영상 분할 및 특징 처리

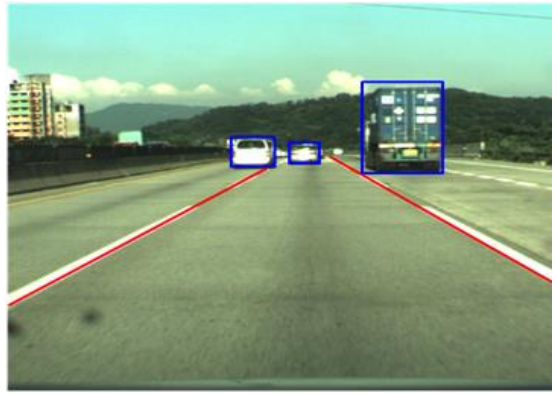
담당교수 : 김민기

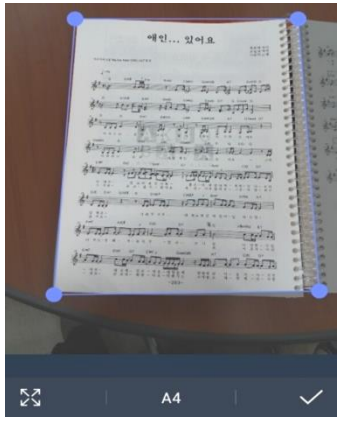
목 차

1. 허프 변환 (Hough Transform)

❖직선 검출
- 영상 내에서 공간 구조를 분석하는데 유용한 도구
- 영상 처리와 컴퓨터 비전분야에서 많은 연구 진행

❖다양한 응용에 사용
- 차선 및 장애물 자동인식 시스템: 차선 검출
- 스캐너의 기능을 대신해 주는 앱: 네 개 모서리 검출





Hough Transform

❖허프변환 좌표계

- 직교 좌표계로 표현되는 영상의 에지 점들을 극 좌표계로 옮겨, 검출하고자 하는 물체의 파라미터(ρ, θ)를 추출하는 방법

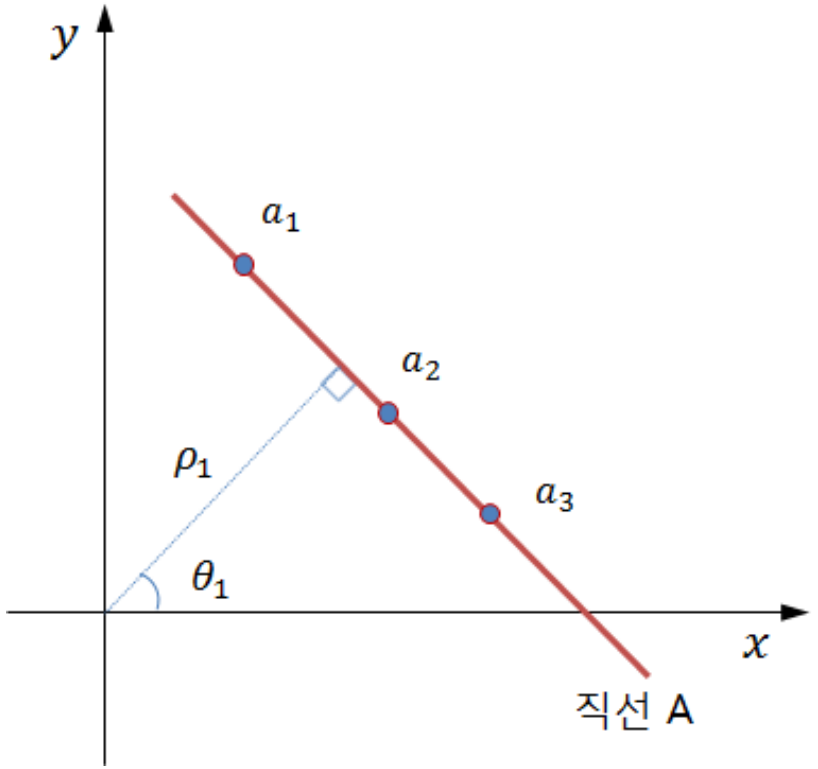
$$y = ax + b \quad \leftrightarrow \quad \rho = x \cdot \cos\theta + y \cdot \sin\theta$$

- 직교좌표의 직선은 허프변환 좌표에서 한점 (ρ_1, θ_1)으로 표현
- 직교좌표의 한점은 허프변환 좌표에서 곡선으로 표현

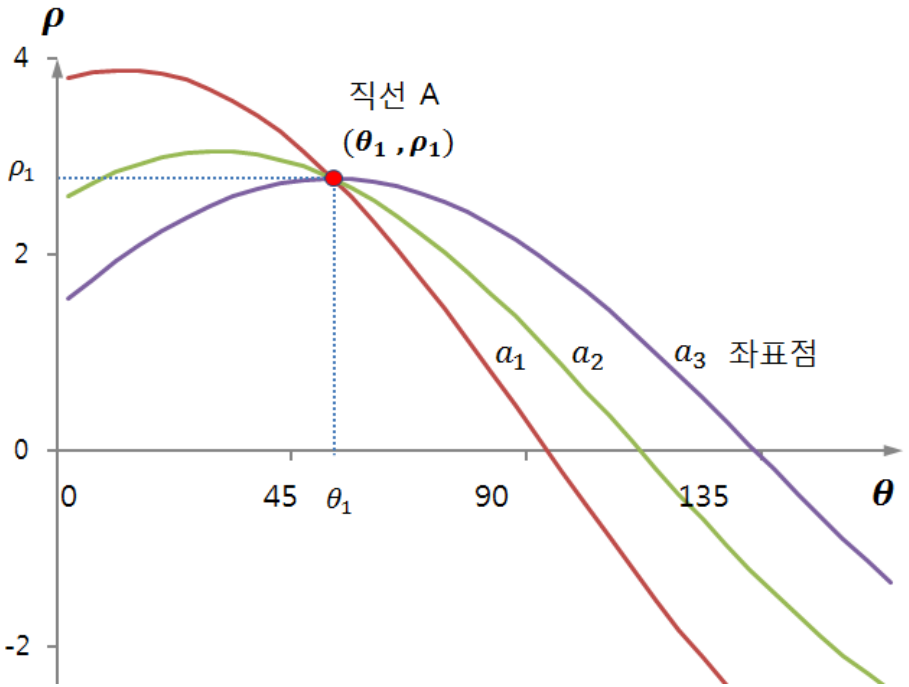
Hough Transform

❖허프 변환 좌표계

- 직교좌표의 직선은 허프변환 좌표에서 한점 (ρ_1, θ_1)으로 표현
- 직교좌표의 한점은 허프변환 좌표에서 곡선으로 표현

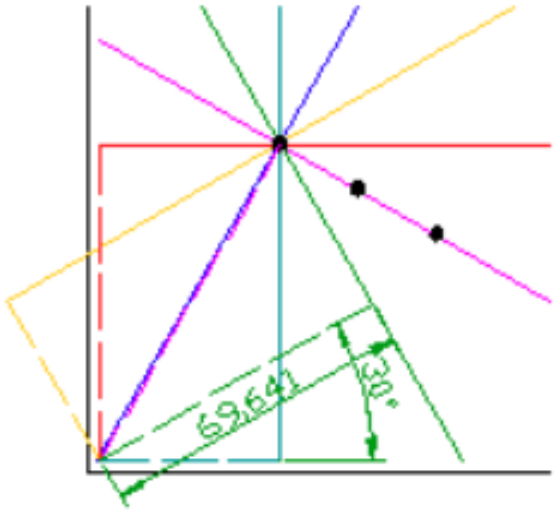


$$y = ax + b \quad \leftrightarrow \quad \rho = x \cdot \cos\theta + y \cdot \sin\theta$$

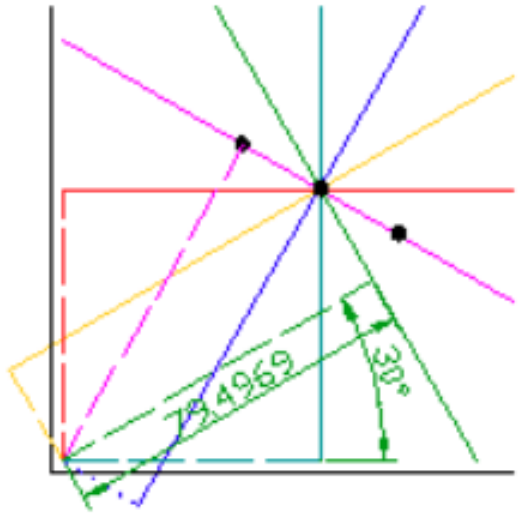


Example 1 [edit]

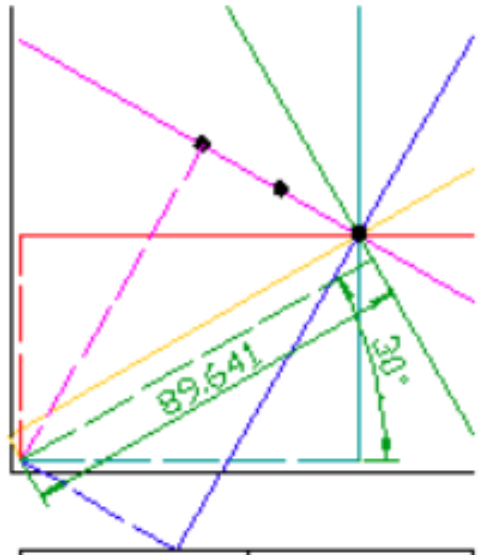
Consider three data points, shown here as black dots.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

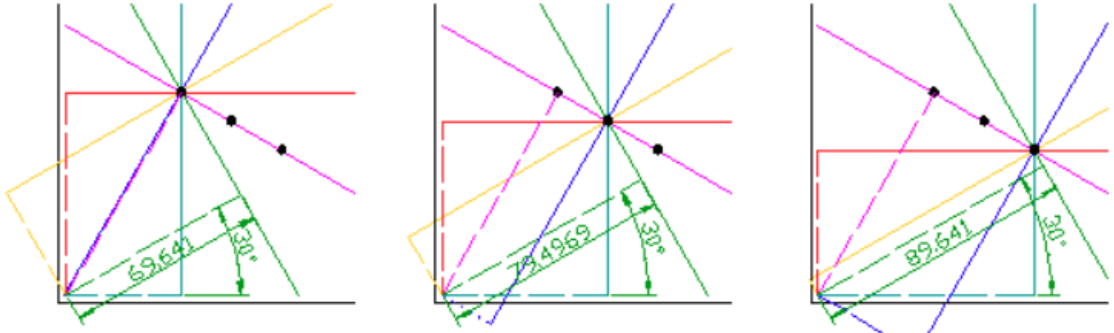


Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6

출처: https://en.wikipedia.org/wiki/Hough_transform

Example 1 [edit]

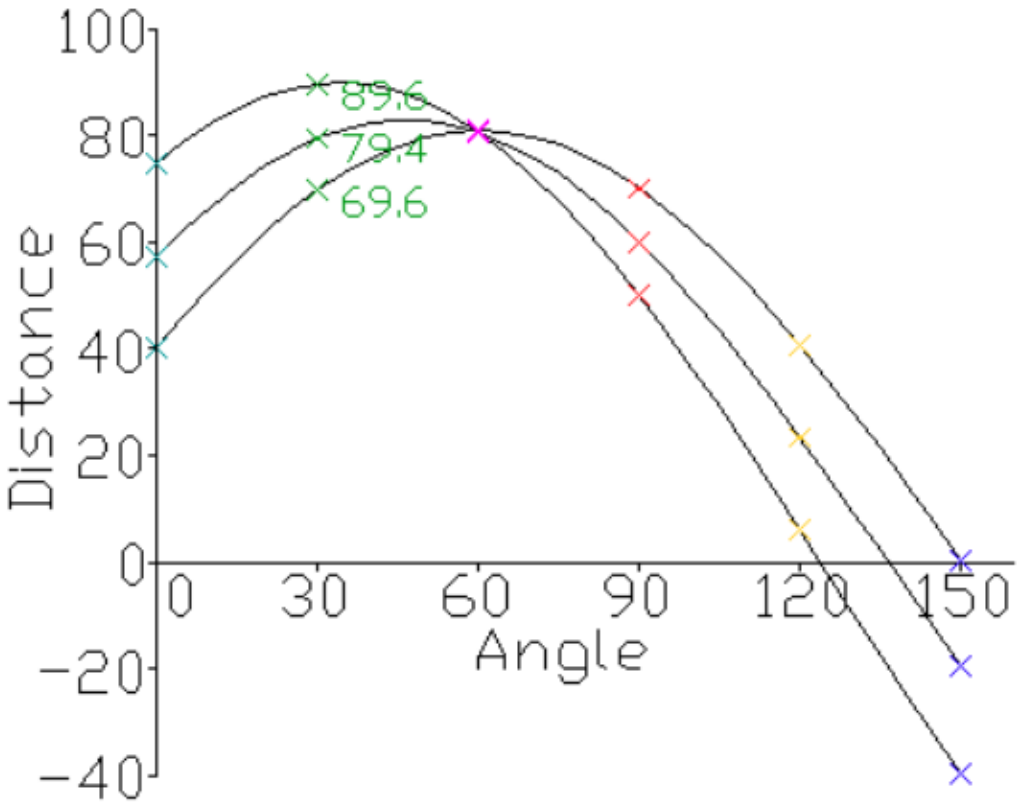
Consider three data points, shown here as black dots.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	−19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	−39.6



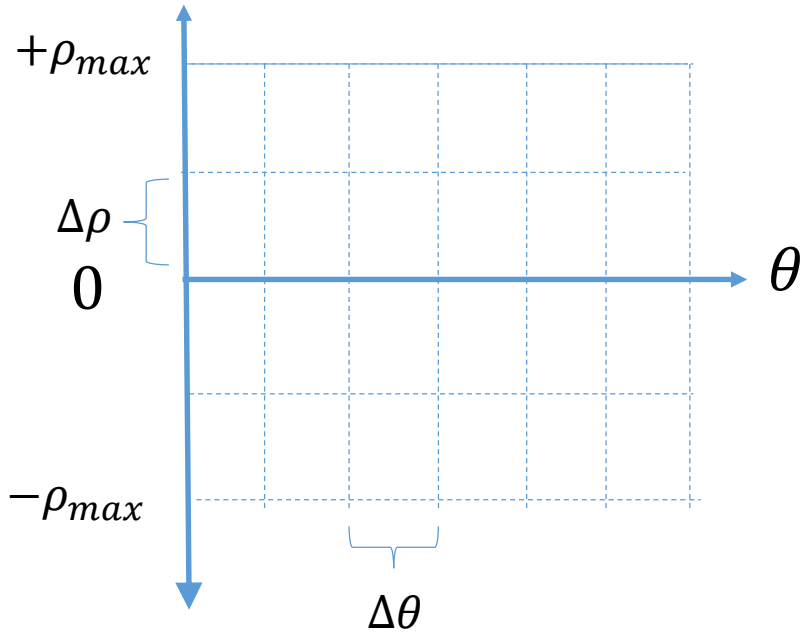
출처: https://en.wikipedia.org/wiki/Hough_transform

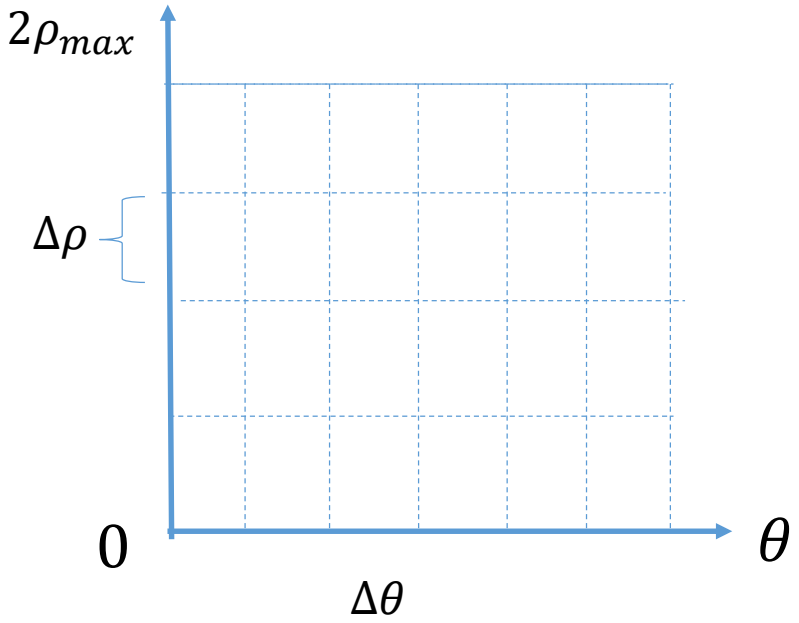
Hough Transform

❖허프 변환 좌표계를 위한 행렬

$$-\rho_{\max} \le \rho \le \rho_{\max}, \quad \rho_{\max} = height + width, \quad acc\_h = \frac{\rho_{\max}{}^{*2}}{\Delta\rho}$$

$$0 \le \theta \le \theta_{\max}, \quad \theta_{\max} = \pi, \quad acc\_w = \frac{\pi}{\Delta\theta}$$





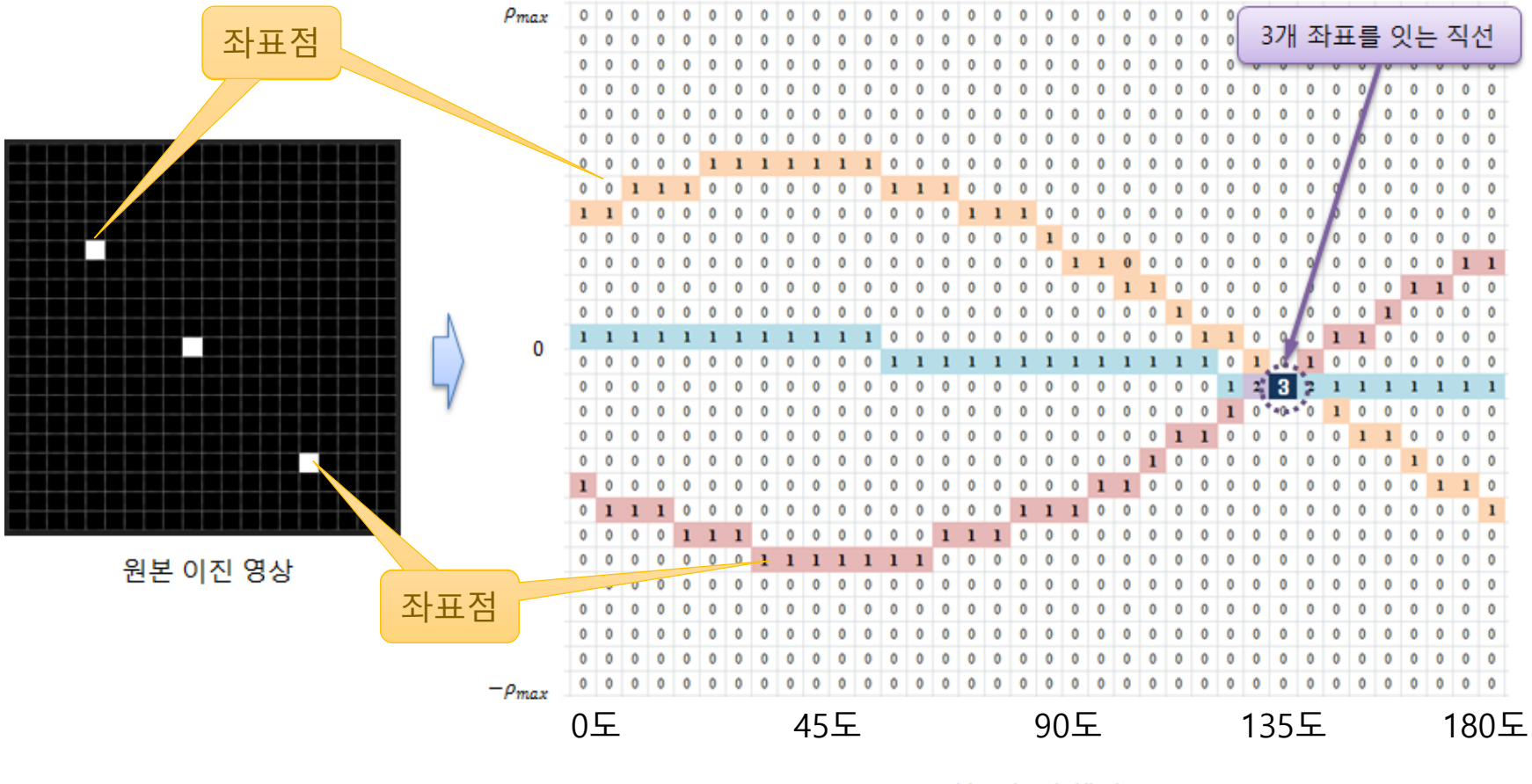
Hough Transform

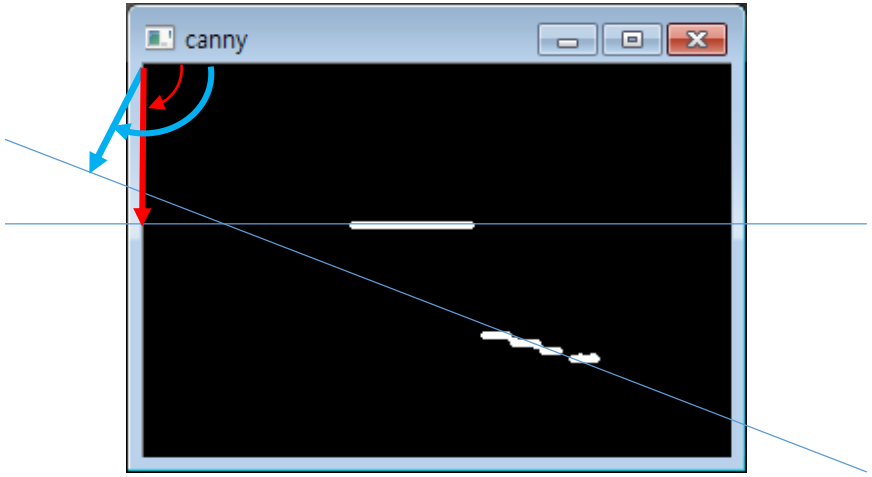
❖Line detection algorithm using Hough transform

(입력영상) $M \times N$

$$E(x,y) = \begin{cases} \neq 0, & \text{if } (x,y) \text{ is an edge pixel.} \\ = 0, & \text{otherwise} \end{cases}$$

1. 허프 변환 좌표계에서 행렬 구성
2. 영상 내 모든 화소의 에지 여부 검사
3. 에지 인지 좌표에 대한 허프 변환 누적 행렬 구성
4. 허프 누적 행렬의 지역 최대값 선정
5. 임계값 이상인 누적값(직선) 선별
6. 직선(ρ_i, θ_j)을 누적값 기준으로 내림차순 정렬


좌표점
원본 이진 영상
좌표점
ρ_{max}
0
$-\rho_{max}$
3개 좌표를 잇는 직선
0도
45도
90도
135도
180도



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	4	5	4	4	4	3	4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	6	5	4	4	4	3	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	6	5	5	5	5	3	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	7	7	7	6	4	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	10	8	6	3	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	6	14	10	5	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	6	17	11	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	11	17	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
0	0	0	0	1	5	12	17	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	6
0	2	4	7	11	12	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	4	0
6	7	8	10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	2	0	0
7	7	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	11	1	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	9	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	8	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8	6	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	8	3	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	8	2	0	0	0	0	0	0	0	0	0	0


마스크 적용 예시 1
마스크 적용 예시 2

```
25  image = cv2.imread("images/hough.jpg", cv2.IMREAD_GRAYSCALE)
26  if image is None: raise Exception("영상파일 읽기 에러")
27  blur  = cv2.GaussianBlur(image, (5, 5), 2, 2)                    # 가우시안 블러링
28  canny = cv2.Canny(blur, 100, 200, 5)                             # 캐니 에지 추출
29
30  rho, theta = 1,  np.pi / 180                                # 수직거리 간격, 각도 간격
31  lines1 = houghLines(canny, rho, theta, 80)                       # 저자 구현 함수
32  lines2 = cv2.HoughLines(canny, rho, theta, 80)                   # OpenCV 함수
33  dst1 = draw_houghLines(canny, lines1, 7)                         # 직선 그리기
34  dst2 = draw_houghLines(canny, lines2, 7)
35
36  cv2.imshow("image", image)
37  cv2.imshow("canny", canny);
38  cv2.imshow("detected lines", dst1)
39  cv2.imshow("detected lines_OpenCV", dst2)
40  cv2.waitKey(0)
```


image


canny


detected lines_OpenCV


detected lines

Line Detection using Hough Transform

```
ndarray[1)]  HoughLines (
        ndarray image,     // 8bits 1채널 이진 영상
        double   rho,          // 거리 간격 (원점으로부터 거리)
        double theta,          // 라디안 간격 (x축과의 각도)
        int   threshold,       // 직선 검출을 위한 임계값
        double   srn=0,   double   stn=0[2)]
)
```

1) ndarray lines // CV_32FC2 행렬로 (ρ, θ)[1)]저장
ρ는 영상의 좌상을 원점으로 하는 거리이고, θ는 회전각도

2) srn=0, stn=0 이면 표준 Hough 변환이 사용되고,
그렇지 않으면 coarse-fine 다중 스케일 Hough 변환이 사용됨
처음에는 rho, theta를 이용하여 거친(coarse) 스케일로 직선을 검출하고,
더욱 자세한(fine) 스케일에서 rho/srn, theta/stn의 정밀도로 계산

Line Detection using Hough Transform

```
ndarrays[1)]  HoughLineP (
        ndarray image,     // 8bits 1채널 이진 영상
         double   rho,       // 거리 간격 (원점으로부터 거리)
        double theta,       // 라디안 간격 (x축과의 각도)
        int   threshold,     // 직선 검출을 위한 임계값
        double   minLineLength=0,   // 검출할 최소 직선의 길이
        double   maxLineGap=0,      // 직선 위의 에지점들의 최대 허용 간격
)
```

1) ndarray lines // CV_32SC4 행렬로 선분의 양 끝점을 저장
CV_32SC4 행렬로 선분의 양 끝점을 저장 (x_1, y_1, x_2, y_2)

https://docs.opencv.org/3.2.0/d7/da8/tutorial_table_of_content_imgproc.html

Hough Transform을 이용한 원 검출

❖Hough Transform

- Line detection
  - 한 개의 직선은 Hough 변환을 통해 한 개의 점으로 사상 됨
- Circle detection
  - 한 개의 원은 Hough 변환을 통해 한 개의 점으로 사상 됨

Circle Detection using Hough Transform

```
circles [1]  HoughCircles (
        ndarray  image,     // 명도 영상
        int method=HOUGH_GRADIENT,   // 원 검출 방식 (디폴트)
        double dp,      // 해상도의 inverse ratio (1이면 입력영상과 동일한 해상도)
        double minDist,  // 원의 중심점들 간의 최소 거리
        double  param1=100,  double  param2=100,[2]
        double  minRadius, double maxRadius
)
```

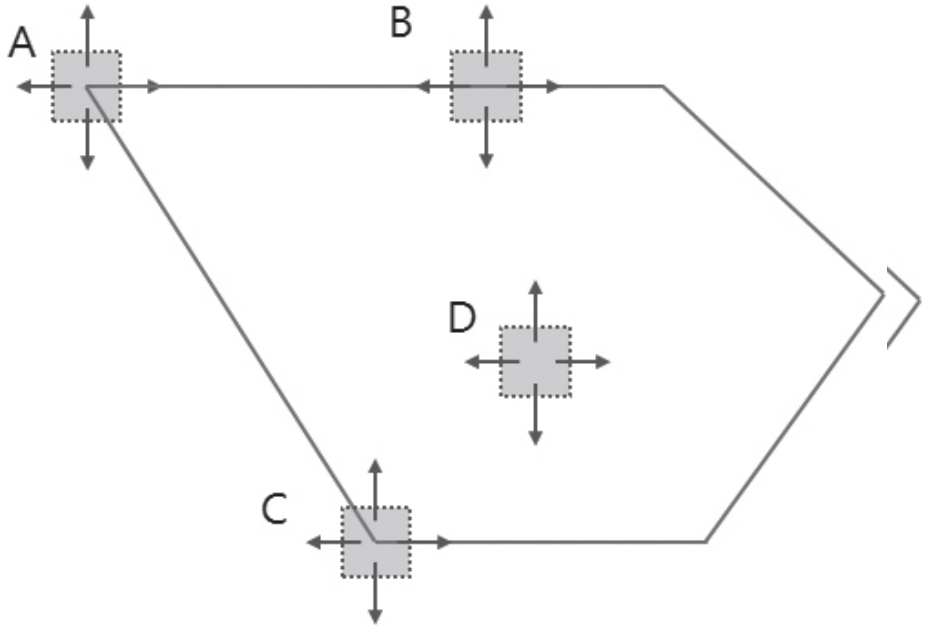
1)ndarray circles // vector<Vec3f> (x_c, y_c, r)저장
원점의 좌표(x_c, y_c) 와 원의 반지름 r

2) $param1$ Canny에지 검출에서 사용되는 higher threshold (lower threshold은 이것의 절반),
$param2$ 원 검출에 사용하는 threshold

2. 코너 검출 (Corner Detection)

❖꼭지점 혹은 코너(corner)

- 모든 방향에서 영상의 밝기 변화가 크게 나타남



Corner Detection

❖모라벡(Moravec) 검출

- 밝기 변화량 E 를 SSD로 계산 (SSD: Sum of Squared Difference)

$$E(u,v) = \sum_{y}\sum_{x} w(x,y) \cdot (I(x+u, y+v) - I(x,y))^2$$

  - 현재 화소에서 u, v 방향으로 이동했을 때의 밝기변화량의 제곱
  - (u,v)를 (1, 0), (1, 1), (0, 1), (-1, 1)의 4개 방향으로 한정
  - 4개 방향의 중 최소값을 해당 픽셀의 영상 변화량으로 지정해서 '특징 가능성' 값으로 결정

- 문제점
  - 0과 1의 값만 갖는 이진 윈도우 사용으로 노이즈에 취약
  - 4개 방향으로 한정시켰기 때문에 45도 간격의 에지만 고려

Corner Detection

❖해리스(Harris) 검출

- 이진 윈도우 $w(x,y)$ 대신에 점진적으로 변화하는 가우시안 마스크 $G(x,y)$ 적용

$$E(u,v)=\sum_y\sum_x G(x,y)\cdot(I(x+u,y+v)-I(x,y))^2$$

- 모든 방향에서 검출할 수 있도록 미분 도입

$$E(u,v)\cong\sum_y\sum_x G(x,y)\cdot(vd_y+ud_x)^2 \quad ,where \quad d_x=\frac{\partial I(x,y)}{\partial x}, d_y=\frac{\partial I(x,y)}{\partial y}$$

$$=\sum_y\sum_x G(x,y)\cdot(v^2d_y^2+u^2d_x^2+2vud_xd_y)$$

$$=\sum_y\sum_x G(x,y)\cdot(u\ v)\begin{pmatrix}d_x^2 & d_xd_y\\ d_xd_y & d_y^2\end{pmatrix}\begin{pmatrix}u\\ v\end{pmatrix}$$

$$=(u\ v)M\begin{pmatrix}u\\ v\end{pmatrix}, \qquad M=\sum_y\sum_x G(x,y)\begin{pmatrix}d_x^2 & d_xd_y\\ d_xd_y & d_y^2\end{pmatrix}$$

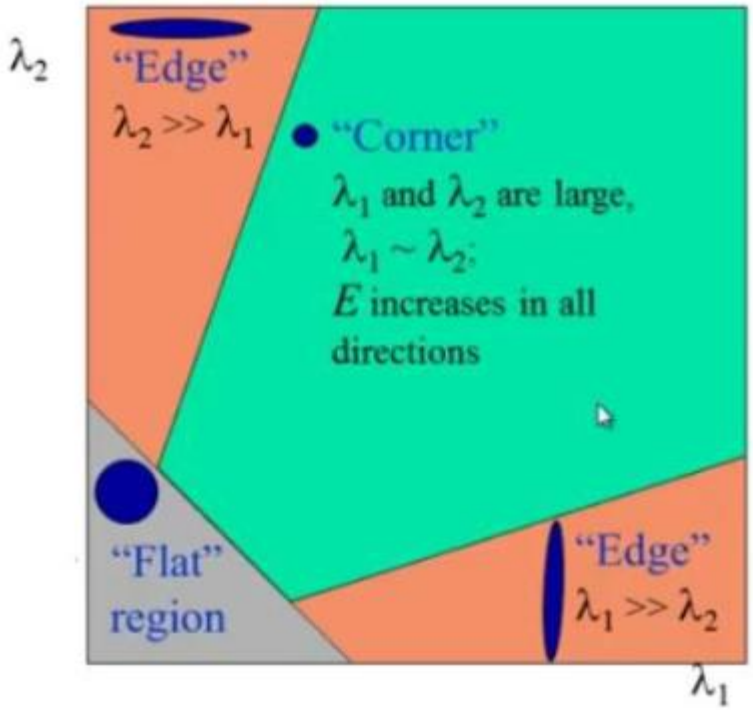
Harris corner detection

- 행렬 *M*에서 고유벡터를 구하면 두 개의 벡터를 얻음

$$M=\sum_{y}\sum_{x}G(x,y)\begin{pmatrix} d_x^2 & d_xd_y \\ d_xd_y & d_y^2 \end{pmatrix}$$

- 행렬 *M*의 고유값(λ_1, λ_2)으로 **코너 응답 함수 *R*** 계산

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$ *,where K*는 상수값으로 0.04~0.06 정도가 적당



|R|이 작으면, 즉 λ_1과 λ_2가 작으면
→ Flat 영역

R<0 이면, 즉 $\lambda_1 \gg \lambda_2$ or $\lambda_1 \ll \lambda_2$ 이면
→ Edge 영역

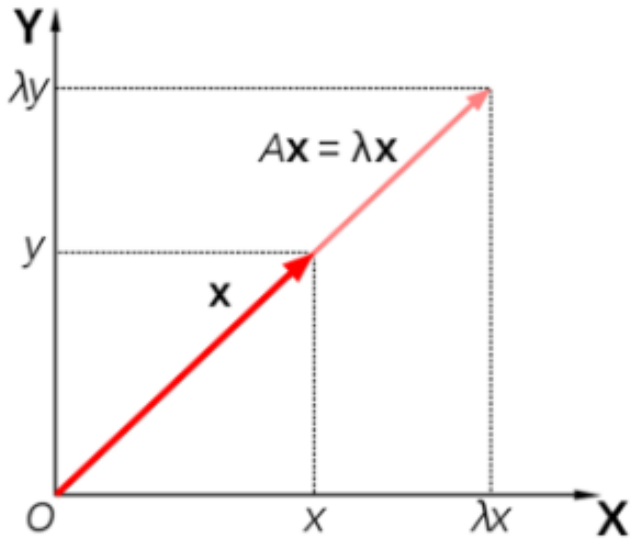
R이 크면, 즉 λ_1과 λ_2가 모두 크면
→ Corner 영역

Harris corner detection

- 고유값과 고유벡터의 수학적 정의
  : $n \times n$ 행렬 A에 0이 아니 벡터 x를 곱합 Ax가 x의 스칼라 배이면,

$$Ax = \lambda x$$

  : x를 A 의 고유벡터, 스칼라 λ를 A 의 고유값이라고 한다.



Harris corner detection

- 행렬 *M*의 고유값(λ_1, λ_2)으로 **코너 응답 함수 *R*** 계산

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$ *,where K*는 상수값으로 0.04~0.06 정도가 적당

- 고유값 대신 행렬식(det)과 대각합(trace)을 코너 응답 함수로 이용
  ∵ 고유값 계산은 고유값 분해의 복잡한 과정 거침

$$M=\begin{pmatrix} d_x^2 & d_xd_y \\ d_xd_y & d_y^2 \end{pmatrix}=\begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

$$R=det(M)-k\cdot trace(M)^2=(ab-c^2)-k\cdot(a+b)^2$$

Harris corner detection

❖Harris 코너 검출 알고리즘

$$M=\begin{pmatrix} d_x^2 & d_x d_y \\ d_x d_y & d_y^2 \end{pmatrix}=\begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

$$R=det(M)-k\cdot trace(M)^2=(ab-c^2)-k\cdot(a+b)^2$$

1. 소벨 마스크로 미분 행렬 계산 (dx, dy)
2. 미분 행렬의 곱 계산 (dx^2, dy^2, dxy)
3. 곱 행렬에 가우시안 마스크 적용
4. 코너 응답함수 $R=det(M)-k\cdot trace(M)^2$ 계산
5. 비최대치 억제

Harris corner detection

```
dstImg [1]  cornerHarris(
            ndarray srcImg,
             int   blockSize,       // 이웃 크기 (blockSize x blockSize)
            int   ksize=3,          // Sobel연산자의 직경
            double k=0.04,          // Harris detector 상수 k
            int   borderType=BORDER_DEFAULT
)
```

1) ndarray dstImg // 코너 응답 값 (자료형: CV_32FC1)

https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

3. k-NN Classifier

❖최근접 이웃 알고리즘 (k=1, NN)

- 기존에 가지고 있는 데이터들을 일정한 규칙에 의해 분류된 상태에서, 새로운 입력 데이터의 종류를 예측하는 분류 알고리즘
- 학습 클래스의 샘플들과 새 샘플의 거리가 가장 가까운(nearest) 클래스로 분류
- 가장 가까운 거리(?)
  - 미지의 샘플과 학습 클래스 샘플간의 유사도가 가장 높은 것을 의미
  - 유클리드 거리(Euclidean distance), 해밍 거리(Hamming distance), 차분 절대값

k-NN classifier

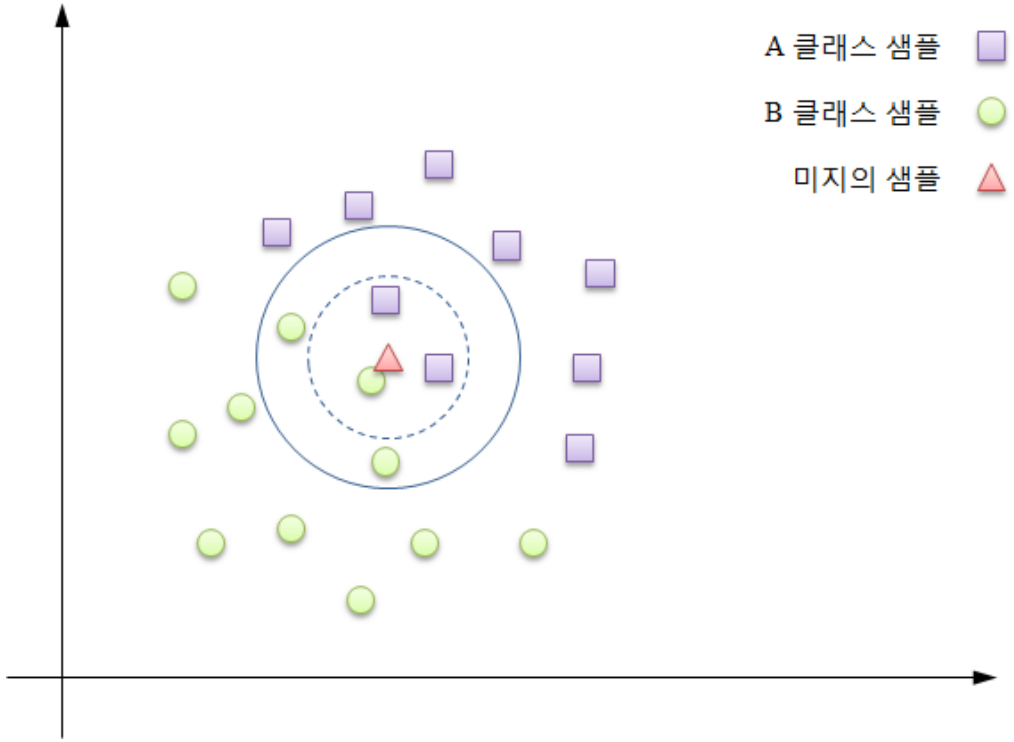
❖k-최근접 이웃 분류(k-Nearest Neighbors: k-NN)

- 학습된 클래스들에서 여러 개(k)의 가까운 이웃을 선출하고 이를 이용하여 미지의 샘플들을 분류하는 방법

If k = 1, then B클래스.

If k = 3, then A클래스.

If k = 5, then B클래스.

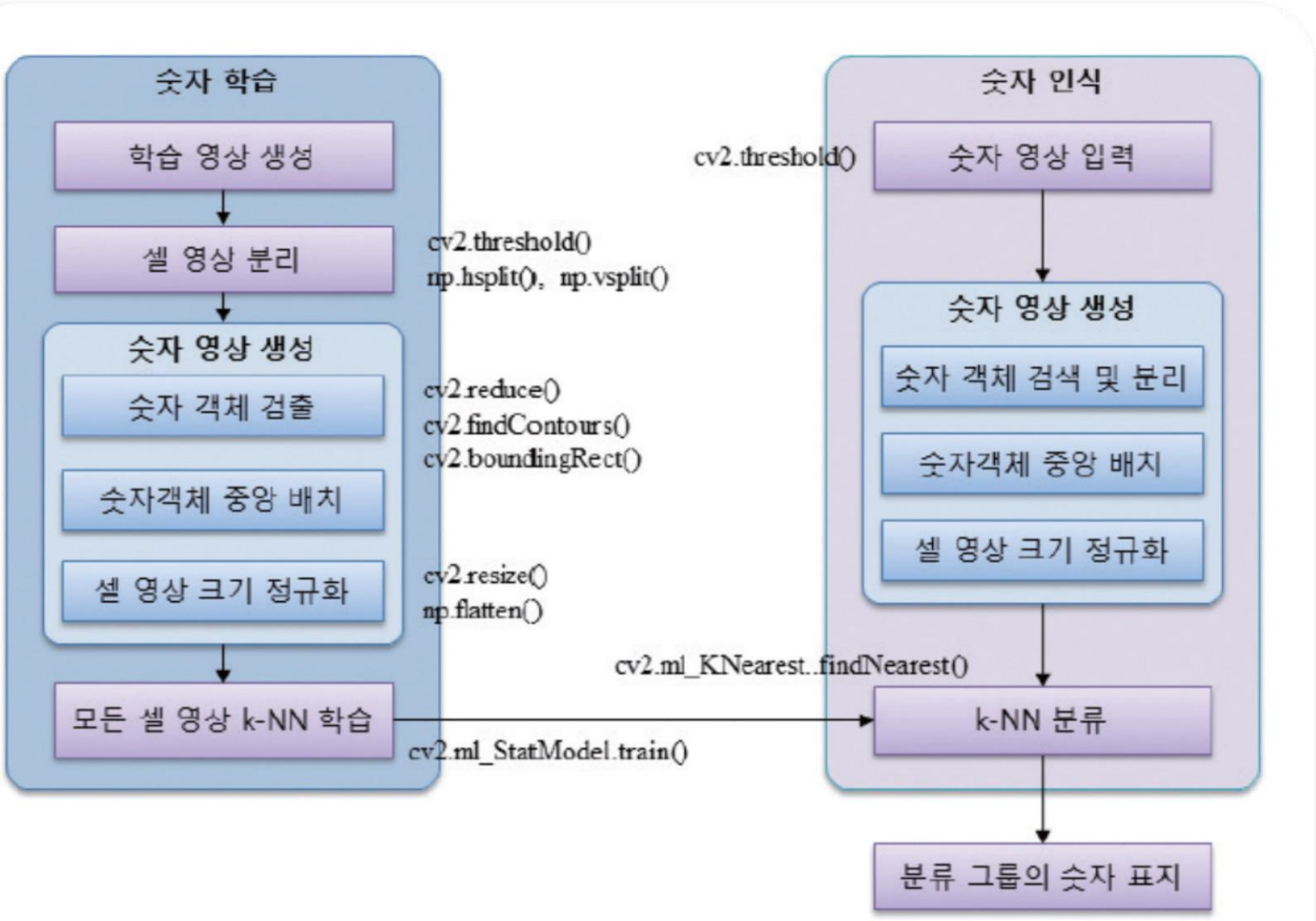


MNIST 데이터 사용

❖MNIST(Modified National Institute of Standards and Technology) 데이터셋

- 필기 숫자 영상으로 구성된 대형 데이터베이스
- 다양한 영상처리 시스템을 학습하기 위해 일반적으로 사용
- 다운로드 http://deeplearning.net/data/mnist/mnist.pkl.gz

k-NN classifier 응용: 숫자 인식기



〈그림 10.3.2〉 숫자 인식 예제 전체 수행 과정

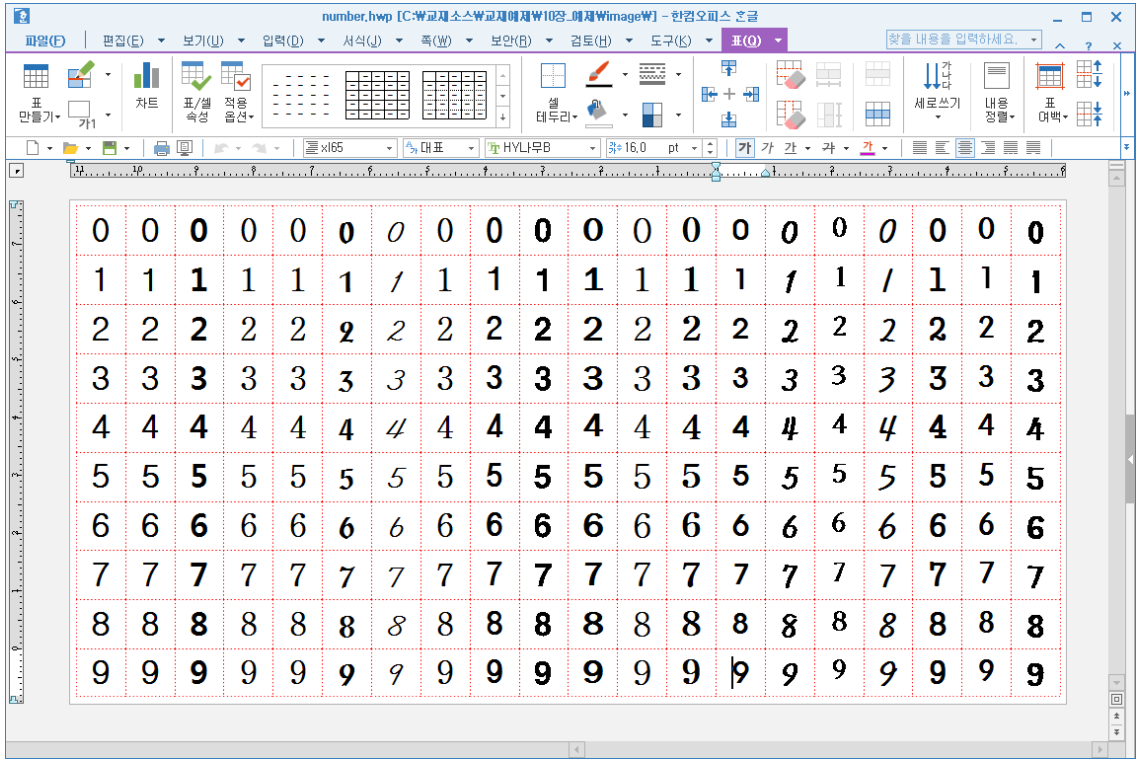
k-NN classifier 응용: 숫자 인식기

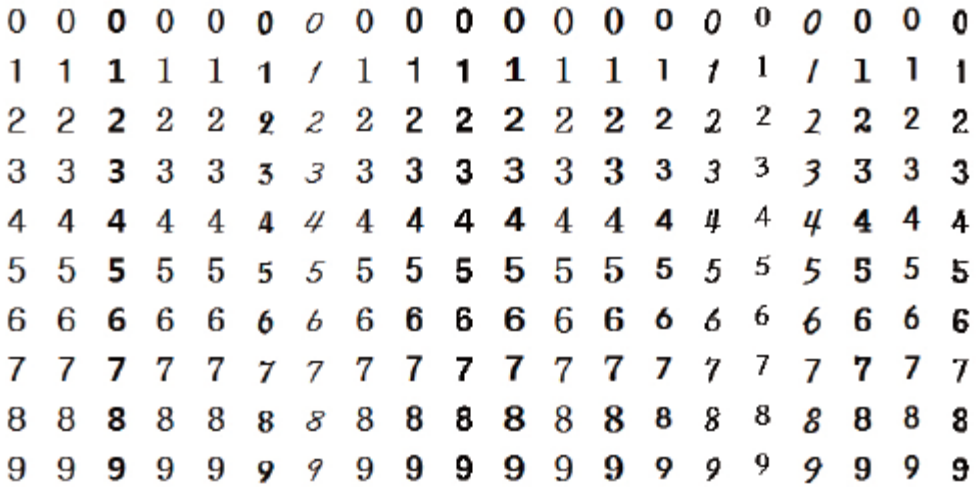
- ❖학습 영상 생성
- ❖학습 영상 이진화
- ❖학습 영상에서 숫자 영상 분리
  - ▪숫자 객체 위치 검색 및 분리 (by projection histogram)
    - find_number()
  - ▪숫자 객체 중앙 배치 및 정규화
    - place_middle()
- ❖학습 or 분류 (by k-NN classifier)
    - Knn->findNearest() // OpenCV 함수 in ML

k-NN classifier 응용: 숫자 인식기

❖학습 영상 생성

- 한글 파일로 작성: number.hwp



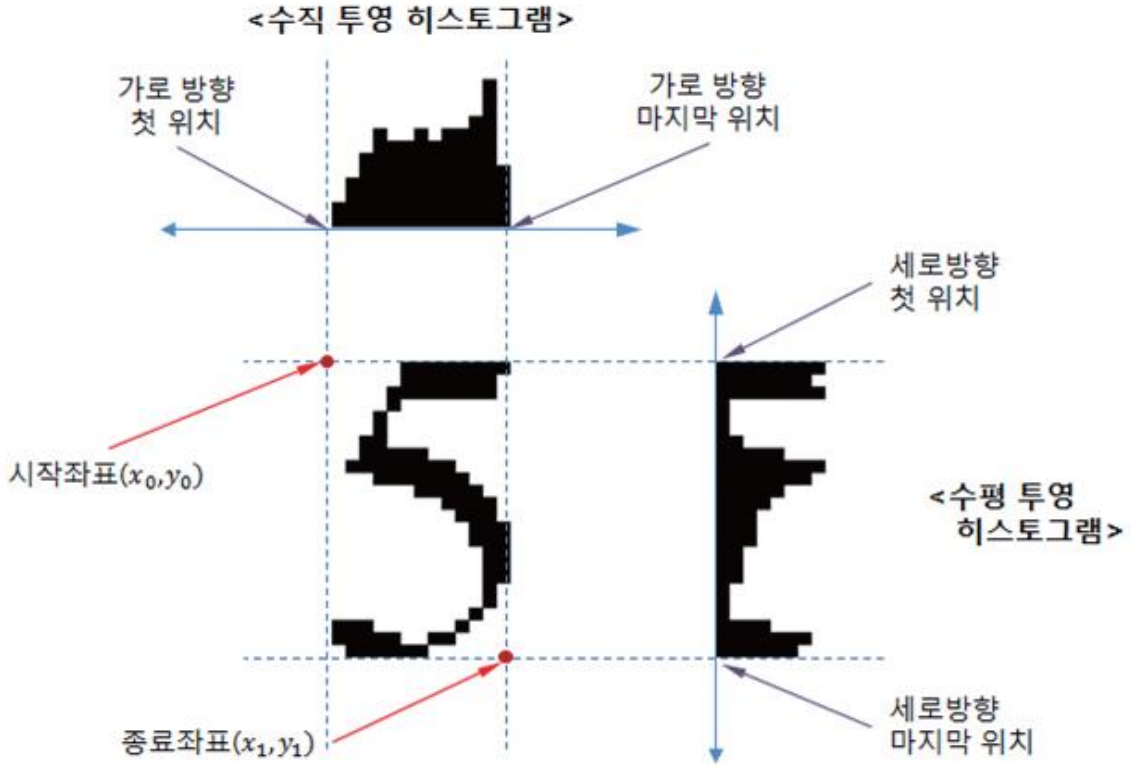


〈그림 10.3.3〉 학습을 위한 숫자 영상의 예

k-NN classifier 응용: 숫자 인식기

❖이진 영상에서 숫자 영상 분리

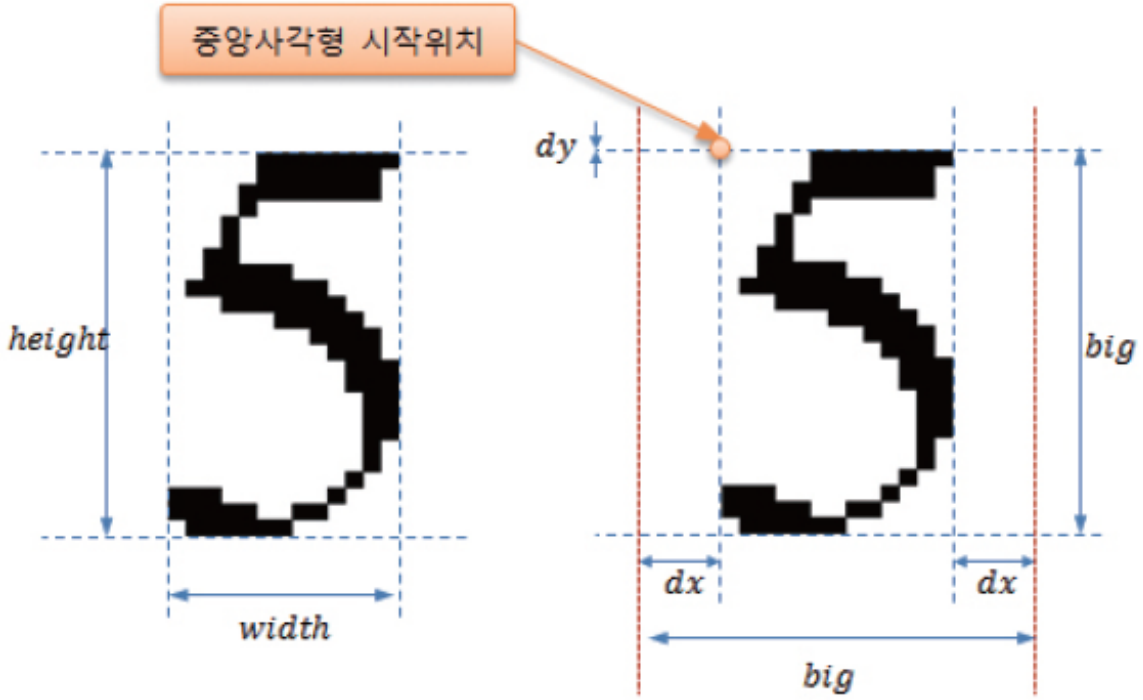
- 투영 히스토그램을 통한 시작 좌표, 종료 좌표 찾기
- cv::reduce() 함수: 가로/세로 방향을 감축 → 이 함수로 투영 구현



k-NN classifier 응용: 숫자 인식기

❖이진 영상에서 숫자 영상 분리

- 숫자 객체를 중앙에 배치 및 정규화



$$big = max(width, height)$$

$$dx = \frac{big - width}{2}$$

$$dy = \frac{big - height}{2}$$

〈그림 10.3.6〉 숫자객체 영상 중심 배치 계산

K-means Clustering Algorithm

Step1: 군집 개수 K를 고정하고, $t = 0$로 초기화한다.
K개의 군집 $C_i^0, i = 1, 2, \dots, K$의 평균 $m_i^0, i = 1, 2, \dots, K$ 을 임의로 선택

Step2: 군집화하려는 데이터 $x_i, 1, 2, \dots, M$ 각각에 K개의 군집 평균과의 최소거리가 되는 군집 C_p^t 로 x_j를 분류한다.

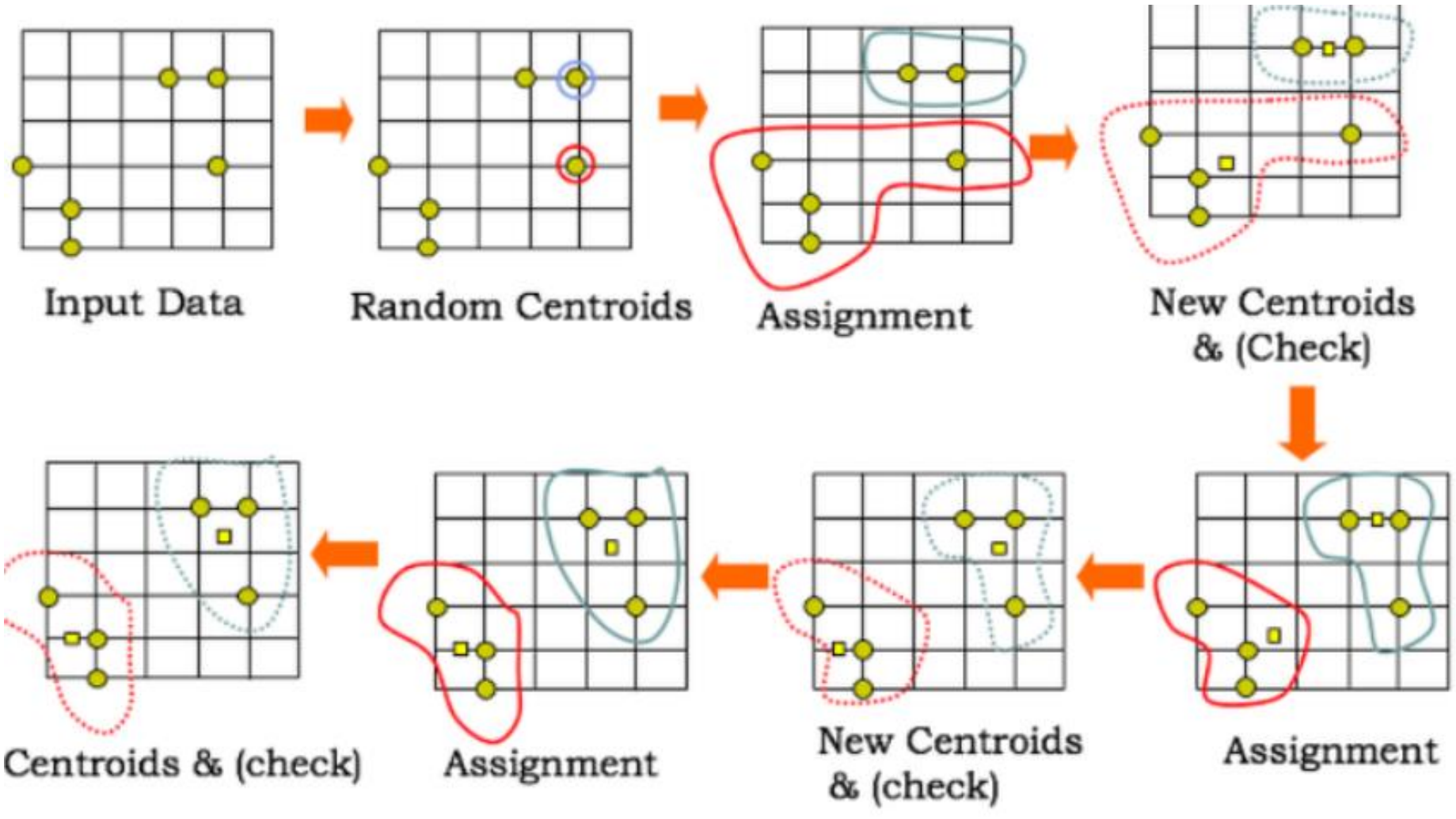
$$p = argmin_i \left| x_j - m_i^t \right|, i = 1, 2, \dots, K$$

Step3: 각 군집 $C_i^t, i = 1, 2, \dots, K$에 속한 데이터를 이용하여 새로운 군집 평균 $m_i^{t+1}, i = 1, 2, \dots, K$ 을 계산한다

$$m_i^{t+1} = \frac{1}{\left| C_i^t \right|} \sum_{x_j \in C_i^t} x_j, i = 1, 2, \dots, K$$

Step4: $t = t + 1$로 증가시키고, 아래 중지조건을 만족시키면 중지하고, 그렇지 않으면 Step2~Step4를 반복한다.

중지조건: $t > MAX\_ITER$ 또는 $\text{err} = \sum_{i=1}^{K} \left| m_i^t - m_i^{t+1} \right| < \varepsilon$



출처: https://blog.naver.com/bsw2428/221400954483

K-means Clustering

```
compactness, label, center   kmeans(
        InputArray   data[1],   // 클러스터링을 위한 데이터
        int   K,                // 클러스터의 개수
        InputOutputArray baseLabels,    // 각 샘플의 클러스터 번호
        TermCriteria   criteria,  // 종료 조건(최대 반복횟수, 허용오차)
        int   attempts,           // 알고리즘을 시도하는 횟수
        int   flags[2],           // K개의 클러스터 중심을 초기화하는 방법
)
```

1) 각 샘플 데이터는 data 행렬의 행에 저장된다.

2) KMEANS_RANDOM_CENTERS : 난수를 사용하여 임의로 설정
   KMENAS_PP_CENTERS : Arthur and Vassilvitskii의 방법으로 설정
   KMENAS_USE_INITIAL_LABELS : 처음 시도에서는 사용자가 제공한 레이블을 사용하고, 다음 시도부터는 난수를 이용하여 설정

3) 각 클러스터의 중심은 centers 행렬의 행에 저장된다.

4. Image Warping and Morphing

❖영상 워핑

- 비선형적인 특정한 규칙에 따라 입력 영상을 재추출하여 영상 형태를 변형하는 기술
- 나사에서 인공위성으로부터 전송된 일그러진 영상을 복원하는 용도로 처음 사용
- 일반 크기 변경과 달리 크기 변화의 정도가 영상 전체에 대해 균일하지 않음
- 고무판 위에 영상이 있는 것과 같이 임의의 형태로 구부리는 효과 → 고무 시트 변환

  Ex) 렌즈 왜곡 보정, 스테레오 영상 정합, 파노라마 영상 합성

- 간단한 영상 워핑 규칙 예
  - [예제 10.4.] 마우스 드래그에 반응하는 워핑 변환

$$x' = x + ratio \cdot (pt2.x - pt1.x), \quad ratio = \begin{cases} x < pt1.x & \frac{x}{pt1.x} \\ otherwise & \frac{width - x}{width - pt1.x} \end{cases}$$

$$y' = y$$

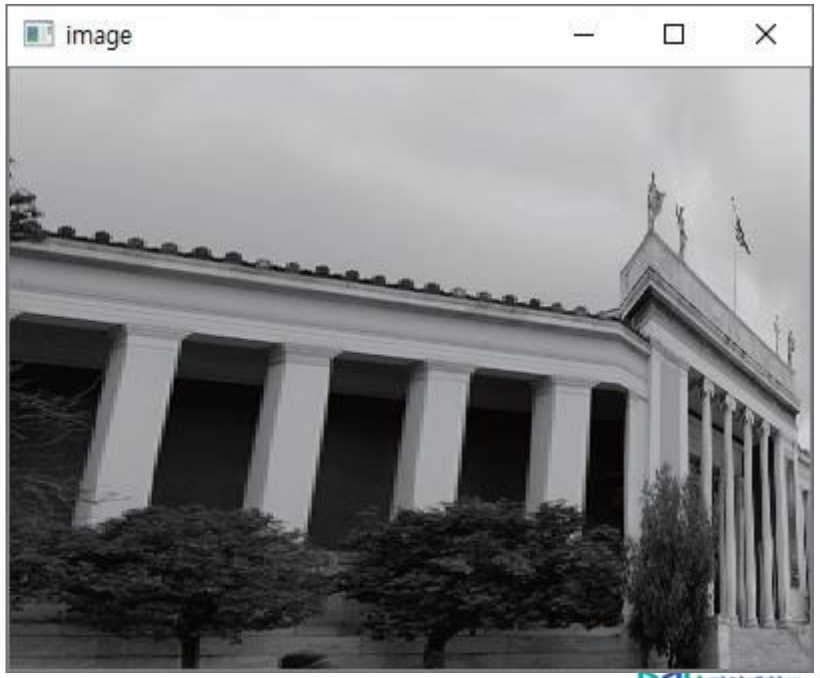
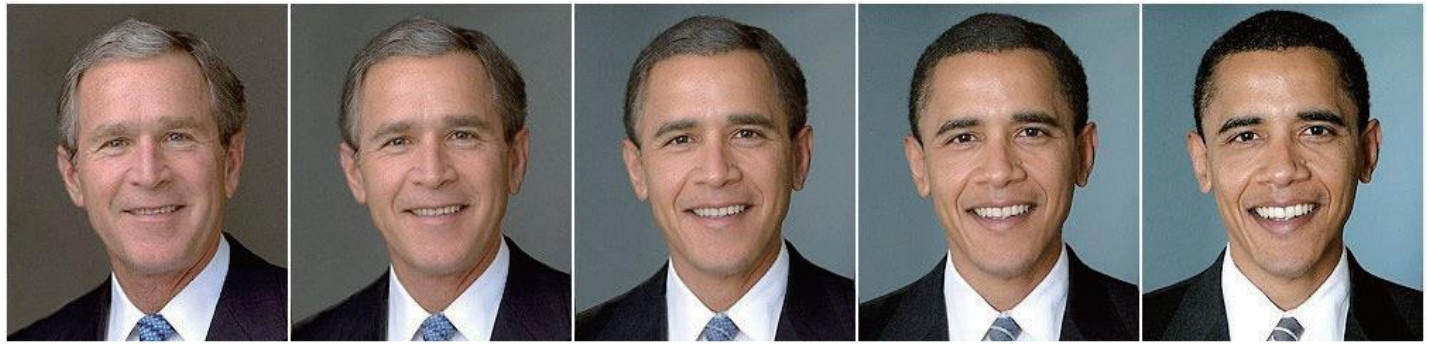


Image Warping and Morphing

❖영상 모핑

- 하나의 영상에서 형체가 전혀 다른 영상으로 변하도록 하는 기법
- 두 개의 서로 다른 영상 사이의 변화하는 과정을 서서히 나타내는 것
- 변형(metamorphosis)이란 단어에서 유래
- 조지 루카스가 설립한 특수 효과 전문회사인 ILM(Industrial Light and Magic)이 개발



Famous Faces Morphing https://www.youtube.com/watch?v=wZurRt0TidI

영상의 왜곡

❖카메라로 찍은 영상의 왜곡 요인

- 외부 파라미터
  - 3차원의 실세계 영상을 2차원의 평면 영상으로 사상할 때 기하학적인 왜곡　예) 원근 투시 왜곡
- 내부 파라미터
  - 카메라 내부의 기구적인 부분에 의한 왜곡
  - 렌즈, 초첨거리, 렌즈와 이미지 센서가 이루는 각

영상의 왜곡

❖카메라 캘리브레이션(camera calibration)

- 내부 파라미터 값을 구하는 과정
  - 영상 좌표로부터 실세계의 3차원 좌표 계산해야하는 경우
  - 실세계의 3차원 좌표를 평면 영상에 투영된 위치로 계산해야하는 경우
- 정확한 좌표 계산을 위해 카메라 캘리브레이션 필요

영상의 왜곡

❖핀홀(pinhole) 카메라 모델

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A \cdot M \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

외부 파리미터

2차원 영상 평면 좌표

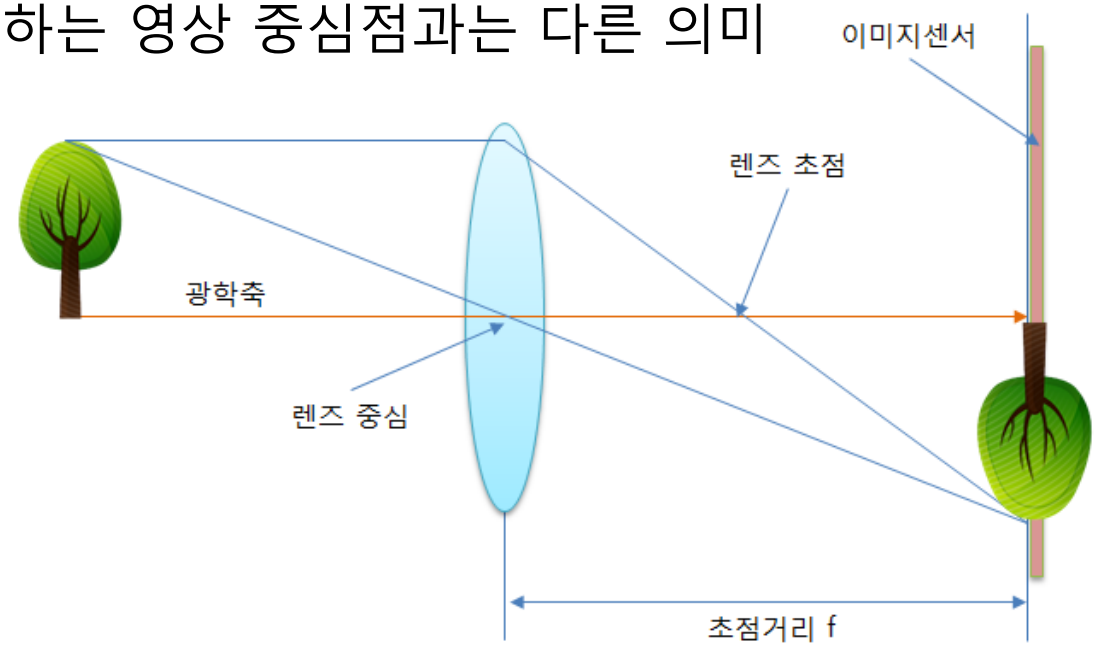
내부 파리미터

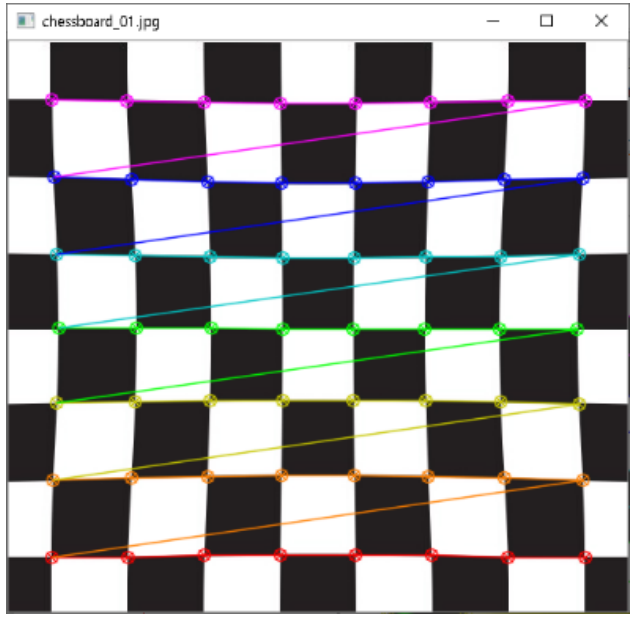
실세계의 3차원 좌표

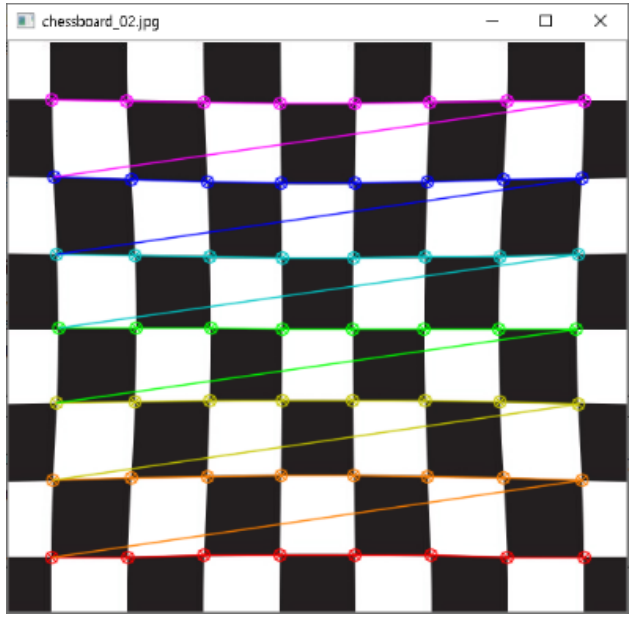
영상의 왜곡

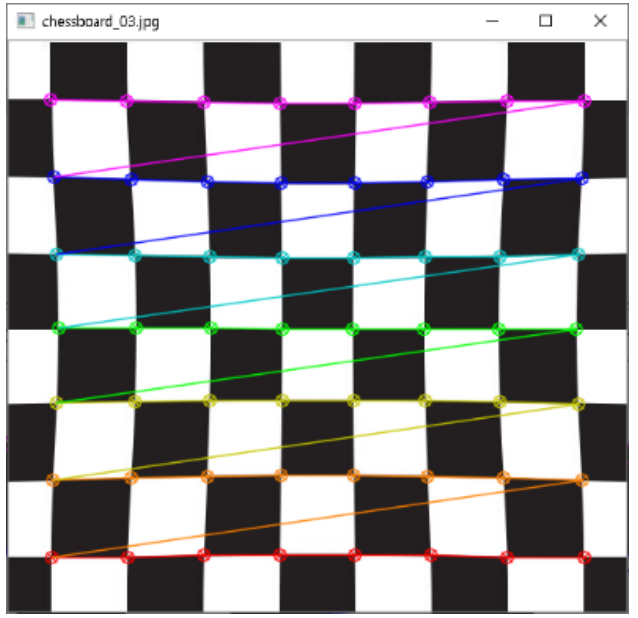
❖초점거리(fx, fy): 렌즈에서 이미지 센서까지의 거리
- 디지털 카메라에서는 mm 단위로 표현, 카메라 모델에서는 픽셀(pixel) 단위로 표현
- 두 개 값으로 표현: 이미지 센서의 가로, 세로의 셀 간격이 다를 수 있기 때문

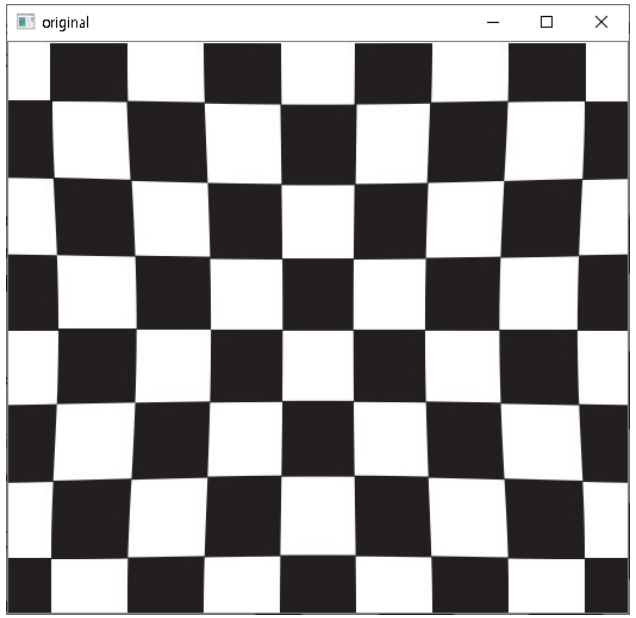
❖주점(Cx,Cy)
- 카메라 렌즈의 중심에서 이미지 센서에 내린 수선의 영상 좌표
- 일반적으로 말하는 영상 중심점과는 다른 의미



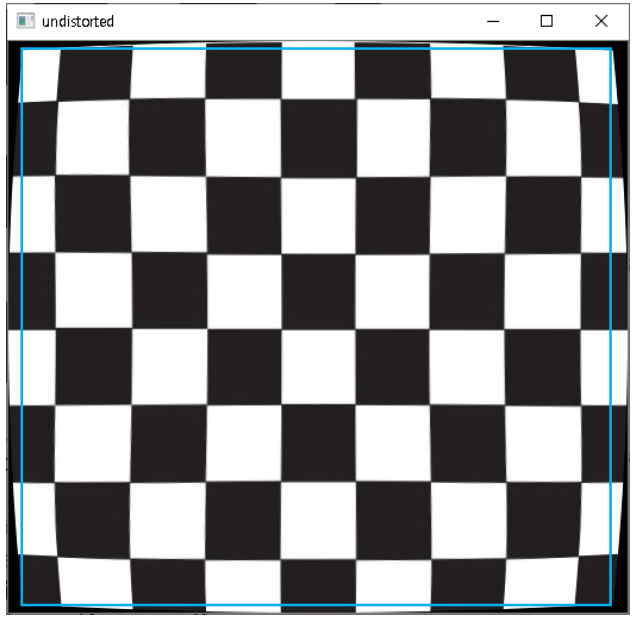




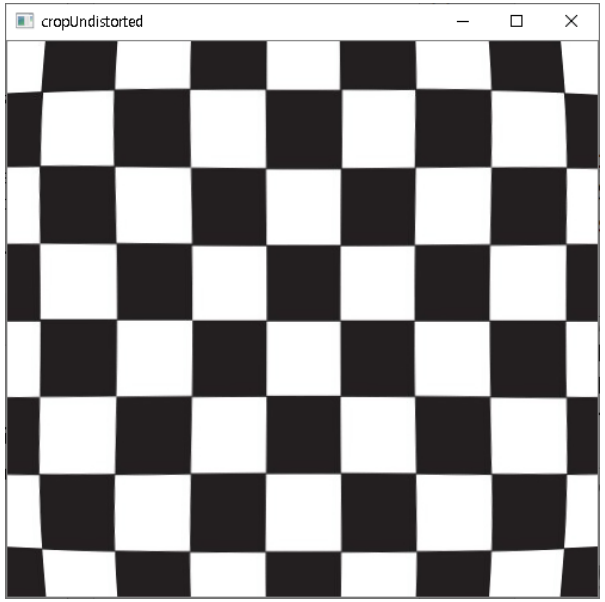




보정할 왜곡 영상



보정 영상



왜곡 제거 보정 영상

요약

- ❖허프 변환: 직선 검출, 원 검출
- ❖코너 검출
  - Corner: 영상에서 경계가 만나는 지점
  - Harris 코너 검출 방법
- ❖최근접이웃(k-NN) 분류(classification) 알고리즘
  - 새로운 미지의 샘플이 입력될 때, 학습 클래스의 샘플들과 새 샘플의 거리가 가장 가까운 클래스로 분류한다.
- ❖K-Means 군집화(clustering) 알고리즘
  - 비지도학습 알고리즘
- ❖워핑과 모핑(warping and morphing)
- ❖카메라 캘리브레이션(camera calibration)