Chapter

10

영상분할 및 특징처리

영상 분할(Image Segmentation)

❖ 적절한 분할이란?

- 저분할과 과분할

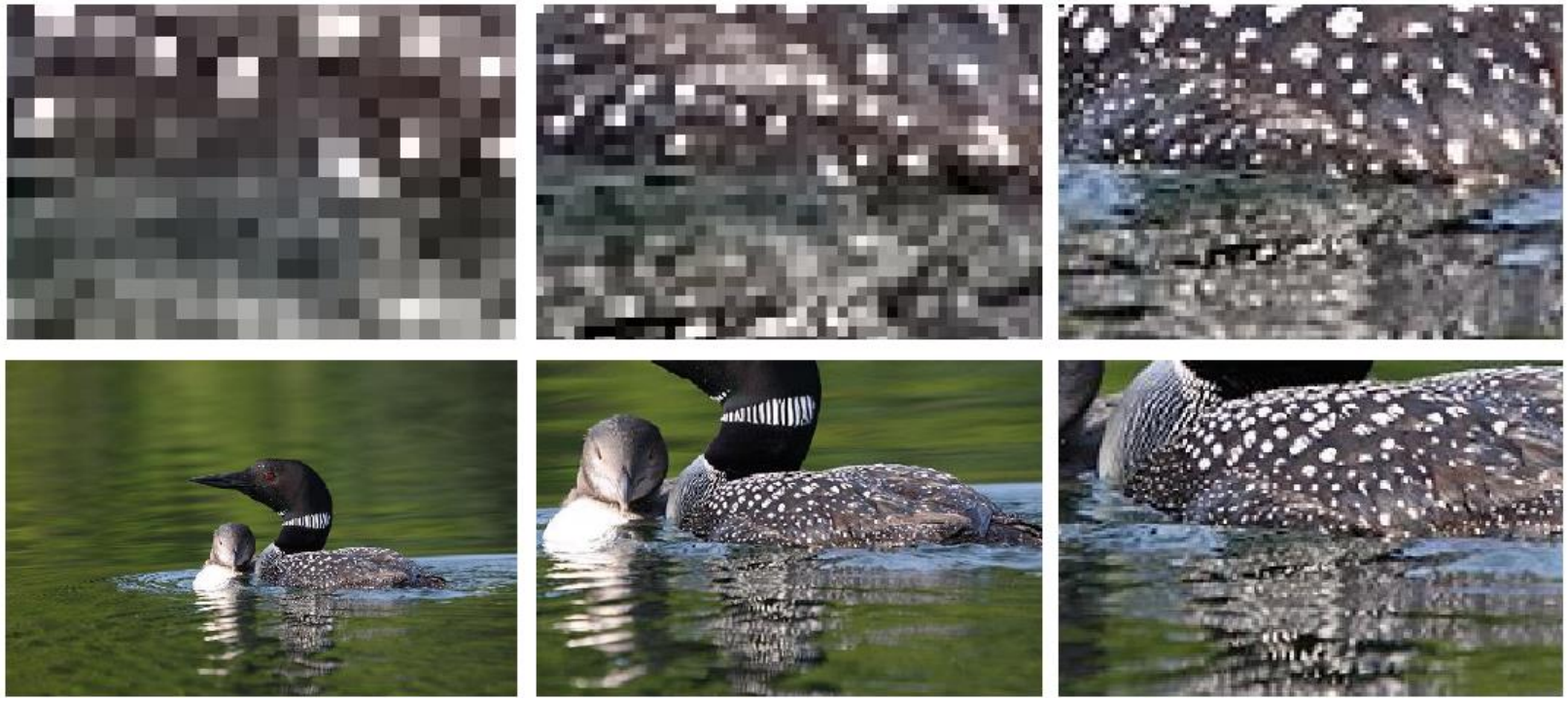
❖ 사람 vs 컴퓨터

- 사람은 선택적 주의집중과 능동 비전 기능을 가지며, 분할 과정에서 고급 지식 사용
  - 물체 모델, 지식, 의도 등
- 컴퓨터 비전은 그런 수준에 이르지 못함
  - 분할이 끝나야만 고급 지식을 이용하여 인식을 수행

❖ 분할의 어려움

- 좁은 영역(이웃 화소 몇 개를 보고)만을 가지고 자신이 영역의 내부인지 경계인지, 어떤 영역에 속하는지 판단할 수 있을까?

→ 전역 연산 필요성



❖ 영역분할(Region Segmentation) 알고리즘

- (다중)임계화를 이용한 알고리즘
- 군집화(clustering)을 이용한 알고리즘
  - k-means
  - mean-shift, cam-shift
- 그래프 방법에 의한 알고리즘
- 분할합병(split and merge) 알고리즘
- 워터세드(watershed) 알고리즘
- 대화식 물체분할
- ……

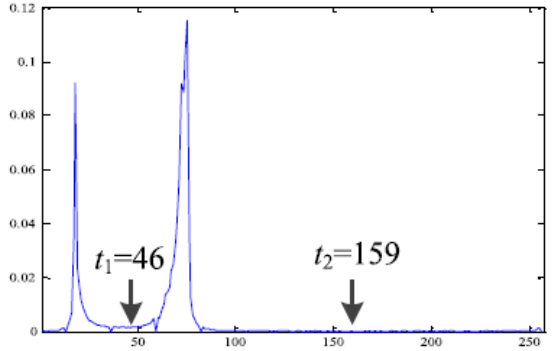
Multi-thresholding

❖ Otsu 알고리즘

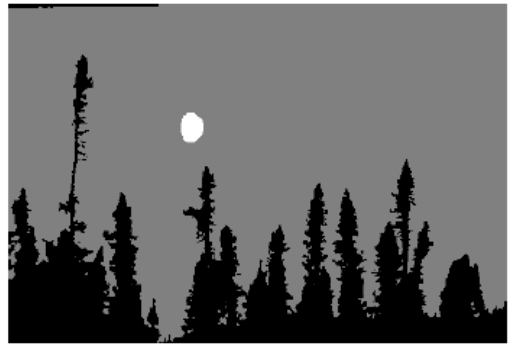
- 이치화를 위한 매우 훌륭한 알고리즘
- 명암단계가 2개 이상인 경우에는 적용불가

❖ 다중화로 확장

(a) 원래 영상



(b) 명암 히스토그램



(c) 이중 임계값으로 분할한 영상

Adaptive thresholding

❖ 적응적 임계화

- 하나 또는 두 개의 임계값을 영상 전체에 적용하는 전역 방법의 한계
  - 지역적으로 명암 분포가 다른 경우 낮은 분할 품질
- 적응적 임계화로 해결: 지역에 따라 적응적으로 임계값 결정
  - Niblack 알고리즘 등

적응이치화(Niblack 알고리즘)

❖ 자동이치화(전역이치화)의 문제점

입력 영상

전역이치화

❖ 하나의 임계치(threshold)만으로 이치화를 하기 때문에 조명 등에 의해 밝기의 변화가 점진적으로 발생하는 경우에는 만족할만한 결과를 얻기 힘듬

Niblack 알고리즘

❖ 적응이치화

- 영상의 밝기 변화를 적용하는 방법

입력 영상

적응 이치화

$$T = m\left[1 - k \bullet \left(1 - \frac{\sigma}{R}\right)\right]$$

R : 표준편차의 최대범위(128)
k : 상수, 보통 0.02
m : 평균
δ : 표준편차

참고: 분산(variance) 구하기(다른방법)

$$\sigma^2 = \frac{\sum (x-\mu)^2}{N}$$: 편차의 제곱 평균

$$= \sum (x-\mu)^2 p(x)$$

$$= \sum (x^2 - 2x\mu + \mu^2) p(x)$$

$$= \sum x^2 p(x) - 2\mu \sum xp(x) + \mu^2 \sum p(x)$$

$$= \sum x^2 p(x) - 2\mu^2 + \mu^2$$

$$= \sum x^2 p(x) - \mu^2$$

$$= \frac{\sum x^2}{N} - \mu^2$$: 제곱의 평균 – 평균의 제곱

$\because \sum xp(x) = \mu$

$\because \sum p(x) = 1$

❖ Niblack 알고리즘 구현

```python
import cv2
import numpy as np
import math

def getMeanVariance(y, x):
    gsum, ssum, count = 0.0, 0.0, 0
    for k in range(y-w, y+w+1):
        if k < 0 or k >= ysize:
            continue
        for l in range(x-w, x+w+1):
            if l < 0 or l >= xsize:
                continue
            value = img.item(k, l)
            gsum += value
            ssum += value ** 2
            count += 1
    mean = gsum / count
    vari = (ssum / count) - mean ** 2
    if vari < 0.0: vari = 0.0
    return mean, vari
```

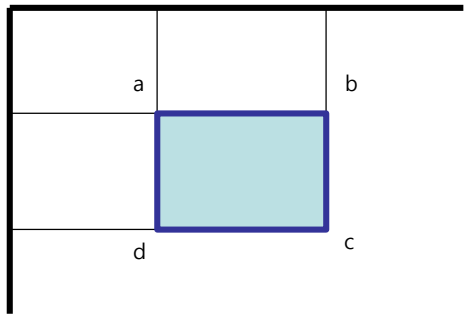
```python
filename = "VIN1.jpg"
img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
cv2.imshow('image', img)

w = 10  # window size
ysize = img.shape[0]
xsize = img.shape[1]
outImg = np.zeros((ysize, xsize), dtype=np.uint8)
for y in range(ysize):
    for x in range(xsize):
        m, v = getMeanVariance(y, x)
        th = m * (1 - 0.02 * (1 - math.sqrt(v)/128))
        if img.item(y, x) < th:
            outImg.itemset((y, x), 0)
        else:
            outImg.itemset((y, x), 255)

cv2.imshow('bin-image', outImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

참고: Integral Imaging

❖ 특정 영역의 면적을 효과적으로 계산하는 방법

(1) summed table 만들기

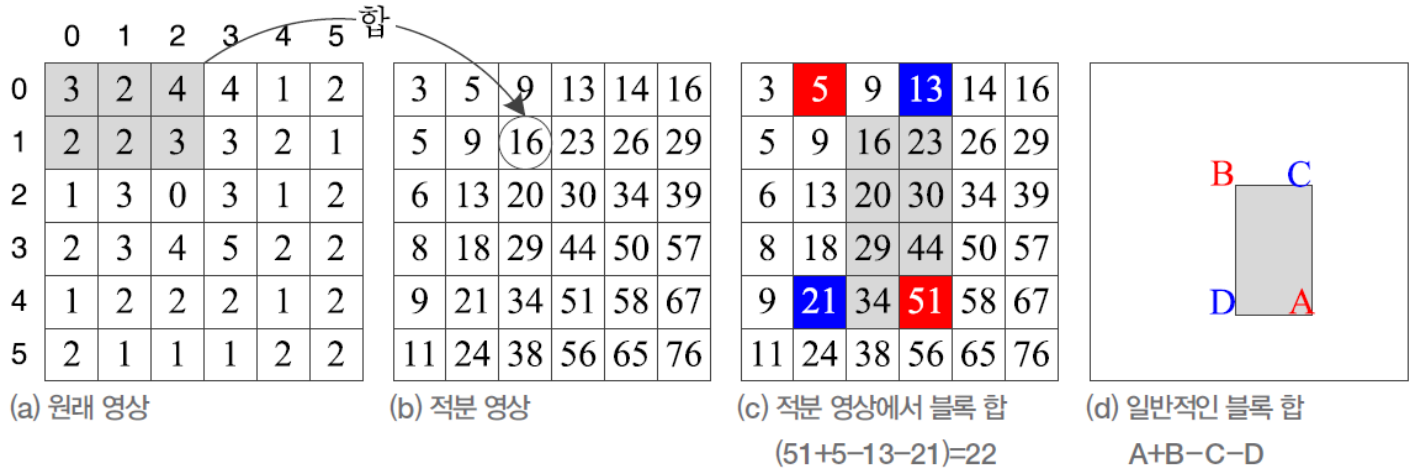
(2) 주어진 점들 사이의 면적 계산



$$Area(a;b;c;d) = S(c) - S(b) - S(d) + S(a)$$

$$S(i) = P(i_{x,y}) + S(i_{x-1,y}) + S(i_{x,y-1}) - S(i_{x,y})$$

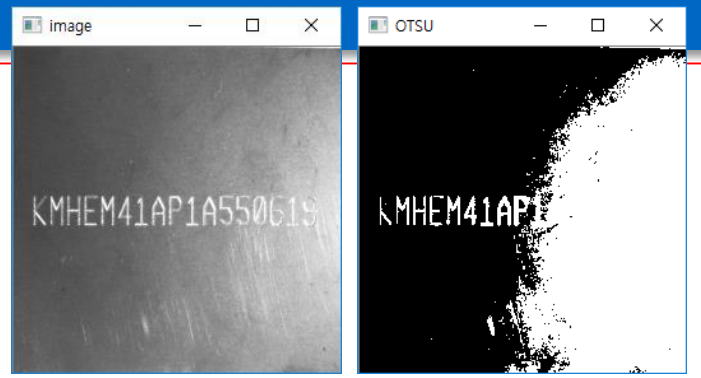
: 원점(0,0)부터 i(x,y)위치까지의 면적



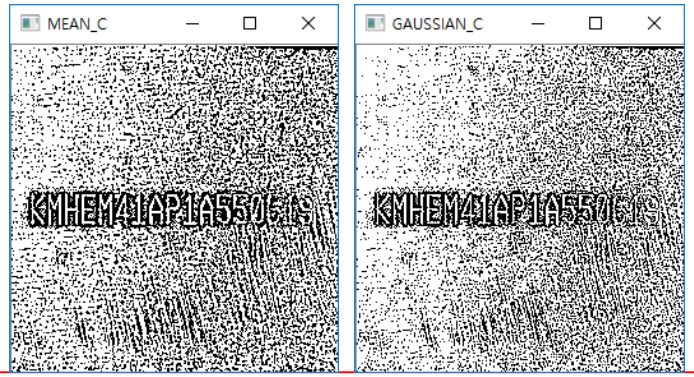
(a) 원래 영상 (b) 적분 영상 (c) 적분 영상에서 블록 합 (d) 일반적인 블록 합

(51+5−13−21)=22 A+B−C−D

cv2.adaptiveThreshold()




```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()    # filename = "VIN0.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    ret, result = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    result1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 5, 2)
    result2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 5, 2)

    cv2.imshow('image', img)
    cv2.imshow('OTSU', result)
    cv2.imshow('MEAN_C', result1)
    cv2.imshow('GAUSSIAN_C', result2)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```

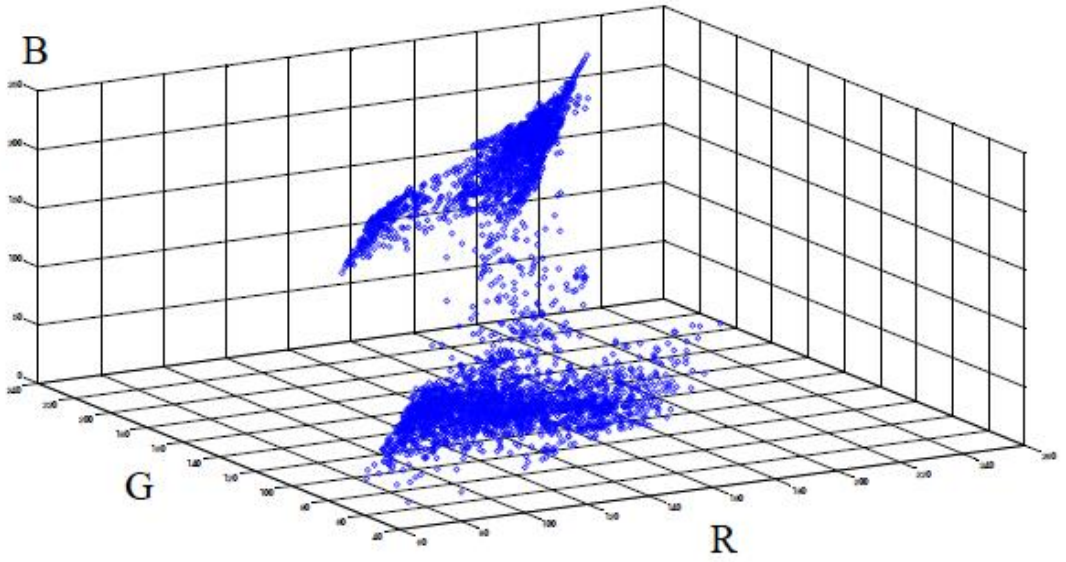


군집화를 이용한 영역분할

❖ 군집화

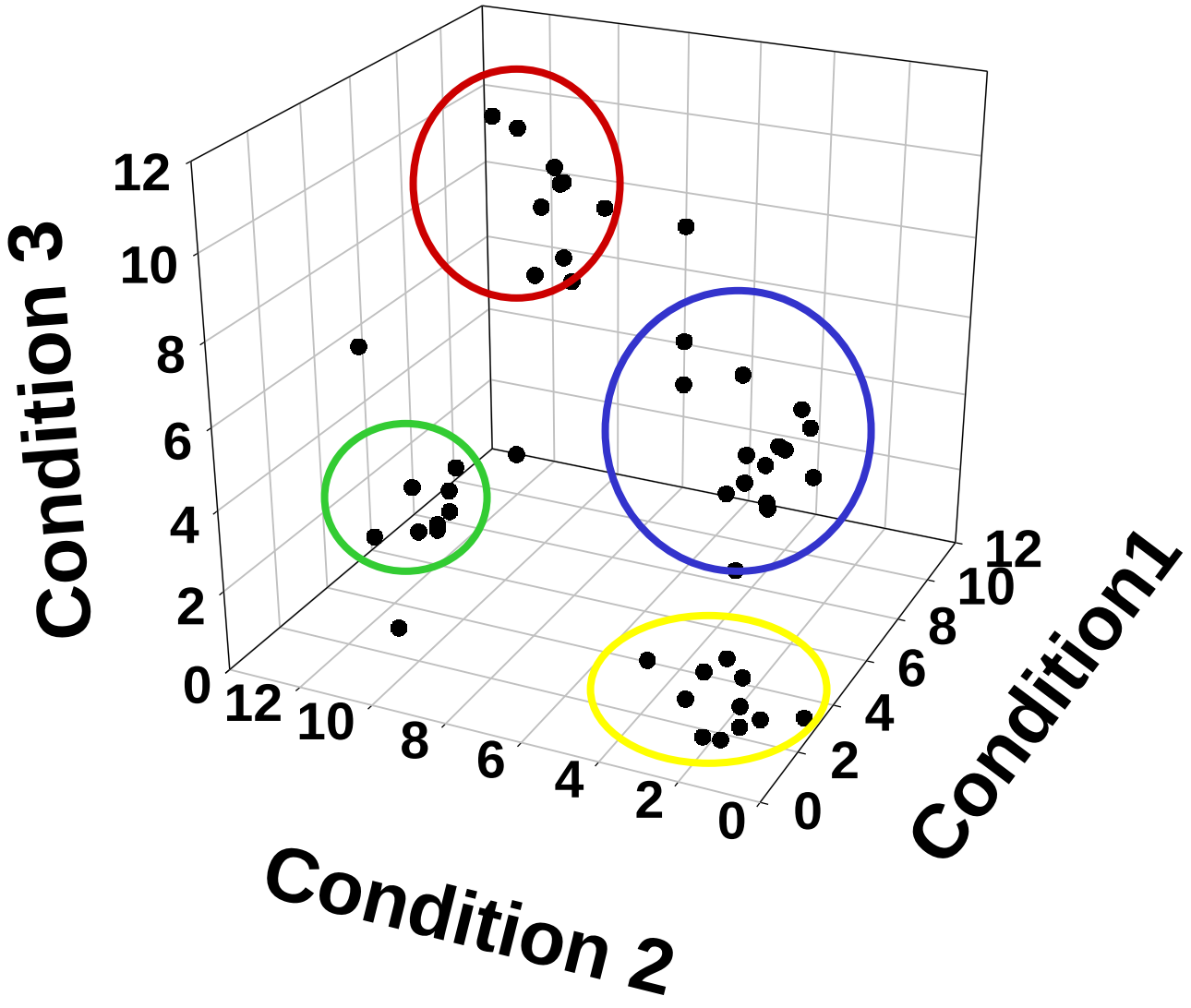
- 화소를 RGB 3차원 컬러 공간으로 매핑 한 후, k-means와 같은 알고리즘으로 군집화

(a) 원래 영상



(b) 3차원 공간에 매핑한 결과

Clustering


Condition 3
12
10
8
6
4
2
0
12
10
8
6
4
2
0
Condition 2
12
10
8
6
4
2
0
Condition1

K-means Algorithm

- For a given assignment C, compute the cluster means m_k:

$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k},\ k = 1, \ldots, K.$$

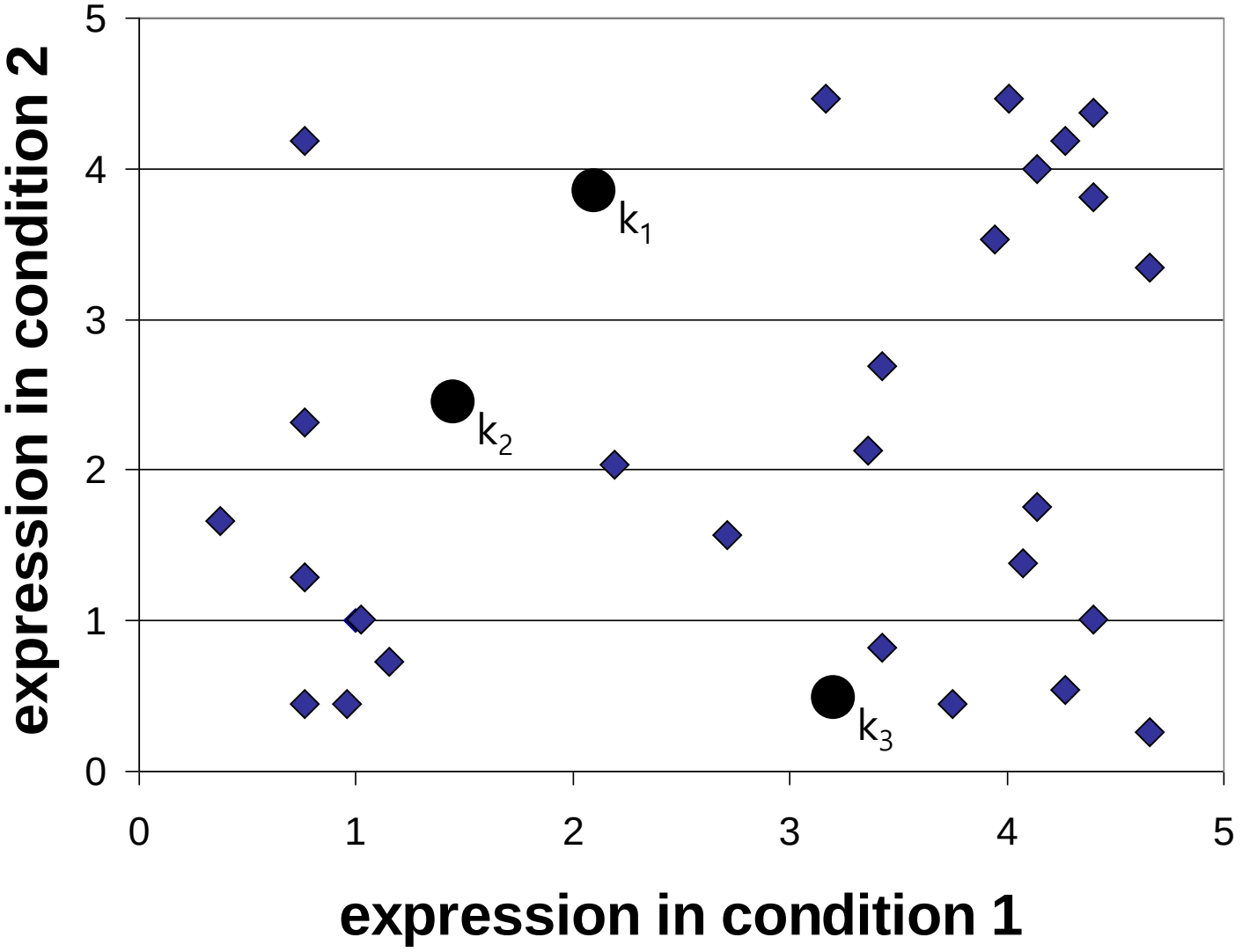
- For a current set of cluster means, assign each observation as:

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2,\ i = 1, \ldots, N$$

- Iterate above two steps until convergence

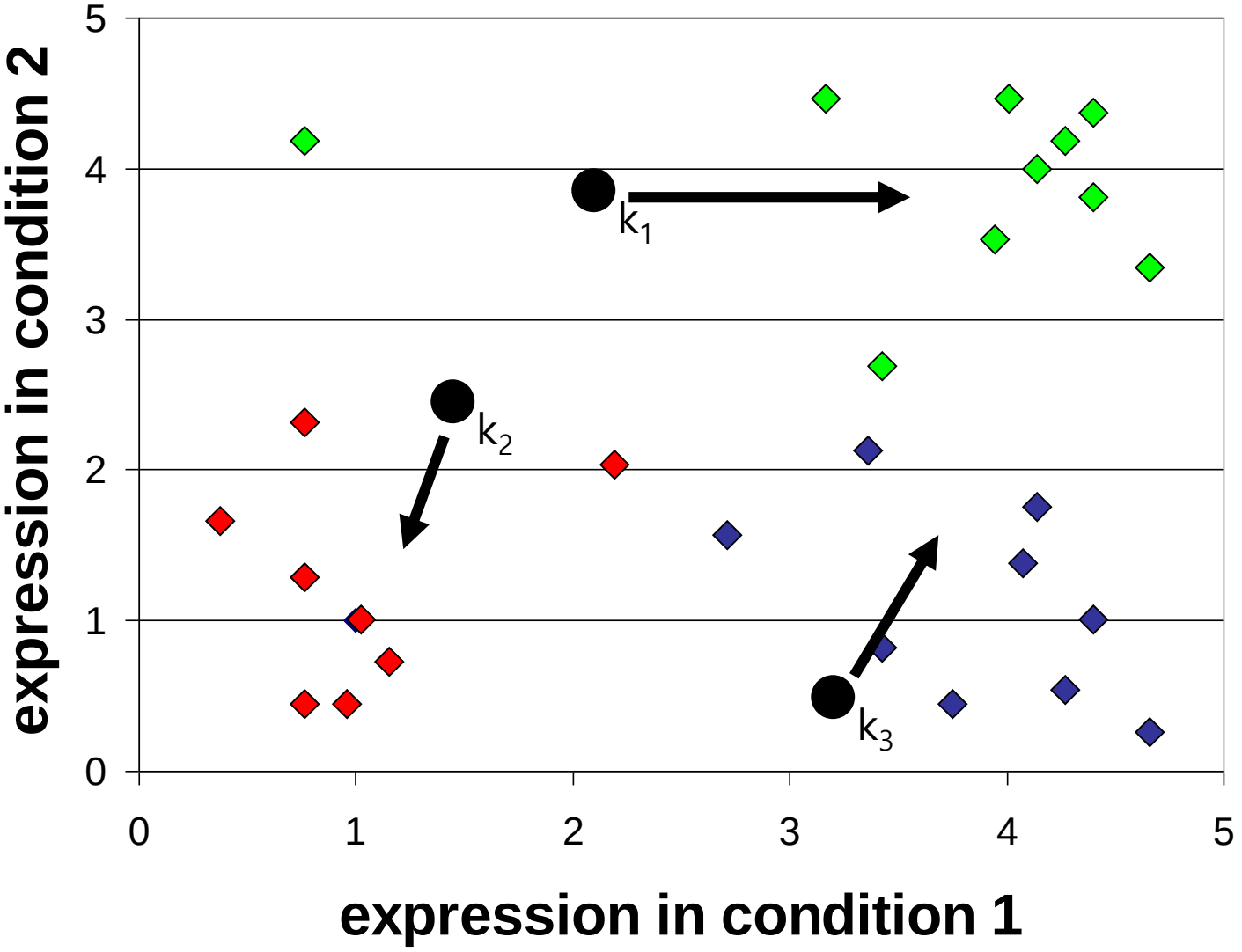
Clustering: Example, Step 1

Algorithm: k-means, Distance Metric: Euclidean Distance



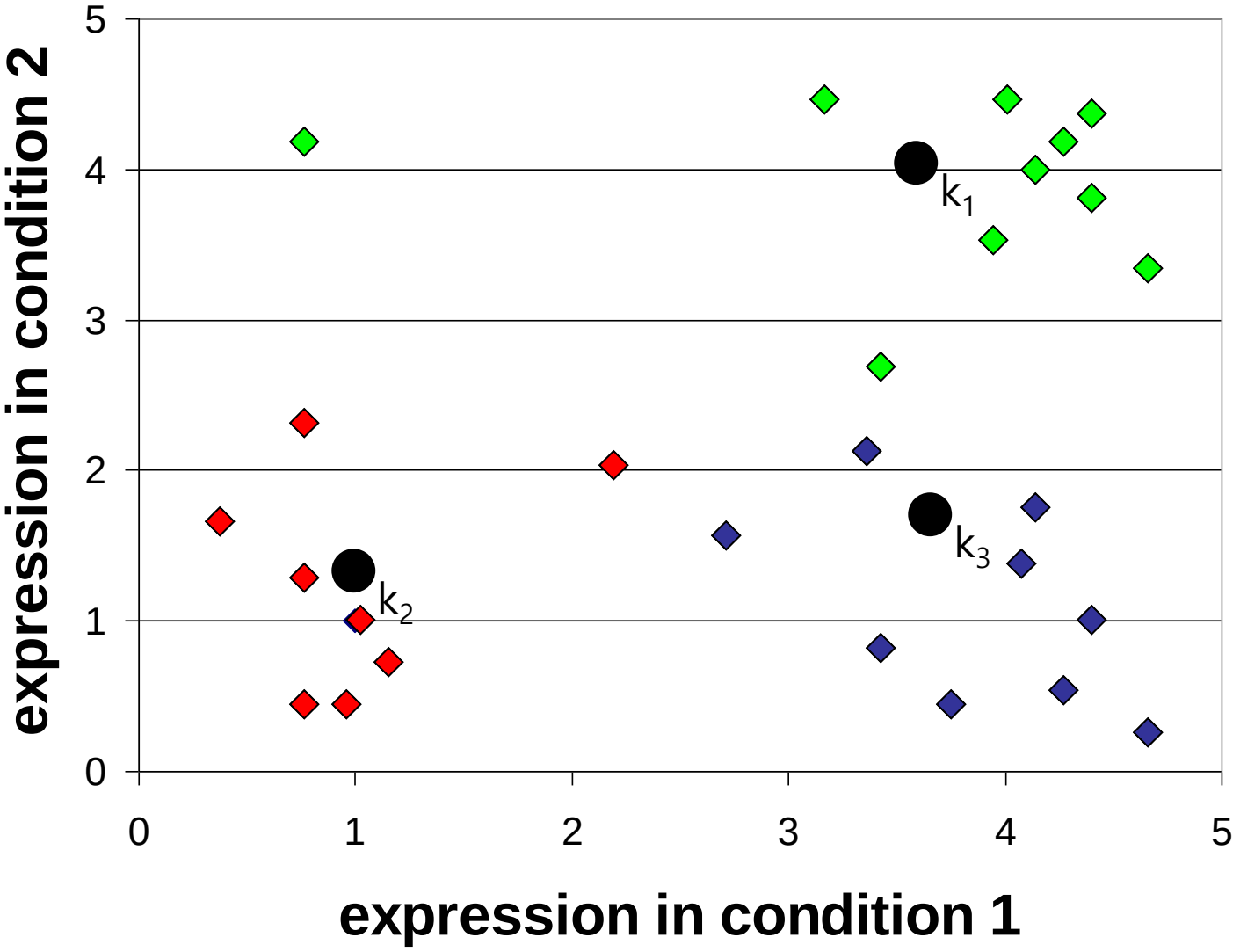
Clustering: Example, Step 2

Algorithm: k-means, Distance Metric: Euclidean Distance



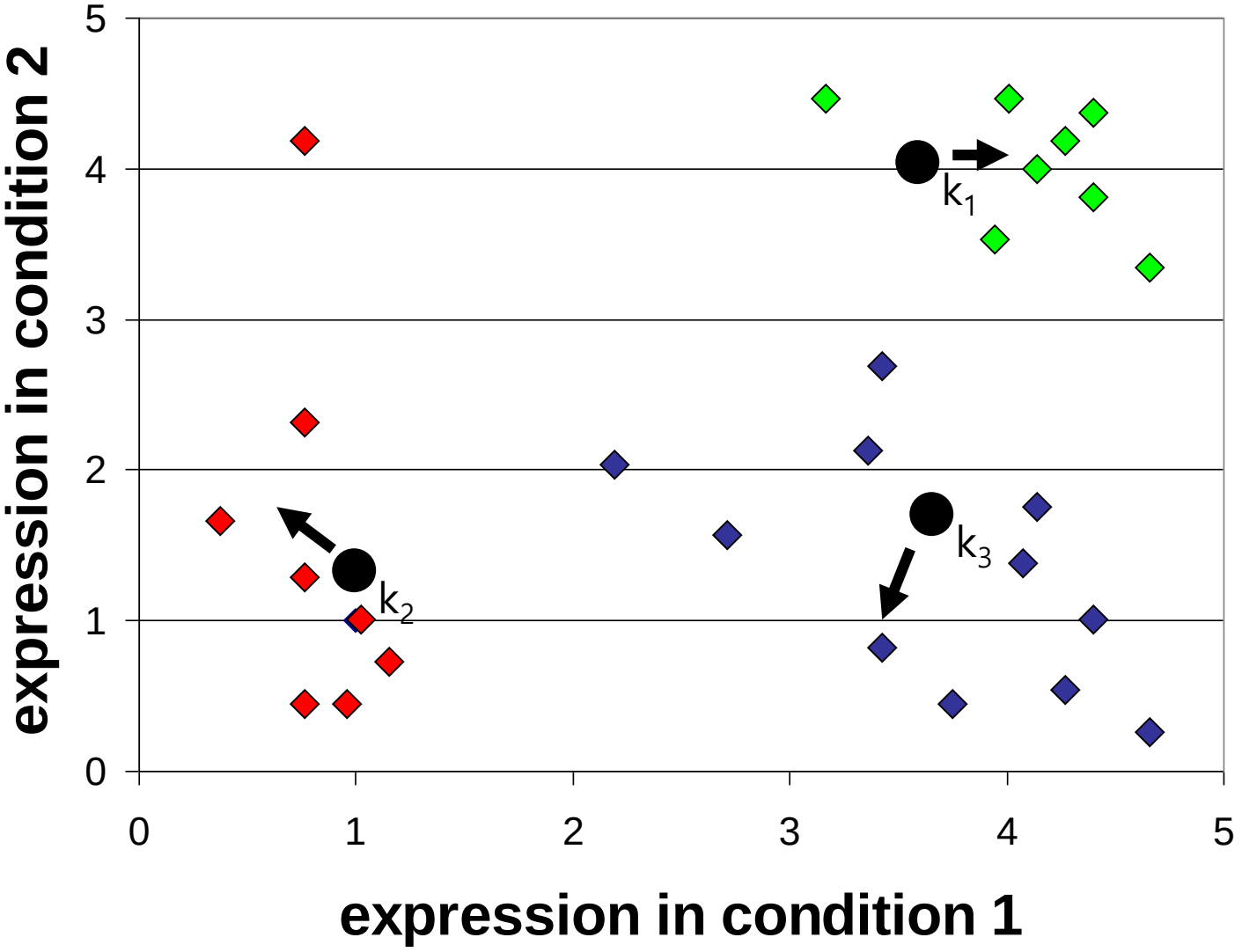
Clustering: Example, Step 3

Algorithm: k-means, Distance Metric: Euclidean Distance



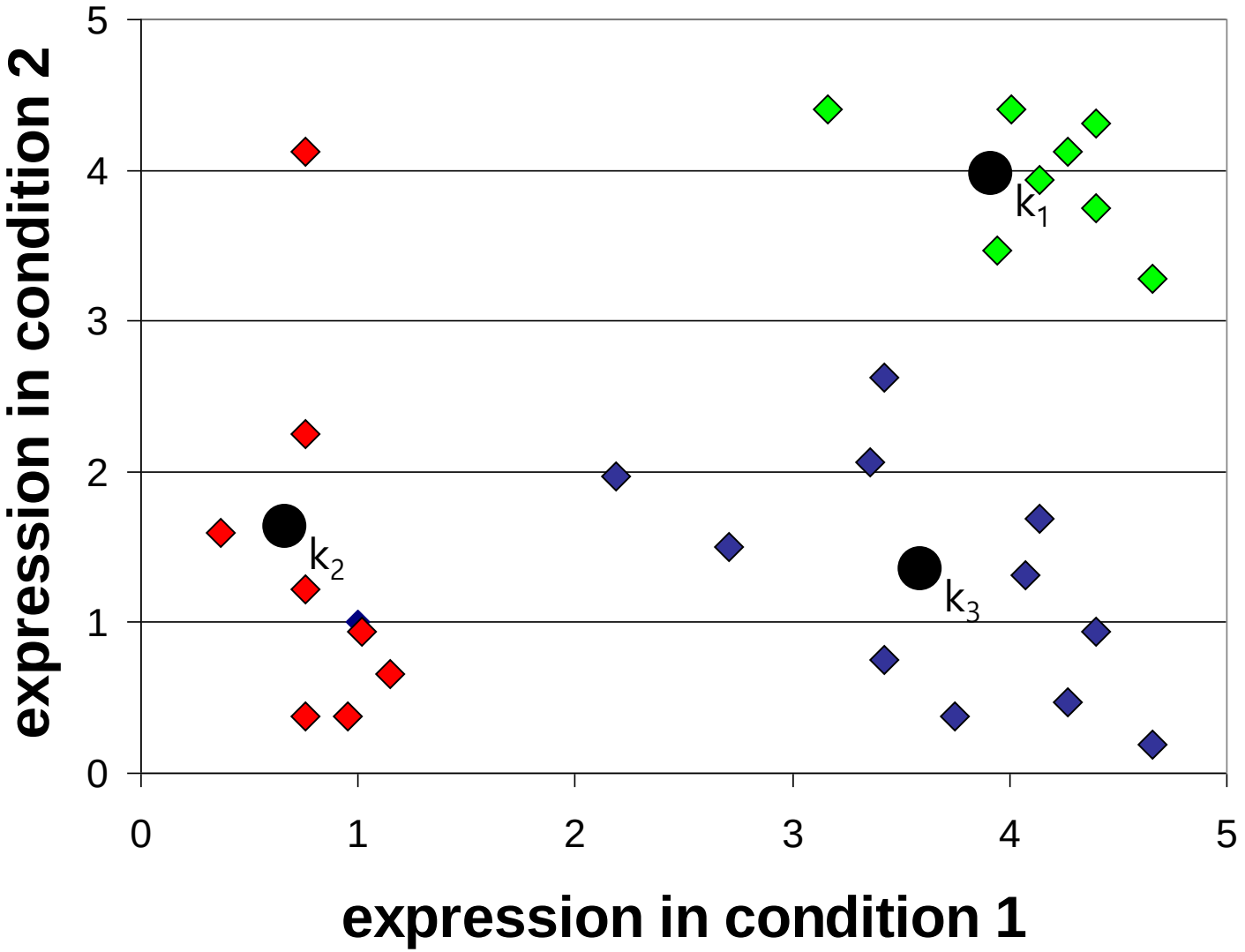
Clustering: Example, Step 4

Algorithm: k-means, Distance Metric: Euclidean Distance



Clustering: Example, Step 5

Algorithm: k-means, Distance Metric: Euclidean Distance



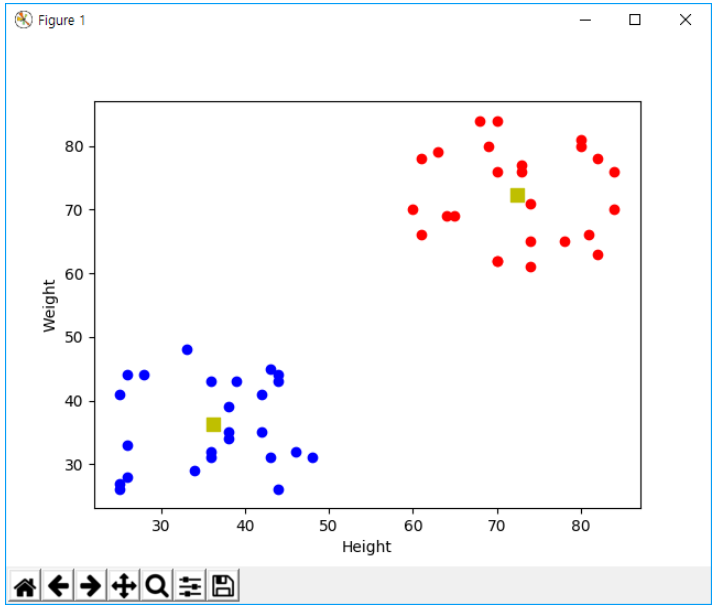
cv2.kmeans()

❖ **cv2.kmeans(data, K, bestLabels, criteria, attempts, flags, [centers=None])** -> retval, bestLabels, centers

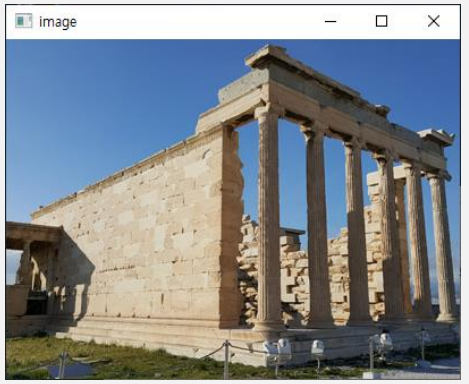
- **data** : 학습 데이터 행렬. shape=(N, d), dtype=np.float32
- **K** : 군집 개수
- **bestLabels** : 각 샘플의 군집번호 행렬. shape=(N, 1), dtype=np.int32.
- **criteria** : 종료 기준. (type, maxCount, epsilon) tuple
  - type : 보통 아래의 두 조건을 +하여 지정
    - » cv2.TERM_CRITERIA_EPS : 지정된 eps에 도달하면
    - » cv2.TERM_CRITERIA_MAX_ITER : 지정된 반복에 도달하면
- **attempts** : 다른 초기 레이블을 이용해 반복 시도할 횟수
- **flags** : 초기 centers 설정 방법.
  - cv2.KMEANS_RANDOM_CENTERS / cv2.KMEANS_PP_CENTERS / cv2.KMEANS_USE_INITIAL_LABELS
- **centers** : 군집 중심을 나타내는 행렬. np.ndarray. shape=(K, d), dtype=np.float32
- **retval** : Compactness measure

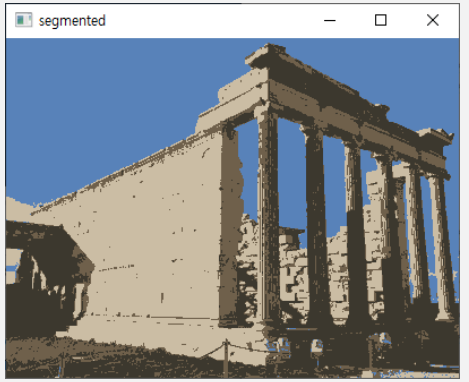
continue…

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt

# 25행 2열의 데이터를 각각 25~50, 60~85사이의 값을 갖도록 생성
X = np.random.randint(25, 50, (25,2))
Y = np.random.randint(60, 85, (25,2))
# 수직방향으로 쌓기
samples = np.vstack((X,Y))

# convert to np.float32
samples = np.float32(samples)

# define criteria and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret, label, center = cv2.kmeans(samples, 2, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# separate the data
A = samples[label.ravel()==0]
B = samples[label.ravel()==1]

# Plot the data
plt.scatter(A[:,0], A[:,1], c = 'b')
plt.scatter(B[:,0], B[:,1], c = 'r')
# marker의 size, color, marker type을 지정
plt.scatter(center[:,0], center[:,1], s = 80, c = 'y', marker = 's')
plt.xlabel('Height'),plt.ylabel('Weight')
plt.show()
```



```python
import numpy as np
import cv2

img = cv2.imread('read_color.jpg', cv2.IMREAD_COLOR)
cv2.imshow('image', img)

# shape을 3개의 열로 변환하고, 자료형을 np.float32로 변환
# (300, 400, 3) => (120000, 3)
src = img.reshape((-1, 3)).astype(np.float32)
# criteria와 cluster의 개수(K)를 설정
criteria = (cv2.TERM_CRITERIA_EPS|cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 4
# k-means 함수 적용
ret, label, center = cv2.kmeans(src, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
# 다시 자료형을 uint8로 변경
center = np.uint8(center)
dst = center[label.flatten()] # center 컬러로 모든 화소를 변경
dst = dst.reshape((img.shape)) # 원래의 이미지 shape으로 변경

cv2.imshow('segmented', dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```





cv2.inRange()

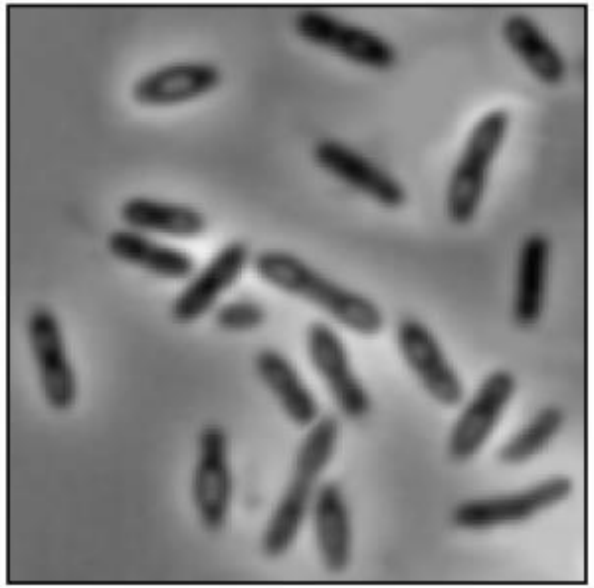
❖ 컬러범위에 의한 영역 분할

❖ cv2.inRange(src, lowerb, upperb, dst=None) -> dst

- src : 입력 영상
- lowerb : 하한 값 행렬 또는 스칼라
- upperb : 상한 값 행렬 또는 스칼라
- dst : 입력 영상과 같은 크기의 마스크 영상(np.uint8)
  - 해당 범위 안에 들어가는 픽셀은 255, 나머지는 0으로 설정

영역 라벨링(Region Labeling)

❖ 이 그림에 얼마나 많은 박테리아가 있을까?



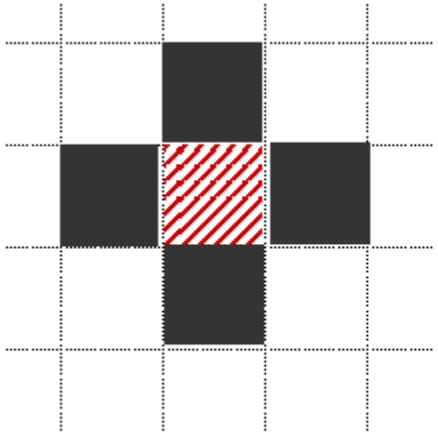
Original *Bacteria* image

after thresholding

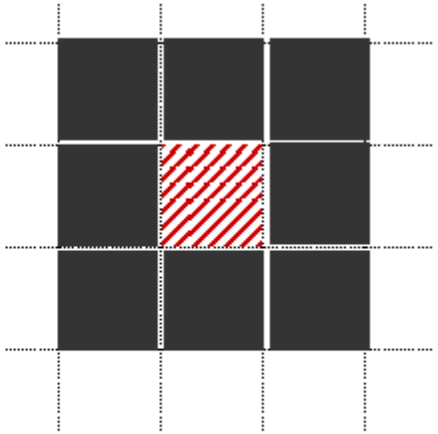
❖ 같은 obejct에 속한 pixel들은 같은 label을 갖도록 함

4 and 8-connected Neighborhoods

❖ 연결성 검사



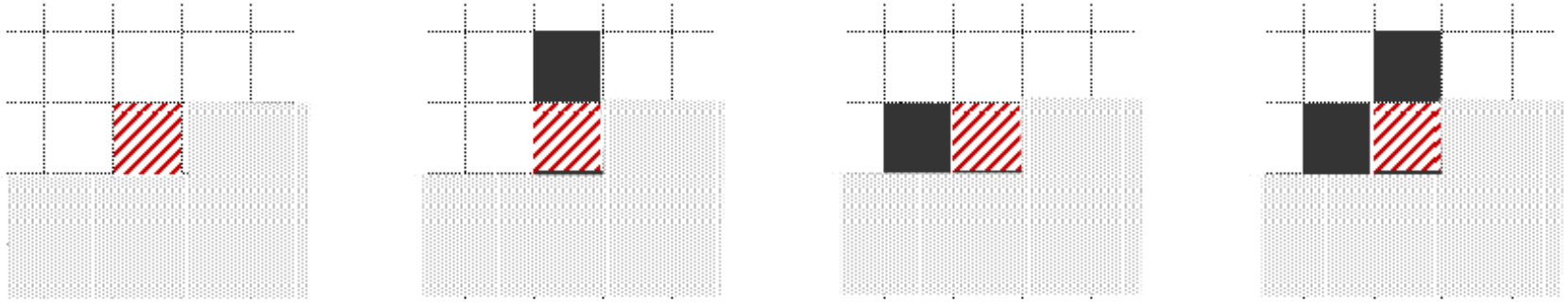
4-neighborhood



8-neighborhood

2-pass Region Labeling Algorithm

- **Loop through all pixels *f(x,y)*, left to right, top to bottom**
- **If *f(x,y)*=0, do nothing**
- **If *f(x,y)*=1, distinguish 4 cases**



Generate new region label

Copy label from above

Copy label from the left

Copy label from the left. If labels above and to the left are different, store equivalence.

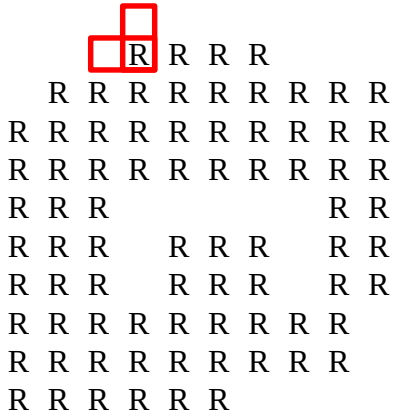
- **Second pass through image to replace equivalent label by the same label**

1-pass Region Labeling Algorithm

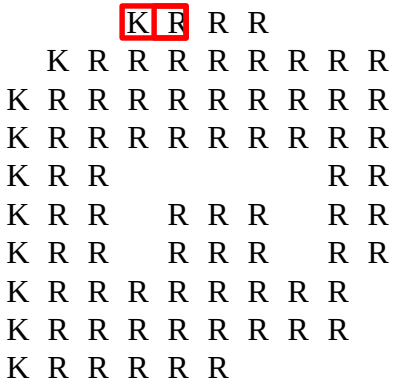
- **Each point of input image is initialized to one of the two value, ‘0’ or ‘R’**
  - background pixels are ‘0’
  - points to be labeled are set to ‘R’
    - a negative integer

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	R	R	R	R	0	0	0	0
0	0	R	R	R	R	R	R	R	R	R	0
0	R	R	R	R	R	R	R	R	R	R	0
0	R	R	R	R	R	R	R	R	R	R	0
0	R	R	R	0	0	0	0	0	R	R	0
0	R	R	R	0	R	R	R	0	R	R	0
0	R	R	R	0	R	R	R	0	R	R	0
0	R	R	R	R	R	R	R	R	R	0	0
0	R	R	R	R	R	R	R	R	R	0	0
0	R	R	R	R	R	R	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

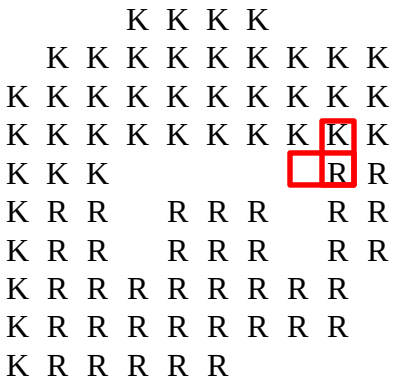
- **Algorithm**
  - Tracking of peripheral boundary region
    - assign a new label
    - find a next point of the boundary(clockwise direction) and then assign the same label
  - Propagation to points in the same horizontal run
  - Tracking the boundary of a hole
    - find a next point of the inside boundary(counterclockwise direction)
    - assign the same label



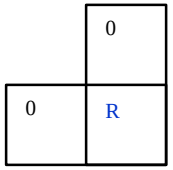
(a)



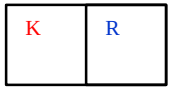
(b)



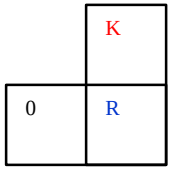
(c)



Peripheral boundary point



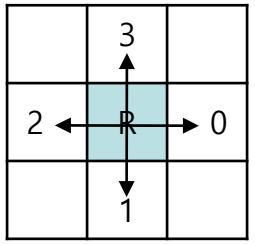
Propagation point



Hole boundary point

Boundary Tracking

- **Start direction code(D) set to '0'**
- **The next point of the boundary is found by searching clockwise**
  - **from** the point in the directions given by D-1
  - **until** a point with 'R' or 'K' is found.
    - R : not yet labeled
    - K : already labeled

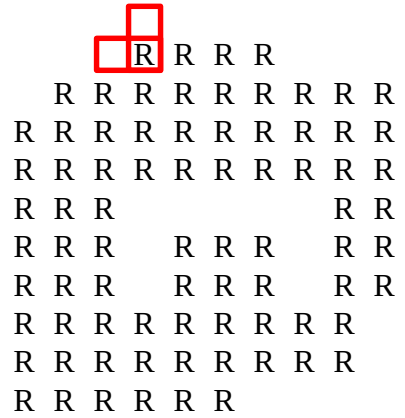


Direction Codes

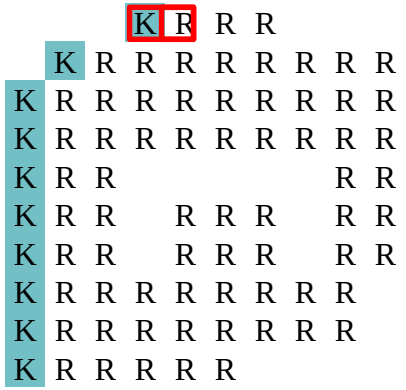
- **The decision to label is determined by the directions**
  - previous point → current point
  - current point → next point

next direction

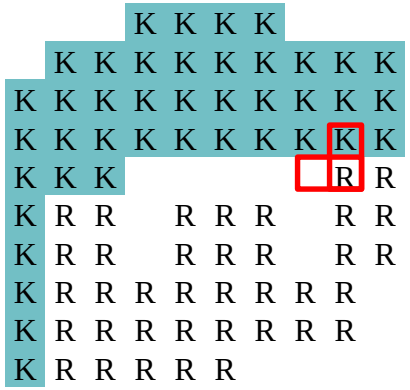
prev. direction	0	1	2	3
0				
1	X	X		O
2	O	X		O
3	O	O		O



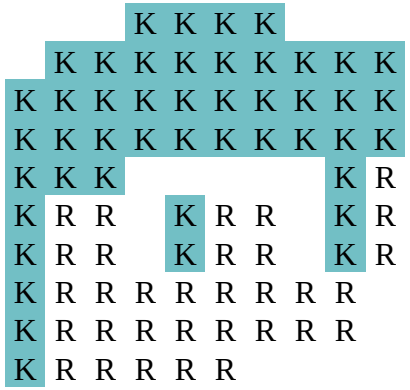
(a)



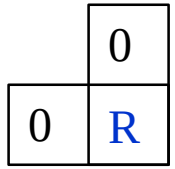
(b)



(c)

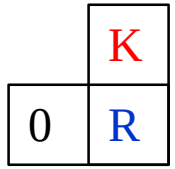


(d)

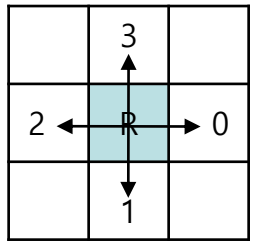


Peripheral boundary point

Propagation point



Hole boundary point

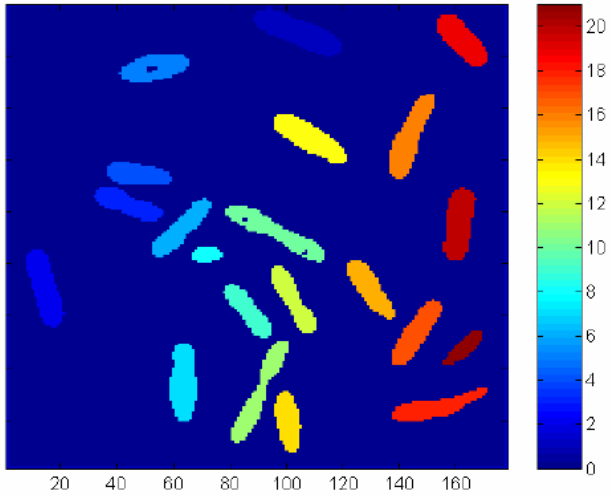


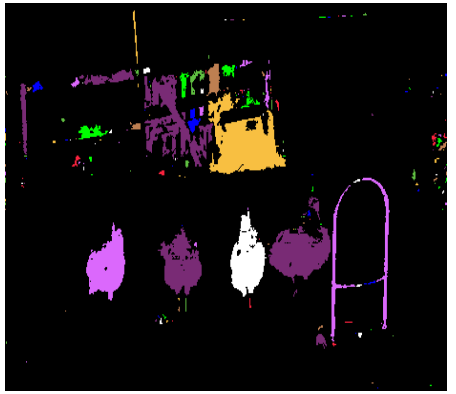
Direction Codes

next direction

prev. direction	0	1	2	3
0				
1	X	X		O
2	O	X		O
3	O	O		O

Example: region labeling





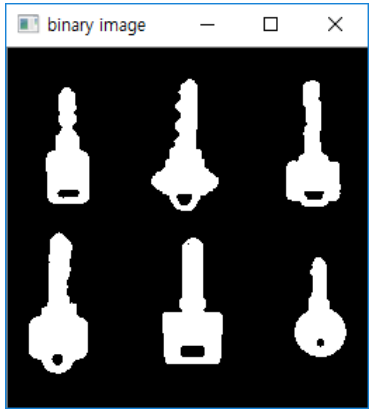
cv2.connectedComponents()

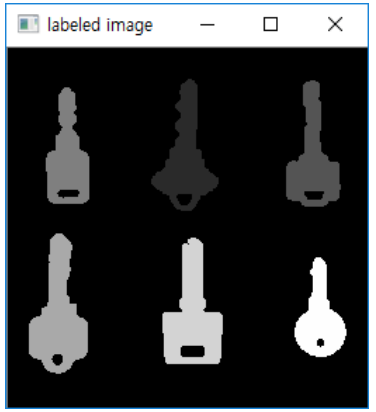
- **라벨링(labeling) : 서로 연결되어 있는 객체 픽셀에 고유한 번호를 지정하는 작업(일반적으로 이진 영상에서 수행)**
  - 배경이 0(검정색)이고 객체가 255(흰색)

- **cv2.connectedComponents(image, labels=None, connectivity=None, ltype=None)** -> retval, labels
  - image : 8비트 1채널 영상
  - labels : 레이블 맵 행렬. 입력 영상과 같은 크기. np.ndarray
  - connectivity : 4 또는 8 (기본값 : 8)
  - ltype : labels 타입. cv2.CV_32S 또는 cv2.CV_16S(기본값 : cv2.CV_32S)
  - retval : 객체 개수. N을 반환하면 [0, N-1]의 레이블이 존재
    - 0은 배경을 의미. (실제 흰색 객체 개수는 N-1개)

continue…

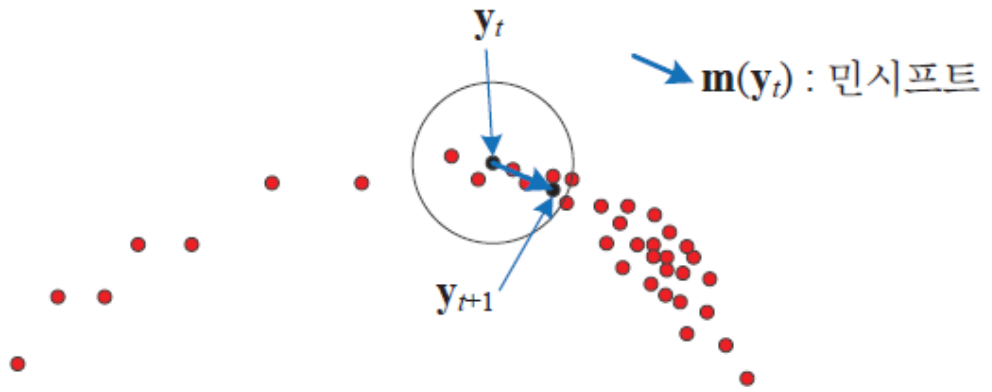
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Thresholding
    ret, threshold = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    cv2.imshow('binary image', threshold)
    # Opening
    kernel = np.ones((3,3), np.uint8)
    threshold = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel, iterations=1)
    # Labeling
    ret, markers = cv2.connectedComponents(threshold)
    cnt = np.amax(markers)
    print('number of labels = ', cnt)
    # display markers
    markers = markers * (254/cnt)
    markers = markers.astype(np.uint8)
    cv2.imshow('labeled image', markers)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```





Mean-Shift Algorithm

- **k의 값을 지정할 필요가 없으며, 우회적으로 소속(모드)를 결정하는 기법**
  - k-means 알고리즘과 유사
  - **kernel의 폭(h)만 지정**
  - **cluster의 개수 자동결정**
- **차원이 클 수록 기하급수적으로 메모리와 계산시간이 증가됨**

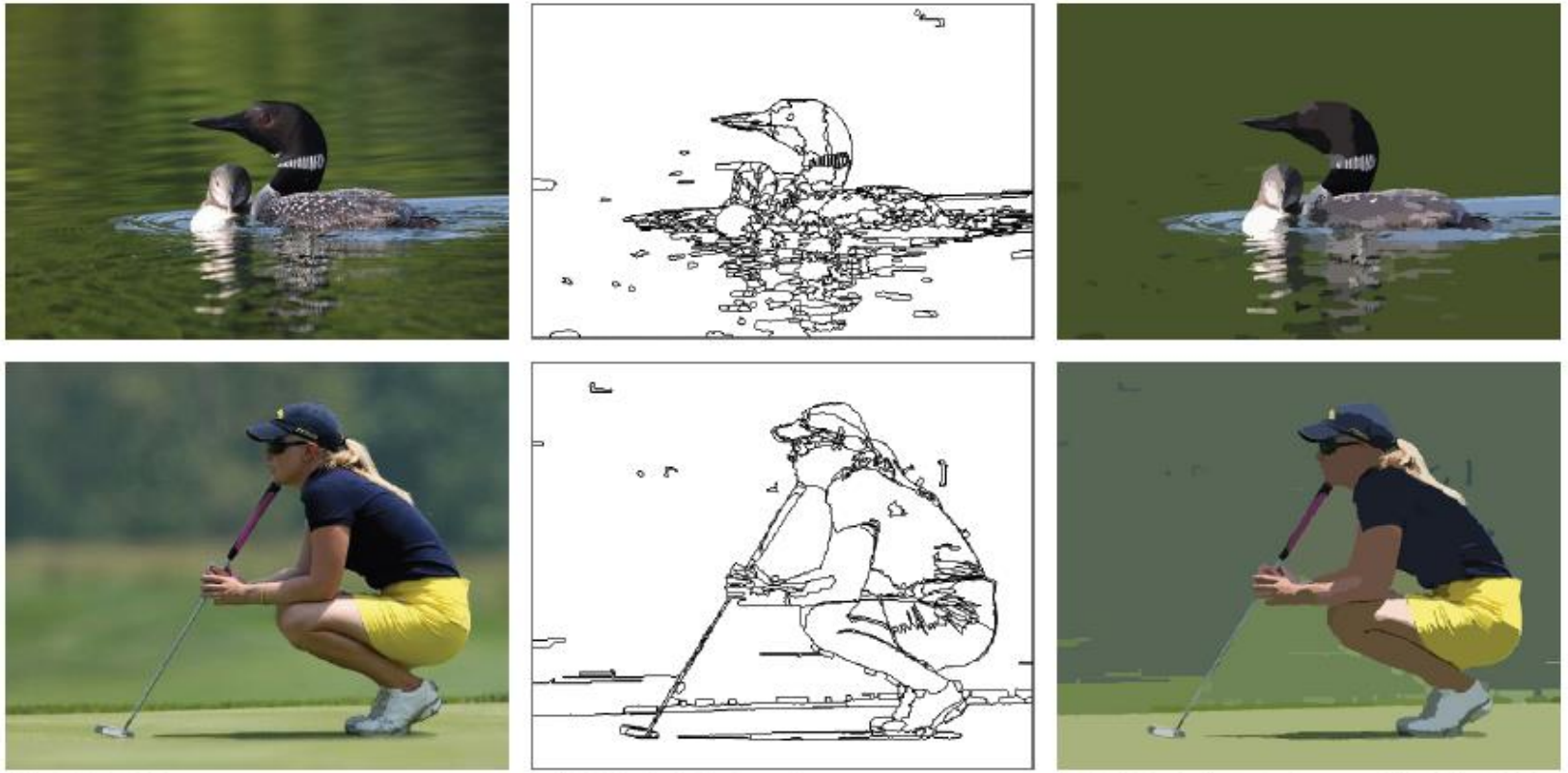


❖ mean-shift 알고리즘의 활용

- 주로 영상에서 미리 지정한 객체 추적을 위해 사용
  - 실제로는 Cam-shift를 많이 사용함
  - 왜냐하면, Cam-shift는 스케일에 무관하기 때문

- 영상분할에 사용
  - 일정한 kernel내의 수렴점을 구한 후,
  - 다른 수렴점을 모두 구하여 이들이 속한 군집을 할당
  - 컬러만 표현하면 *k*-means와 비슷한 결과
    - 컬러 $\mathbf{x}^r=(r,g,b)$와 화소의 위치 $\mathbf{x}^s=(y,x)$를 같이 표현 → 5차원 벡터 $\mathbf{x}=(r,g,b,y,x)$

❖ mean-shift로 분할한 영상의 예



(a) 원래 영상 (b) 영역의 경계를 표시 (c) 영역의 평균 컬러로 표시된 분할 영상

cv2.pyrMeanShiftFiltering()

❖ 피라미드 기반 영상분할을 수행

- 공간윈도우와 컬러윈도우를 이용하여 반복적으로 meanShift를 수행

❖ cv2.pyrMeanShiftFiltering(src,

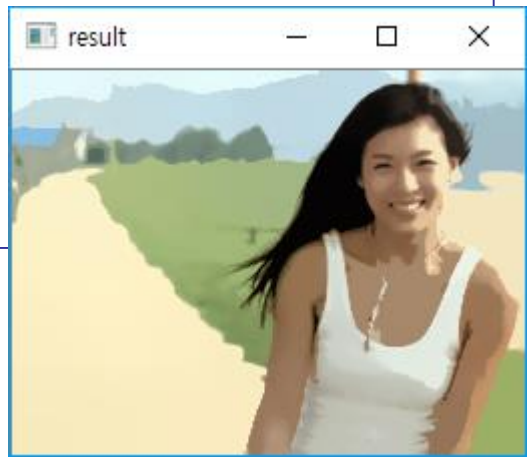
sp, sr, dst, maxLevel, termcrit) -> dst

- **src** – source 8-bit, 3-channel image.
- **sp** – spatial window radius.
- **sr** – color window radius.
- **dst** – destination image
- **maxLevel** – maximum level of the pyramid
- **termcrit** – termination criteria

continue…

❖ pyrMeanShiftFiltering를 이용한 영상분할의 예

```
import cv2
from tkinter.filedialog import askopenfilename

def main():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_COLOR)
    cv2.imshow('input', img)

    img = cv2.pyrMeanShiftFiltering(img, 20, 30)

    cv2.imshow('result', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

main()
```



MeanShift/CamShift 물체추적

❖ MeanShift

- 추적하고자 하는 물체의 히스토그램 역투영과 초기 탐색윈도우를 이용하여 물체의 중심을 반복적으로 탐색

❖ CamShift

- 물체의 중심, 크기, 방향을 반복적으로 탐색
- 먼저 MeanShift로 물체의 중심을 찾고, 물체의 크기와 방향을 계산
- 크기가 변하는 물체도 추적 가능
- 알고리즘

(1) 민시프트 알고리즘으로 이동 위치 계산

(2) 민시프트가 수렴한 위치에서 검색 윈도우의 윈도우 크기를 약간(약 10픽셀)씩 키움

(3) 키운 윈도우 안에서 객체의 위치 탐색

(4) 특징 공간을 가장 잘 표현하는 타원을 만들어서 타원의 크기만큼 윈도우 크기를 키움

(5) 객체의 크기에 맞게 타원의 크기 조정

(6) 새로운 크기의 윈도우를 이용하여 다시 민시프트 수행

cv2.CamShift()

❖ **cv2.CamShift(probImage, window, criteria)** -> retval, window

- **probImage** : 관심 객체에 대한 히스토그램 역투영 영상(확률 영상)
- **window** : 초기 검색 영역 윈도우(결과 영역)
- **criteria** : 알고리즘 종료 기준
  - (type, maxCount, epsilon) 튜플
  - (예) term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)

    ➔ 최대 10번 반복하며, 정확도가 1이하이면 (즉, 이동 크기가 1픽셀보다 작으면) 종료
- **retval** : 추적하는 객체의 모양을 나타내는 회전된 사각형 정보 반환
  - ((cx, cy), (width, height), angle) 튜플

continue…

❖ CamShift를 이용한 물체추적의 예

```
import numpy as np
import cv2

cap = cv2.VideoCapture('slow.flv')

# take first frame of the video
ret,frame = cap.read()

# setup initial location of window
r,h,c,w = 250,90,400,125  # simply hardcoded the values
track_window = (c,r,w,h)

# set up the ROI for tracking
roi = frame[r:r+h, c:c+w]
hsv_roi =  cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)),
                                    np.array((180.,255.,255.)))
roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)

# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
```

```
while(1):
    ret ,frame = cap.read()
    if ret == True:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)

        # apply meanshift to get the new location
        ret, track_window = cv2.CamShift(dst, track_window, term_crit)

        # Draw it on image
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv2.polylines(frame,[pts],True, 255,2)
        cv2.imshow('img2',img2)
        k = cv2.waitKey(60) & 0xff
        if k == 27:
            break
        else:
            cv2.imwrite(chr(k)+".jpg",img2)
    else:
        break

cv2.destroyAllWindows()
cap.release()
```