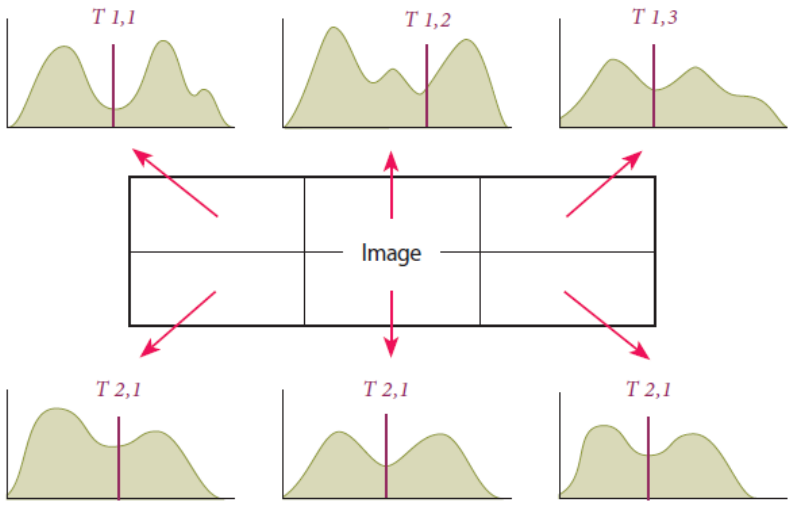
입력 영상

전역이진화

적응 이진화

적응적 이진화

- 개념



- 임계값 결정방법
  - Chow & Kaneko 방법
  - 지역 임계값 방법
    - T = 인접 화소등의 평균값 (mean)
    - T = 인접 화소들의 중간값 (median)
    - T = (max + min) / 2

적응적 이진화

- OpenCV 함수

```
void cv::adaptiveThreshold ( InputArray  src,
                             OutputArray dst,
                             double      maxValue,
                             int         adaptiveMethod,
                             int         thresholdType,
                             int         blockSize,
                             double      C
                           )
```

src 그레이 스케일 입력 이미지

dst 결과 이미지

maxValue threshold값보다 클 경우 결과 이미지의 해당 픽셀에 지정될 값

adaptiveMethod Adaptive thresholding algorithm

ADAPTIVE_THRESH_MEAN_C

또는 ADAPTIVE_THRESH_GAUSSIAN_C

thresholdType Thresholding type

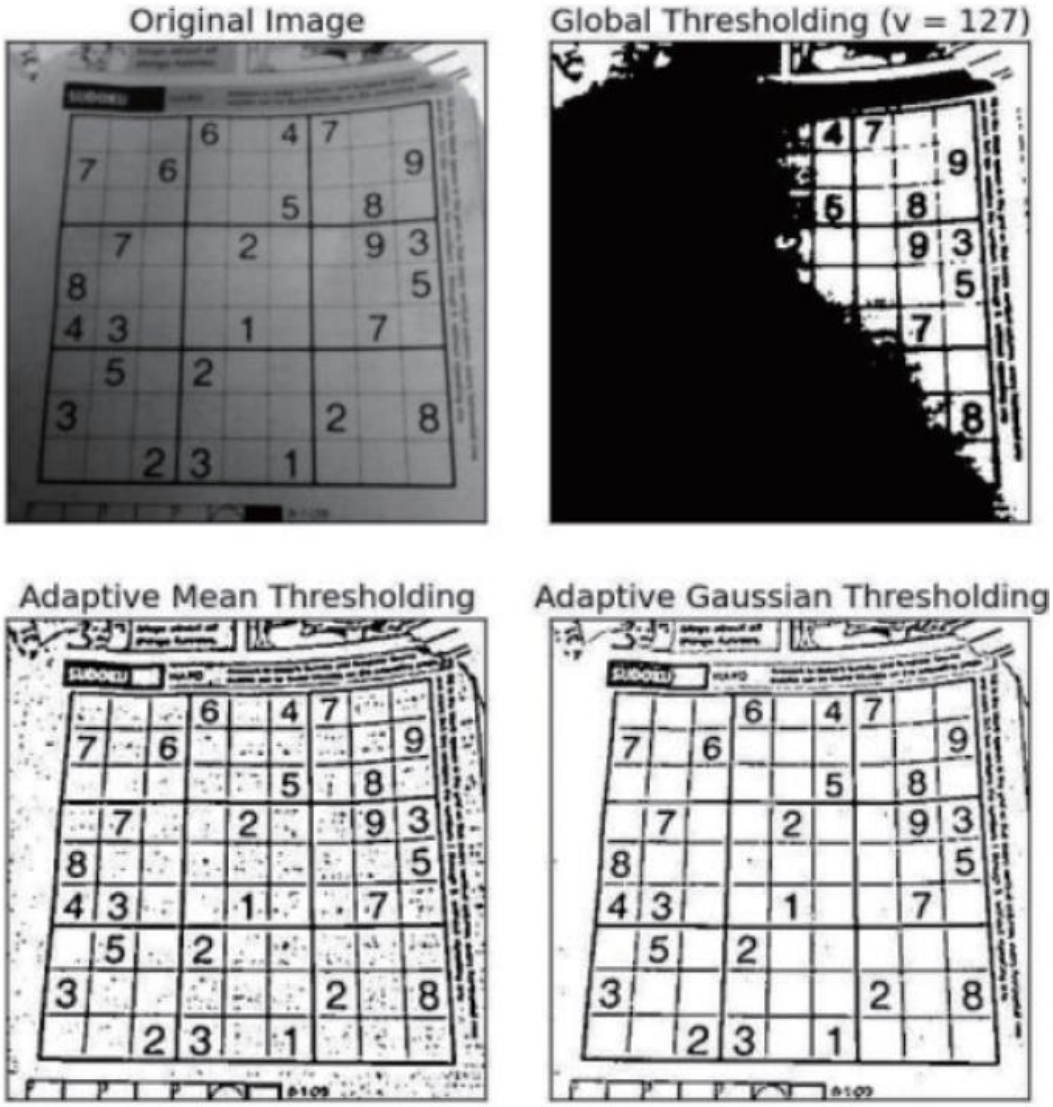
THRESH_BINARY(thredhold보다 크면 maxvalue)

또는 THRESH_BINARY_INV(threshold보다 작으면 maxvalue)

blockSize threshold값을 계산하기 위해 사용되는 블록 크기 : 3,5,7 등의 값

c 계산된 평균으로부터 뺄 상수값

적응적 이진화



출처: OpenCV 튜토리얼

실습 3

```
int main()
{
    Mat src = imread("d:/book1.jpg", IMREAD_GRAYSCALE);
    Mat img, th1, th2, th3, th4;
    medianBlur(src, img, 5);
    threshold(img, th1, 127, 255, THRESH_BINARY);
    adaptiveThreshold(img, th2, 255, ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 11, 2);
    adaptiveThreshold(img, th3, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, 11, 2);

    imshow("Original", src);
    imshow("Global Thresholding", th1);
    imshow("Adaptive Mean", th2);
    imshow("Adaptive Gaussian", th3);
    waitKey();
    return 0;
}
```

실습 3

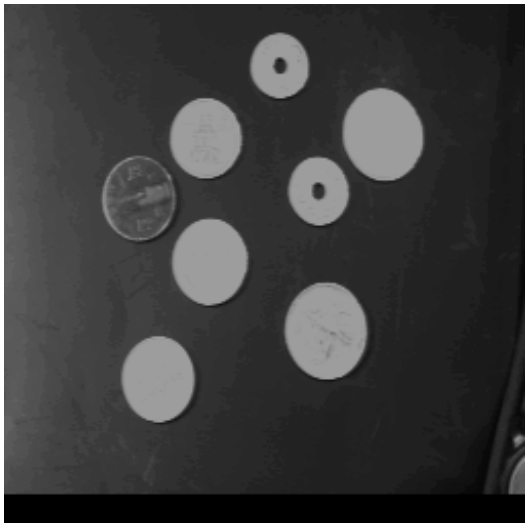
전역이진화

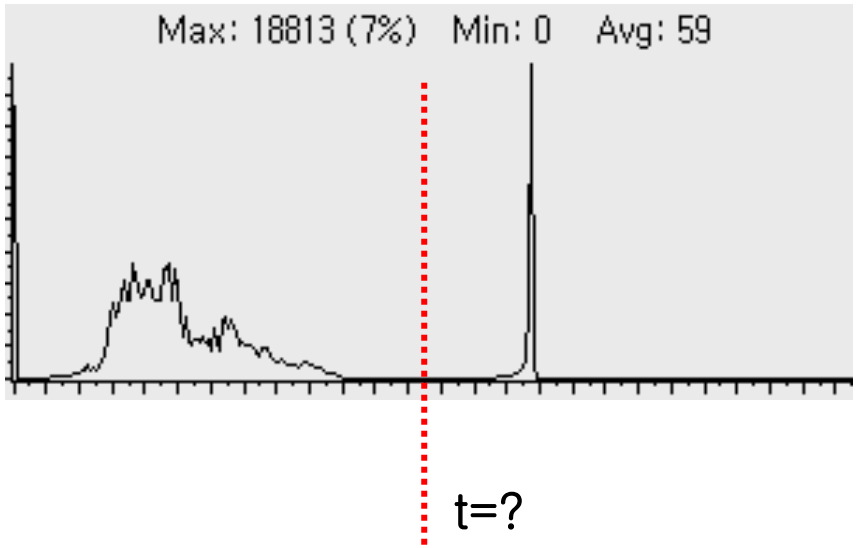
Adaptive Mean 방법

Adaptive Gaussian 방법

자동 이진화

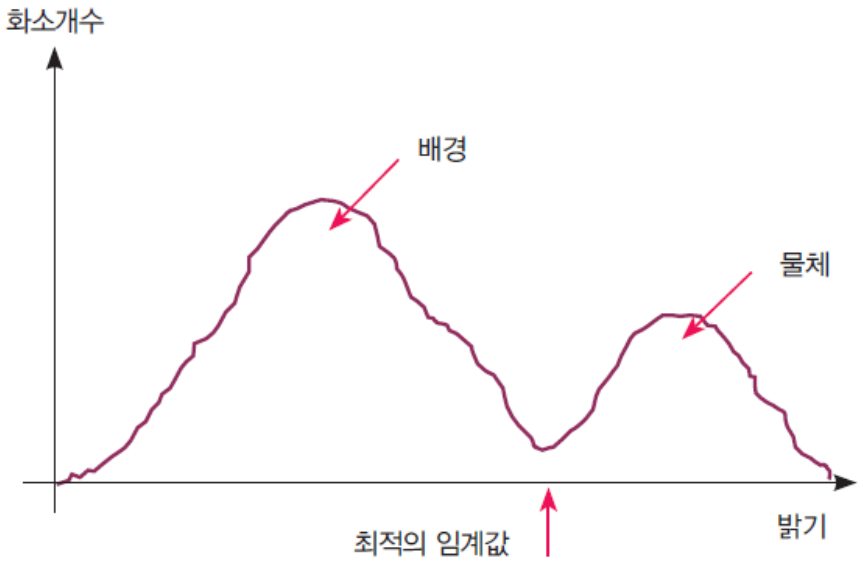
- Automatic Thresholding
  - 입력영상으로 부터 자동으로 임계치 (threshold value) 결정
  - 영상 분할 성능에 큰 영향을 줌





Otsu 이진화

- Otsu 이진화
  - 두 개의 peak (배경 + 물체) 로 구성된 히스토그램
  - 각 peak 분포를 정규분포로 모델링
  - 총 분산을 최소화시키는 임계값 (t) 를 찾는방법



Otsu 이진화

- 히스토그램의 총 분산 계산 방법

$P(i) = \#\{(r,c) \mid \text{Image}(r,c) == i\} / (\#R \times C)$: 밝기 i의 히스토그램 확률

$q_1(t) = \sum_{i=1}^{t} P(i)$: t 보다 작은 픽셀그룹 (왼쪽분포) 의 누적 확률

$q_2(t) = \sum_{i=t+1}^{I} P(i)$: t 보다 큰 픽셀그룹 (오른쪽분포) 의 누적 확률

$\mu_1(t) = \sum_{i=1}^{t} iP(i)/q_1(t)$: 왼쪽분포의 평균

$\mu_2(t) = \sum_{i=t+1}^{I} iP(i)/q_2(t)$: 오른쪽분포의 평균

$\sigma_1(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 P(i)/q_1(t)$: 왼쪽분포의 분산

$\sigma_2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 P(i)/q_2(t)$: 오른쪽분포의 분산

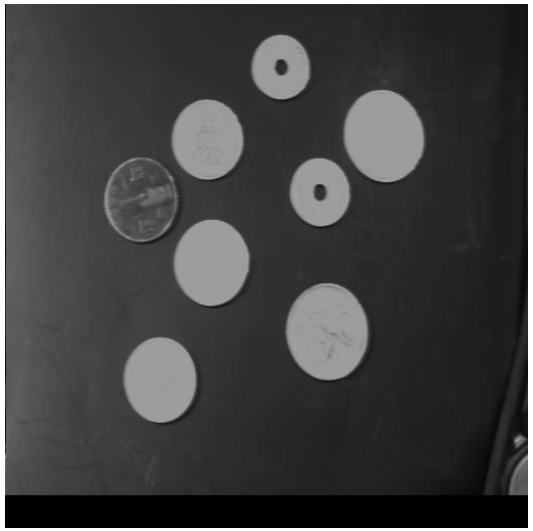
$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

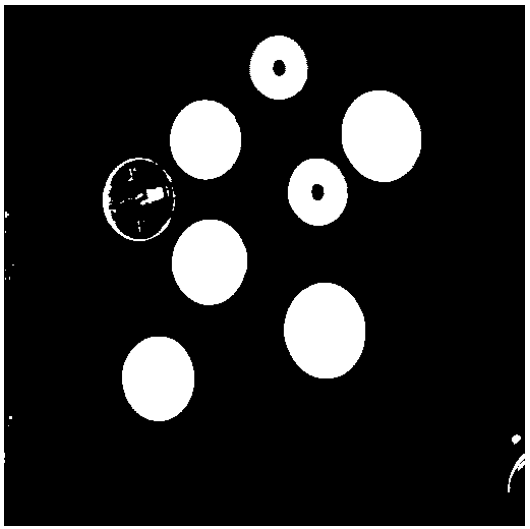
: 총 분산

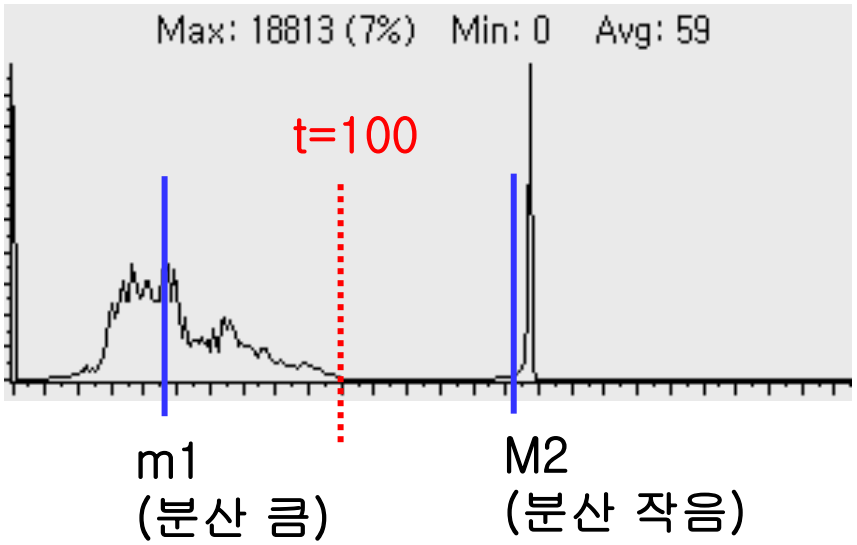
Otsu 이진화

```
void CWinTestDoc::Otzu_Threshold(unsigned char *orgImg, unsigned
char *outImg, int height, int width)
{
  register int i,t;

  // Histogram 계산
  inthist[256];
  float   prob[256];
  for(i=0; i<256; i++) { hist[i]=0; prob[i] = 0.0f; }
  for(i=0; i<height*width; i++) hist[(int)orgImg[i]]++;
  for(i=0; i<256; i++) prob[i] = (float)hist[i]/(float)(height*width);


  float wsv_min = 1000000.0f;
  float wsv_u1, wsv_u2, wsv_s1, wsv_s2;
  int wsv_t;

  for(t=0; t<256; t++)
  {
  // 누적확률 q1, q2 계산
     float q1 = 0.0f, q2 = 0.0f;

     for(i=0; i<t; i++)
           q1 += prob[i];
     for(i=t; i<256; i++) q2 += prob[i];

     if(q1==0 || q2==0) continue;

     // 평균 u1, u2 계산
     float u1=0.0f, u2=0.0f;
     for(i=0; i<t; i++)   u1 += i*prob[i]; u1 /= q1;
     for(i=t; i<256; i++) u2 += i*prob[i]; u2 /= q2;
     // 분산 s1, s2 계산
     float s1=0.0f, s2=0.0f;
     for(i=0; i<t; i++)   s1 += (i-u1)*(i-u1)*prob[i]; s1 /= q1;
     for(i=t; i<256; i++) s2 += (i-u2)*(i-u2)*prob[i]; s2 /=
     q2;
     //총 분산 계산
     float wsv = q1*s1+q2*s2;

     if(wsv < wsv_min)
     {
        wsv_min = wsv;
        wsv_t = t;                                  // 최소치 저장
        wsv_u1 = u1; wsv_u2 = u2;
        wsv_s1 = s1; wsv_s2 = s2;
     }
  }

  // thresholding
  for(i=0; i<height*width; i++)
         if(orgImg[i]<wsv_t)
               outImg[i]=0;
          else
               outImg[i]=255;
  }
```

Otsu 이진화







실험결과

- Optimal threshold: 100
- Group 1 Mean: 46.575
  Group 2 Mean: 152.622
- Group 1 Variance: 448.173
  Group 2 Variance: 133.493

실습 4

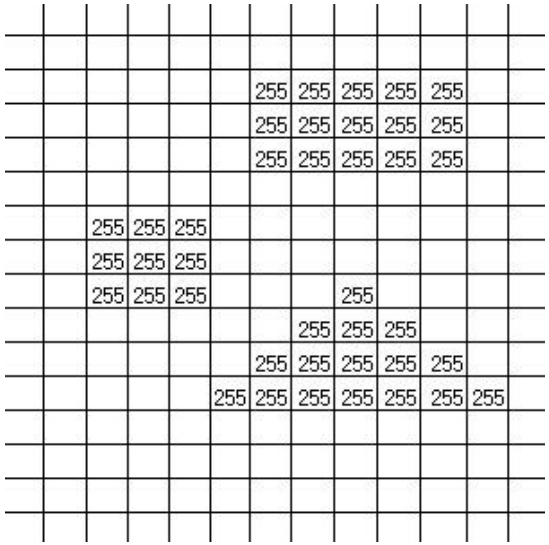
```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat blur, th1, th2, th3, th4;
    threshold(src, th1, 127, 255, THRESH_BINARY);
    threshold(src, th2, 0, 255, THRESH_BINARY | THRESH_OTSU);

    Size size = Size(5, 5);
    GaussianBlur(src, blur, size, 0);
    threshold(blur, th3, 0, 255, THRESH_BINARY | THRESH_OTSU);

    imshow("Original", src);
    imshow("Global", th1);
    imshow("Ostu", th2);
    imshow("Ostu after Blurring", th3);
    waitKey();
    return 0;
}
```

실행 결과

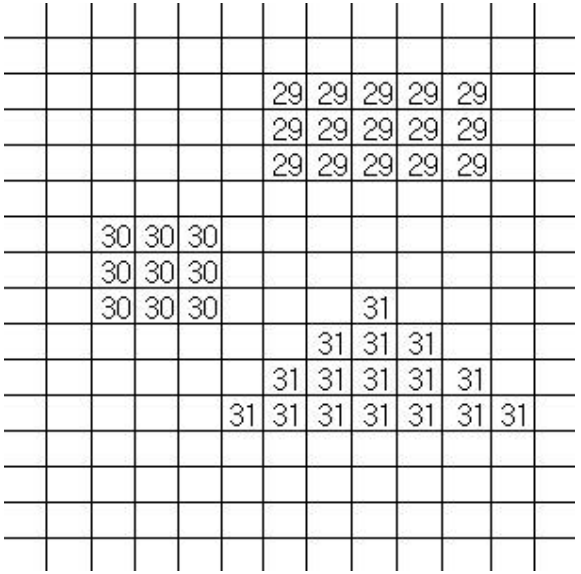
연결 성분 레이블링

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

(이진화 영상)

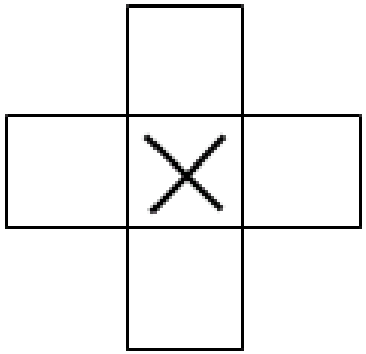


(레이블링된 영상)

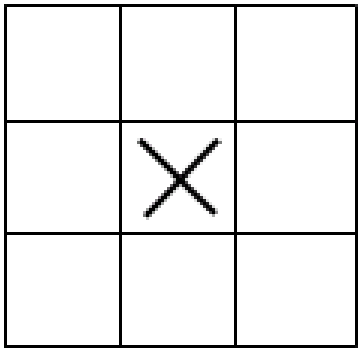


연결 성분 레이블링

- 픽셀의 인접성 (neighborhood)



(a) 4-근방 화소들



(b) 8-근방 화소들

	(1)	
(2)	x	(0)
	(3)	

(3)	(2)	(1)
(4)	x	(0)
(5)	(6)	(7)

연결 성분 레이블링

1. 재귀 알고리즘

- 연결 성분의 첫 번째 원소를 찾으면, 이 원소와 연결된 모든 화소의 레이블 지정
- Grassfire 알고리즘
- 내부 Stack 사용

① 영상을 스캔하여 레이블링되어 있지 않은 전경 화소를 찾아 새로운 레이블 *L*을 부여한다.

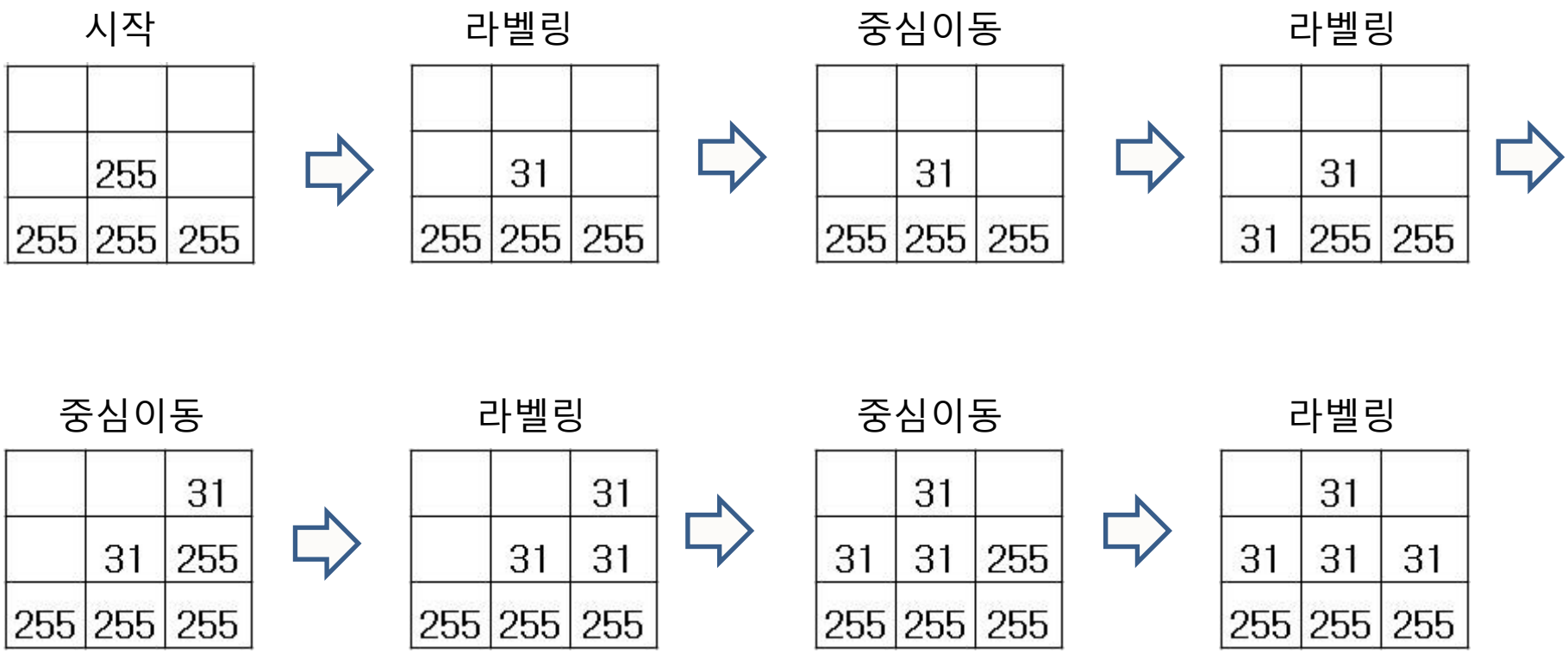
② 재귀적으로 레이블 *L*을 모든 이웃의 전경 화소(4-이웃 또는 8-이웃)에 부여한다.

③ 더 이상 레이블링되어 있지 않은 전경 화소가 없으면 멈춘다.

④ 단계 ①로 간다.

연결 성분 레이블링

1. 재귀 알고리즘 (계속)



연결 성분 레이블링

2. 2-패스 알고리즘 (Hoshen-Kopelman)

- 영상을 2번 스캔
  - 1st pass: 각 화소에 임시 레이블 부여 & 등가 테이블 기록
  - 2nd pass: 등가 테이블의 최소값으로 임시 레이블 대체

Algorithm 11.2

알고리즘: 4-연결을 이용한 2-패스 연결 성분 레이블링

첫 번째 패스

① 영상을 위에서 아래로, 왼쪽으로 오른쪽으로 스캔한다.

② 현재 화소가 1이면

(a) 왼쪽 화소만이 레이블을 가지면 그 레이블을 현재 화소에 부여한다.

(b) 위쪽 화소만이 레이블을 가지면 그 레이블을 현재 화소에 부여한다.

(c) 위쪽과 왼쪽 화소가 다른 레이블을 가지면 이 사실을 등가 테이블에 기록한다.

(d) 위의 경우가 아니면 이 화소에 새로운 레이블을 부여한다.

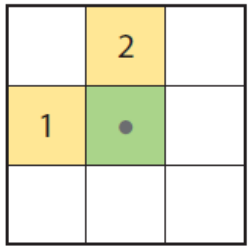
③ 고려해야 할 더 이상의 화소가 없으면 멈춘다.

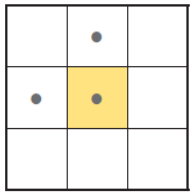
두 번째 패스

① 등가 테이블에서 각 등가 레이블 집합에서 최소의 레이블을 찾는다.

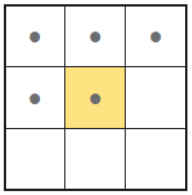
② 영상을 조사하여 레이블을 등가 집합의 최소 레이블로 바꾼다.

4-connected





4 연결

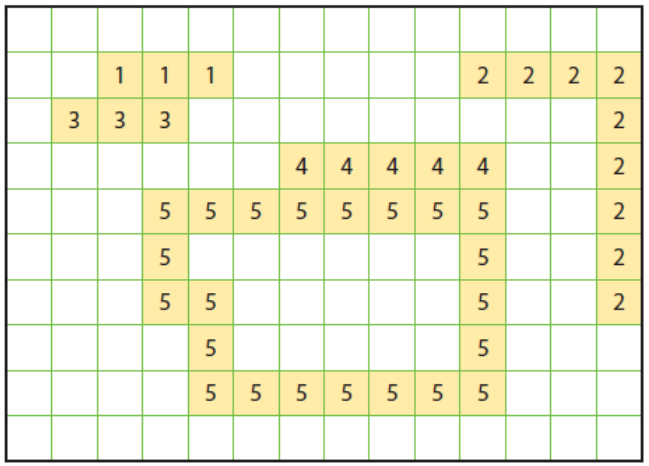


8 연결

연결 성분 레이블링

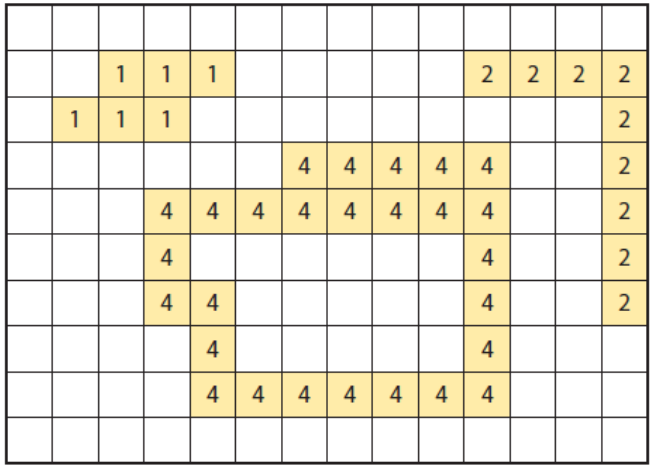
2. 2-패스 알고리즘: 예

1st 패스



{ 1, 3 }
{ 4, 5 }

2nd 패스



연결 성분 레이블링

2. 2-패스 알고리즘: 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

2-패스 알고리즘

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

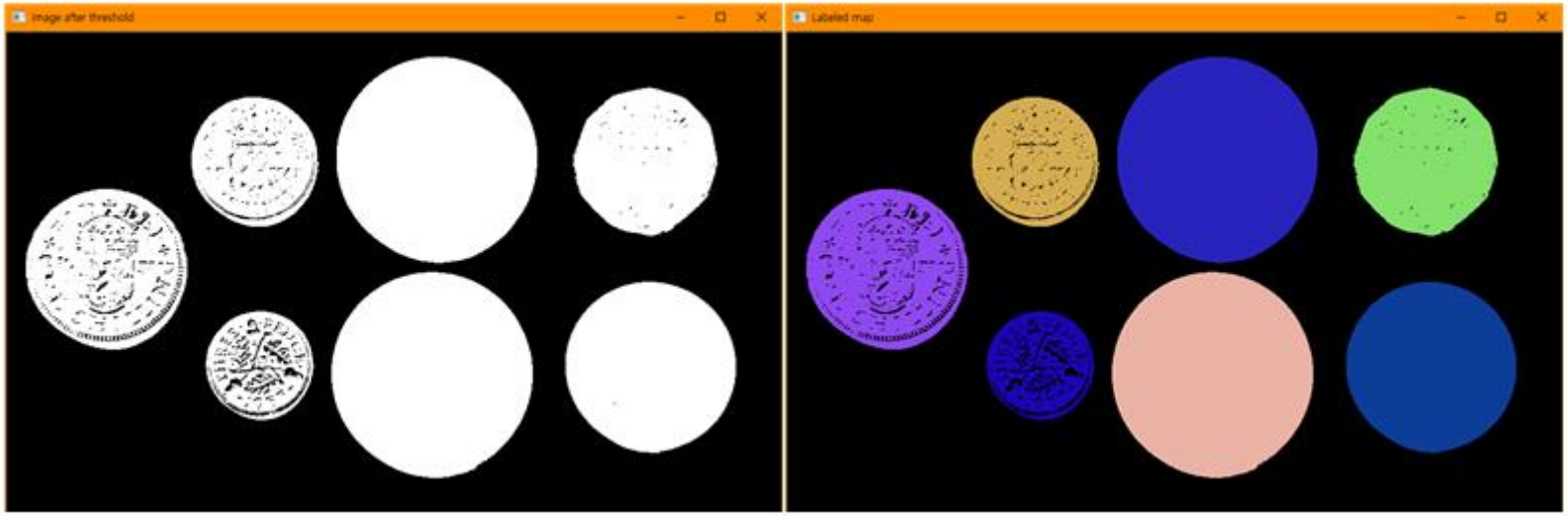
매개 함수	설명
image	입력 영상
labels	레이블 영상
connectivity	8-연결성이나 4-연결성
Itype	출력 영상의 레이블 타입 CV_32S 또는 CV_16U
statsv	각 레이블에 대한 통계 자료
반환값	레이블의 개수

실습 5

```
int main() {
    Mat img, img_edge, labels, centroids, img_color, stats;
    img = cv::imread("d:/coins.png", IMREAD_GRAYSCALE);

    threshold(img, img_edge, 128, 255, THRESH_BINARY_INV);
    imshow("Image after threshold", img_edge);

    int n = connectedComponentsWithStats(img_edge, labels, stats, centroids);

    vector<Vec3b> colors(n + 1);
    colors[0] = Vec3b(0, 0, 0);
    for (int i = 1; i <= n; i++) {
        colors[i] = Vec3b(rand() % 256, rand() % 256, rand() % 256);
    }
    img_color = cv::Mat::zeros(img.size(), CV_8UC3);
    for (int y = 0; y < img_color.rows; y++)
        for (int x = 0; x < img_color.cols; x++)
        {
                int label = labels.at<int>(y, x);
                img_color.at<cv::Vec3b>(y, x) = colors[label];
        }

    cv::imshow("Labeled map", img_color);
    cv::waitKey();
    return 0;
}
```

실습 5

실행결과

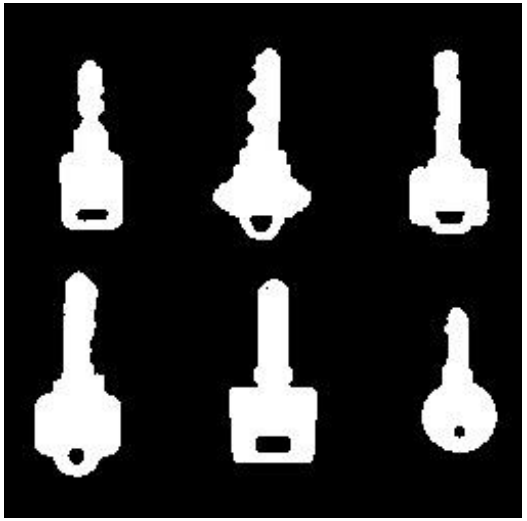


윤곽선 추출

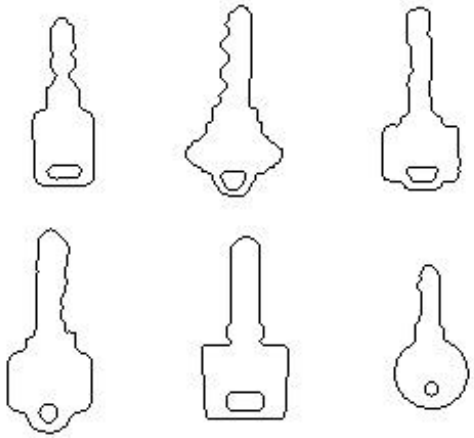
- Contouring (Edge following)
  - 이진화된 영상 또는 레이블링된 영상에서 영역의 경계를 추적하여, 경계 픽셀의 순서화된 정보 (chain code) 추출
  - 물체의 기하학적 특징 추출에 사용

(원 영상)

(이진화 영상)



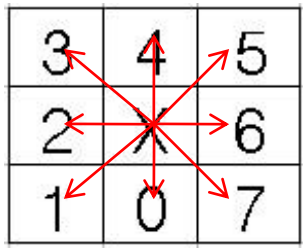
(경계 영상)



윤곽선 추출

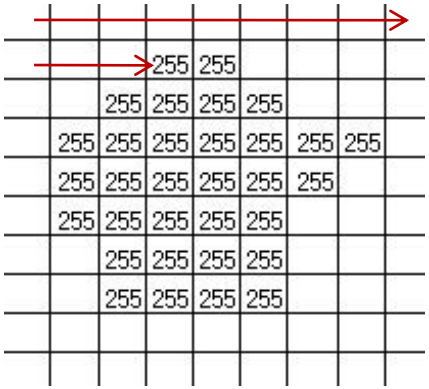
- 알고리즘

주위 픽셀번호

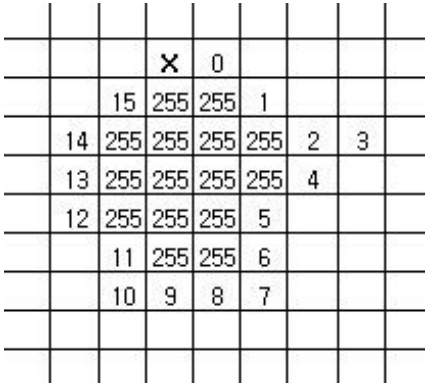


탐색순서

4→5 → 6 → … → 3



scan



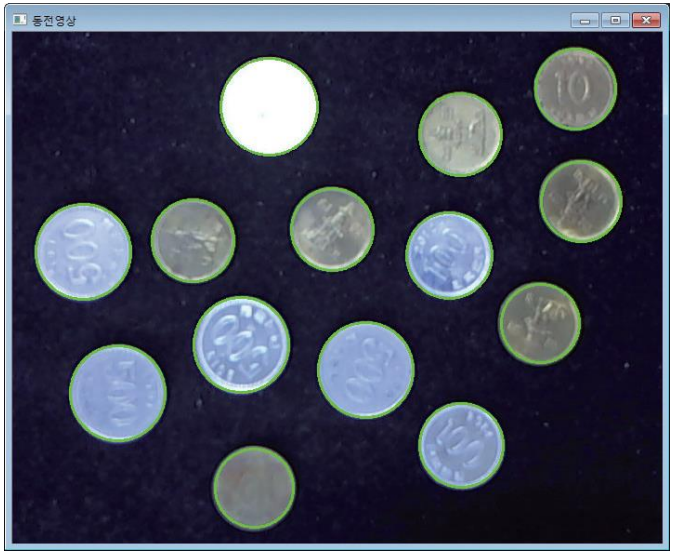
윤곽선 추출

- OpenCV 함수

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

실습 6

동전 분할 프로그램

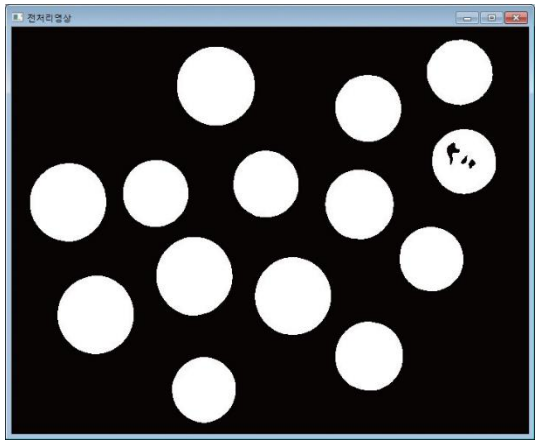


실습 6

```
int main()
{
    int  coin_no = 20;                                          // 동전 파일 번호
    String  fname = format("../image/%2d.png", coin_no);
    Mat  image  = imread(fname, 1 );
    CV_Assert(image.data);

    Mat th_img = preprocessing(image);                          // 전처리 수행
    vector<RotatedRect> circles = find_coins(th_img);           // 동전객체 회전사각형

    for (int i = 0; i < circles.size(); i++) {
        float radius = circles[i].angle;                        // 동전객체 반지름
        circle(image, circles[i].center, radius, Scalar(0, 255, 0), 2); // 동전 표시
    }

    imshow("전처리영상", th_img);
    imshow("동전영상", image);
    waitKey();
    return 0;
}
```

실습 6

```
Mat  preprocessing(Mat img)                                          // 전처리 함수
{
     Mat gray, th_img;
     cvtColor(img, gray, CV_BGR2GRAY);                               // 명암도 변환
     GaussianBlur(gray, gray, Size(7, 7), 2, 2);                     // 블러링

     threshold(gray, th_img, 130, 255, THRESH_BINARY | THRESH_OTSU);  // 이진화
     morphologyEx(th_img, th_img, MORPH_OPEN, Mat(), Point(-1, -1), 1);  // 열림연산

     return th_img;
}
```



실습 6

```
vector<RotatedRect>  find_coins(Mat img)
{
     vector<vector<Point> > contours;
     // 외곽선 검출
     findContours(img.clone(), contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

     vector<RotatedRect> circles;
     for (int i = 0; i< (int)contours.size(); i++)
     {
          RotatedRect  mr = minAreaRect(contours[i]);          // 외곽선의 최소영역 사각형
          mr.angle = (mr.size.width + mr.size.height) / 4.0f;     // 반지름을 각도에 저장

          if (mr.angle > 18) circles.push_back(mr);     // 반지름이 18 이상인 개체 저장
     }
     return circles;
}
```