Chapter

8

기하학적 처리

1

기하학적 변환과 매핑

기하학적 변환이란?

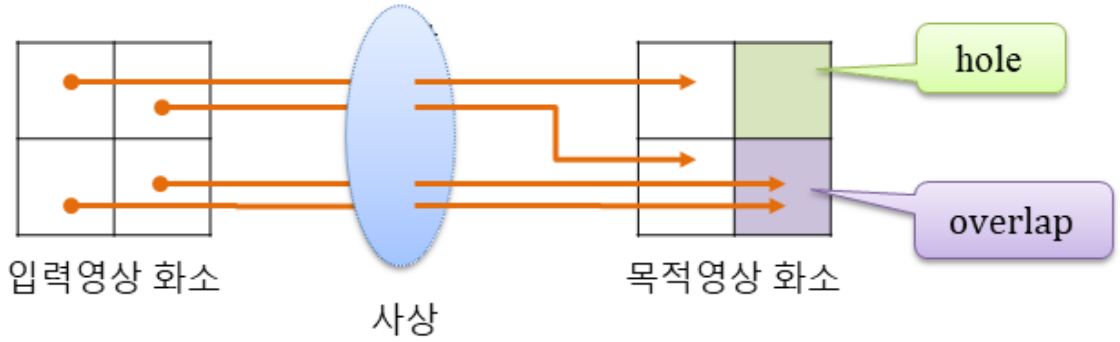
❖ 영상의 기하 변환

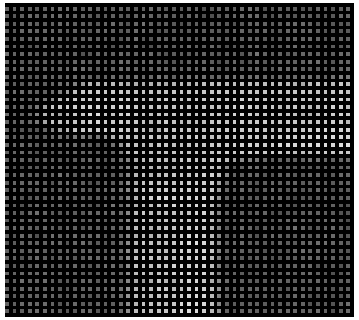
- 영상의 확대, 축소
- 영상의 회전
- 영상의 반사
- 영상의 이미지 모델에 의한 변환
- 사용자 지정 객체(사각형, 구, 타원체)로의 매핑(mapping)

영상의 사상(매핑)

❖ 순방향 사상(forward mapping)

- 원 영상의 각 위치를 목적 영상의 위치(좌표)로 계산 후, 위치를 변환
- 공백(hole) 또는 충돌(overlap)이 발생



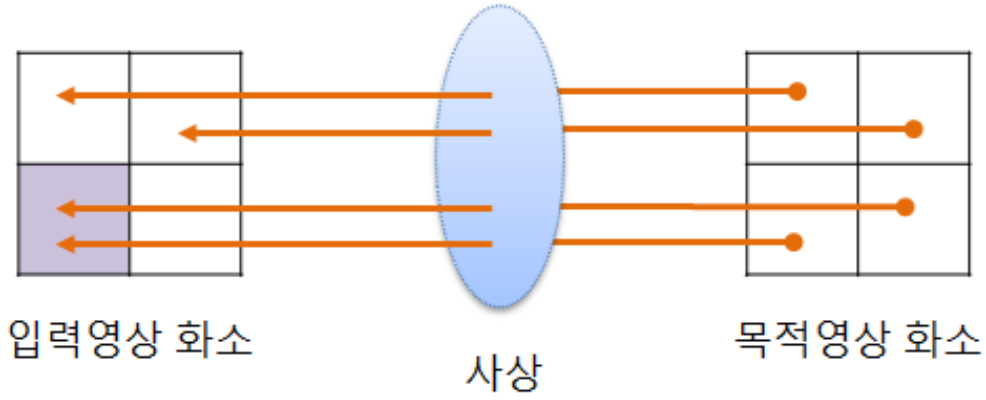


❖ 역방향 사상(backward mapping)

- 순방향 사상의 단점이 발생하지 않음
- 목적영상의 각 위치를 스캔하면서, 해당 목적영상으로 매핑되는 원 영상 위치의 값으로 변환
- 실제 구현에서는 역방향 사상을 사용함

역방향 사상

❖ **해당되는 역 매핑의 좌표가 정수가 아니라 실수 값인 경우가 많음**



❖ **주변 정수값을 이용하여 보간(interpolation)해 주어야 됨**

- 양선형 보간(bilinear interpolation) 기법을 많이 사용함

❖ **최근접 이웃 보간법**

- 확대비율이 커지면, 모자이크 현상 혹은 경계부분에서 계단현상 발생

❖ **양선형 보간법**

- 직선의 선상에 위치한 중간 화소들의 값은 직선의 수식을 이용해서 쉽게 계산

2

확대/축소/이동

확대/축소

❖ 크기 변경(scaling)

- 입력영상의 가로와 세로로 크기를 변경해서 목적영상을 만드는 방법
- 입력영상보다 변경하고자 하는 영상의 크기가 커지면 확대, 작아지면 축소

❖ 크기 변경 수식

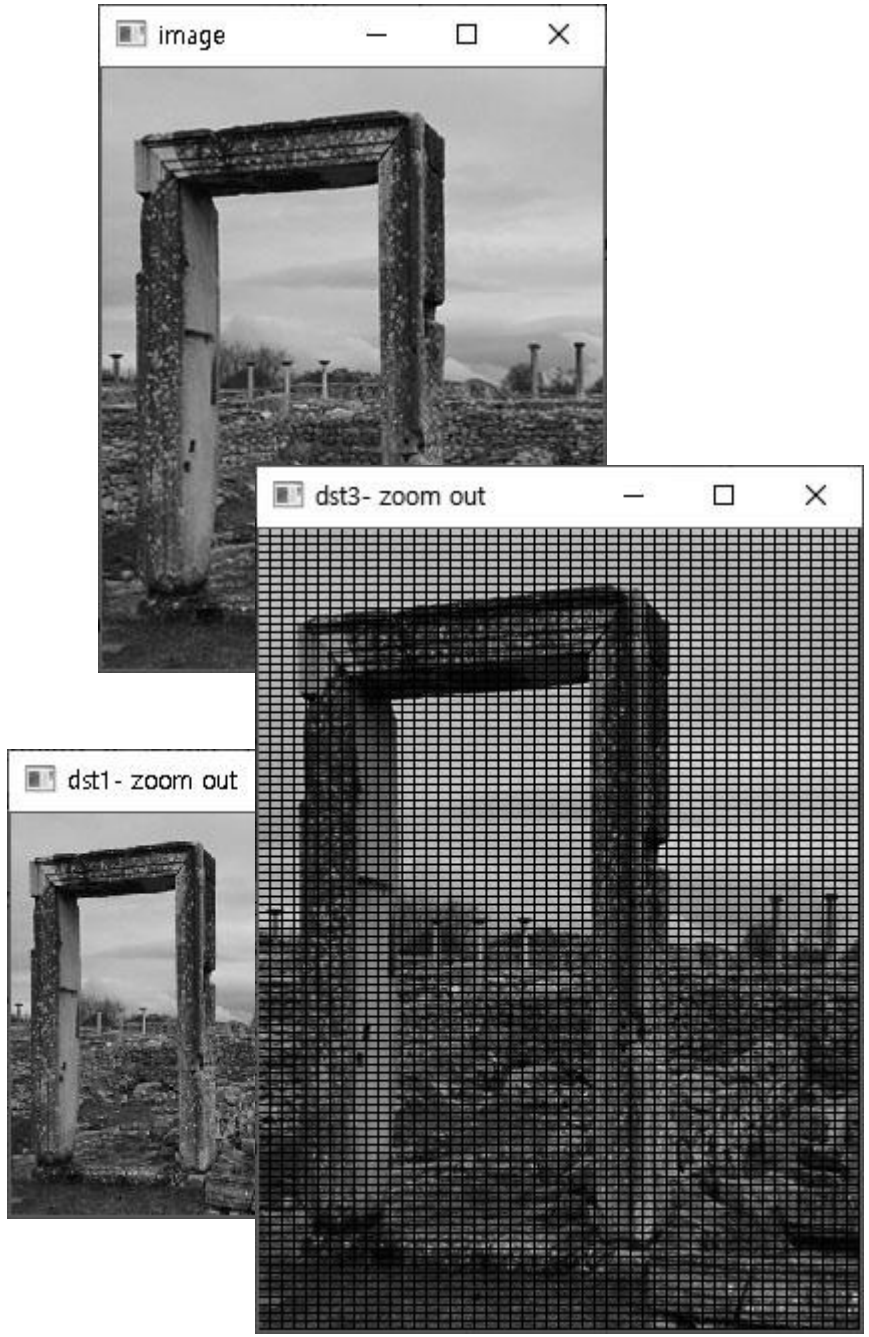
- 변경 비율 및 변경 크기 이용

$$x' = x \cdot ratio\ X$$
$$y' = y \cdot ratio\ Y$$

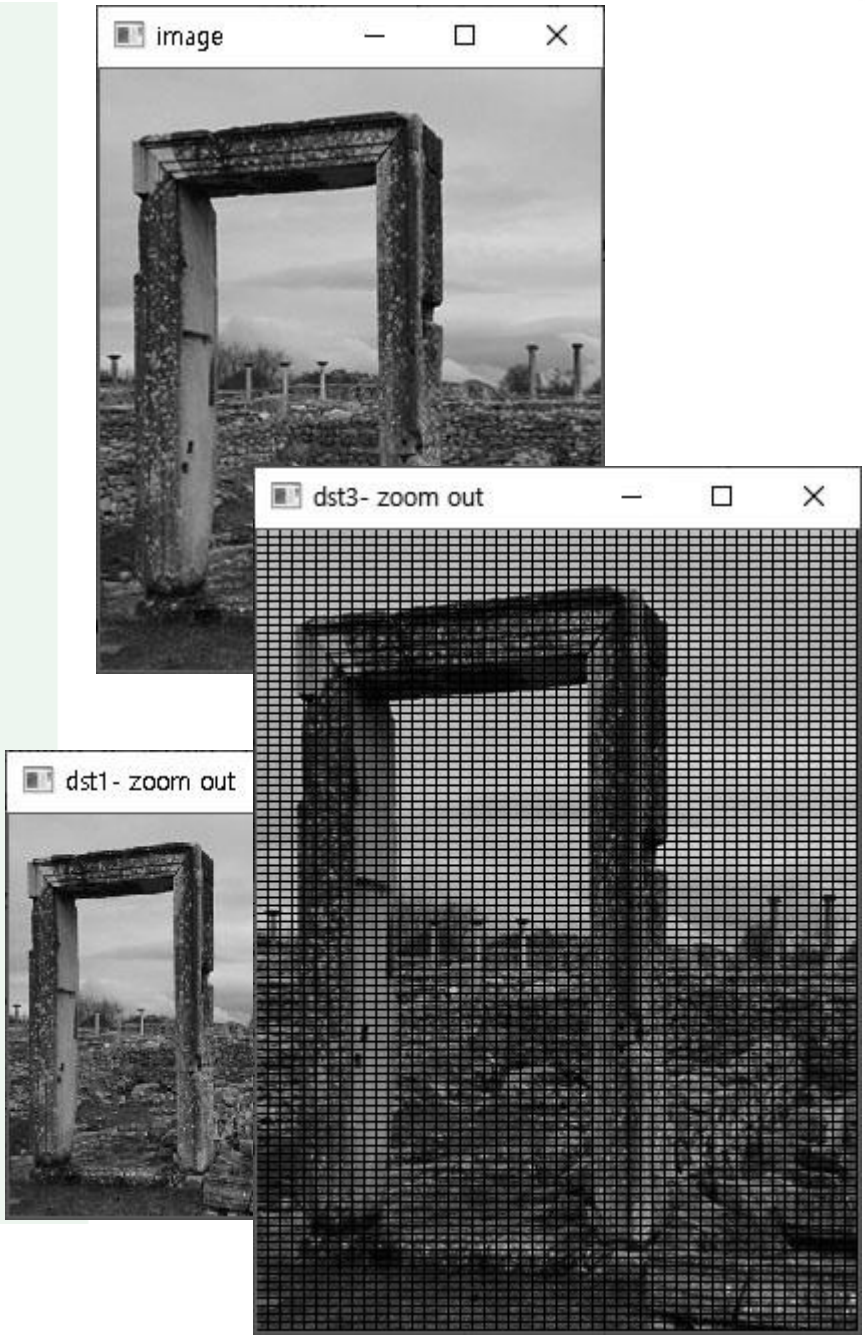
$$ratio\ X = \frac{dst_{width}}{org_{width}}, \quad ratio\ Y = \frac{dst_{height}}{org_{height}}$$

예제 8.2.1 영상 크기 변경 – 01.scaling.cpp

```
01  import numpy as np, cv2
02
03  def scaling(img, size):                                  # 크기 변경 함수1
04      dst = np.zeros(size[::-1], img.dtype)                # size와 shape는 원소 역순
05      ratioY, ratioX = np.divide(size[::-1], img.shape[:2])          # 비율 계산
06      y = np.arange(0, img.shape[0], 1)                    # 입력 영상 세로(y) 좌표 생성
07      x = np.arange(0, img.shape[1], 1)                    # 입력 영상 가로(x) 좌표 생성
08      y, x = np.meshgrid(y, x)                             # i, j 좌표에 대한 정방행렬 생성
09      i, j = np.int32(y * ratioY), np.int32(x * ratioX)    # 목적 영상 좌표
10      dst[i, j] = img[y, x]                                # 정방향 사상→목적 영상 좌표 계산
11      return dst
12
13  def scaling2(img, size):                                 # 크기 변경 함수2
14      dst = np.zeros(size[::-1], img.dtype)
15      ratioY, ratioX = np.divide(size[::-1], img.shape[:2])
16      for y in range(img.shape[0]):                        # 입력 영상 순회– 순방향 사상
17          for x in range(img.shape[1]):
18              i, j = int(y * ratioY), int(x * ratioX) # 목적 영상의 y, x 좌표
19              dst[i, j] = img[y, x]
20      return dst
21
```



```
22  def time_check(func, image, size, title):              # 수행시간 체크 함수
23      start_time = time.perf_counter()
24      ret_img = func(image, size)
25      elapsed = (time.perf_counter() - start_time) * 1000
26      print(title, "수행시간 = %0.2f ms" % elapsed)
27      return ret_img
28
29  image = cv2.imread("images/scaling.jpg', cv2.IMREAD_GRAYSCALE)
30  if image is None: raise Exception("영상파일 읽기 에러")
31
32  dst1 = scaling(image, (150, 200))                      # 크기 변경– 축소
33  dst2 = scaling2(image, (150, 200))                     # 크기 변경– 축소
34  dst3 = time_check(scaling, image, (300,400), "[방법1]: 정방행렬 방식>")      # 확대
35  dst4 = time_check(scaling2, image, (300,400), "[방법2]: 반복문 방식>")      # 확대
36
37  cv2.imshow("image", image)
38  cv2.imshow("dst1- zoom out", dst1)
39  cv2.imshow("dst3- zoom out" , dst3)
40  cv2.resizeWindow("dst1- zoom out", 260, 200)           # 윈도우 크기 확장
41  cv2.waitKey(0)
```



OpenCV의 보간법

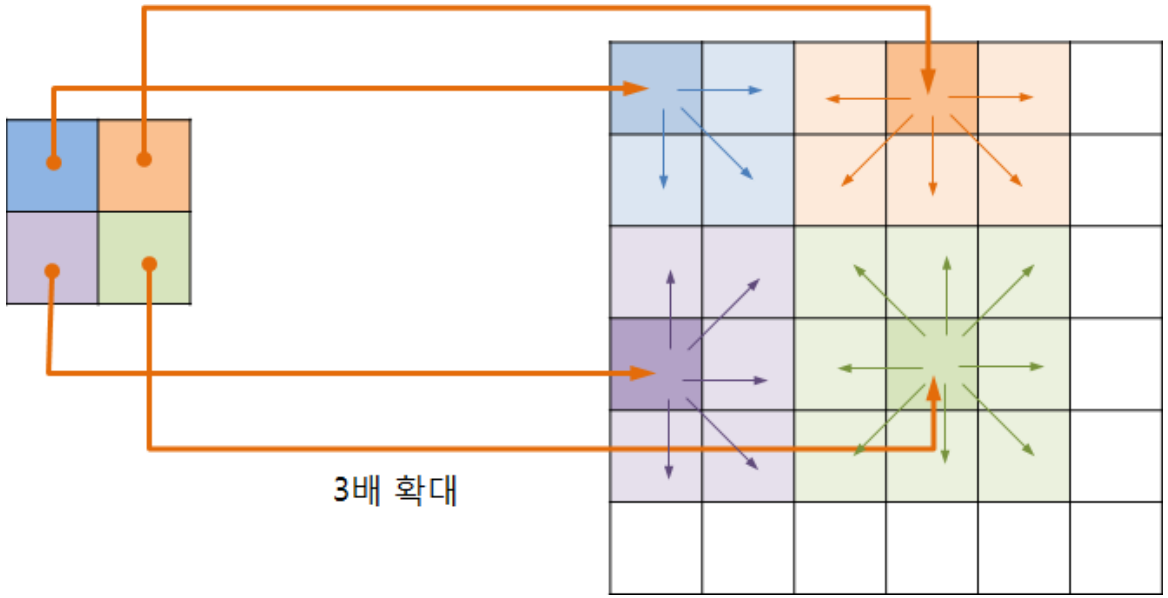
❖ cv2.resize(), cv2.remap(), cv2.warpAffine(), cv2.warpPerspective() 등의 함수에서 사용

〈표 8.3.1〉 보간 방법에 대한 flag 옵션

옵션 상수	값	설명
INTER_NEAREST	0	최근접 이웃 보간
INTER_LINEAR	1	양선형 보간 (기본값)
INTER_CUBIC	2	바이큐빅 보간 – 4x4 이웃 화소 이용
INTER_AREA	3	픽셀 영역의 관계로 리샘플링
INTER_LANCZOS4	4	Lanczos 보간 – 8x8 이웃 화소 이용

최근접 이웃 보간법

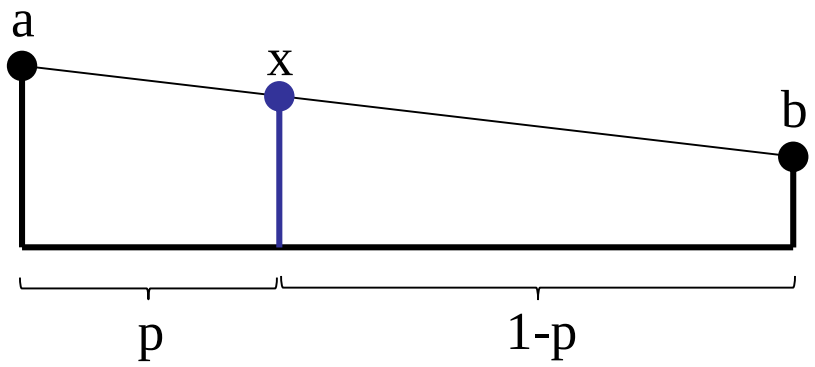
- **목적영상을 만드는 과정에서 홀이 되어 할당 받지 못하는 화소들의 값을 찾을 때, 목적영상의 화소에 가장 가깝게 이웃한 입력영상의 화소값을 가져오는 방법**
  - 확대비율이 커지면, 모자이크 현상 혹은 경계부분에서 계단현상 발생



$$y' = y \cdot ratio\ Y \quad x' = x \cdot ratio\ X \quad \Rightarrow \quad y = \frac{y'}{ratio\ Y},\ x = \frac{x'}{ratio\ X}$$

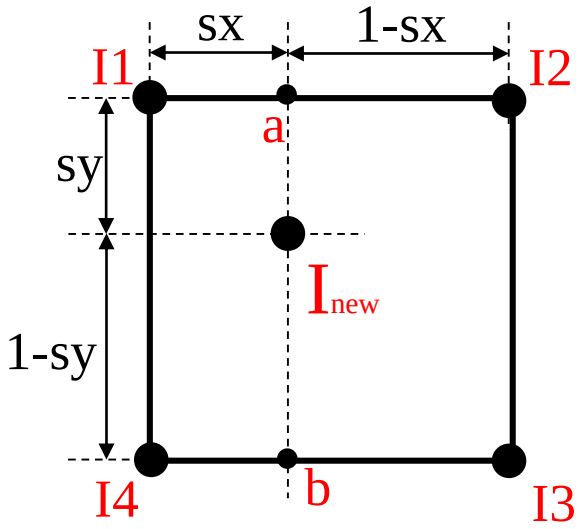
양선형 보간(bilinear interpolation)

❖ 비례법칙



$$x = (1-p)a + pb$$

❖ 양선형 보간



$$a = (1-sx)\,I1 + sx\,I2$$

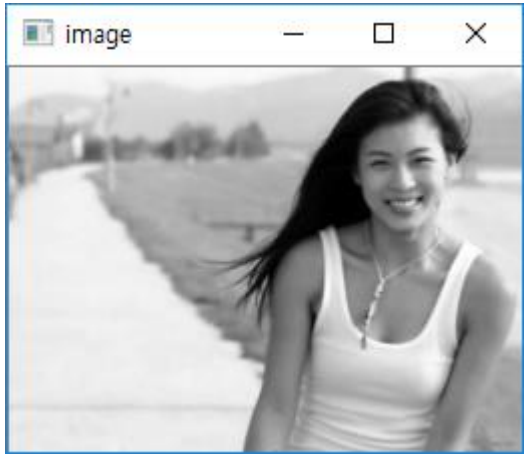
$$b = (1-sx)\,I4 + sx\,I3$$

$$\begin{aligned} I_{new} &= (1-sy)\,a + sy\,b \\ &= I1 * (1-sx)\,(1-sy) + \\ &\quad I2 * sx * (1-sy) + \\ &\quad I3 * sx * sy + \\ &\quad I4 * (1-sx) * sy \end{aligned}$$

예제: 양선형보간을 이용한 확대축소

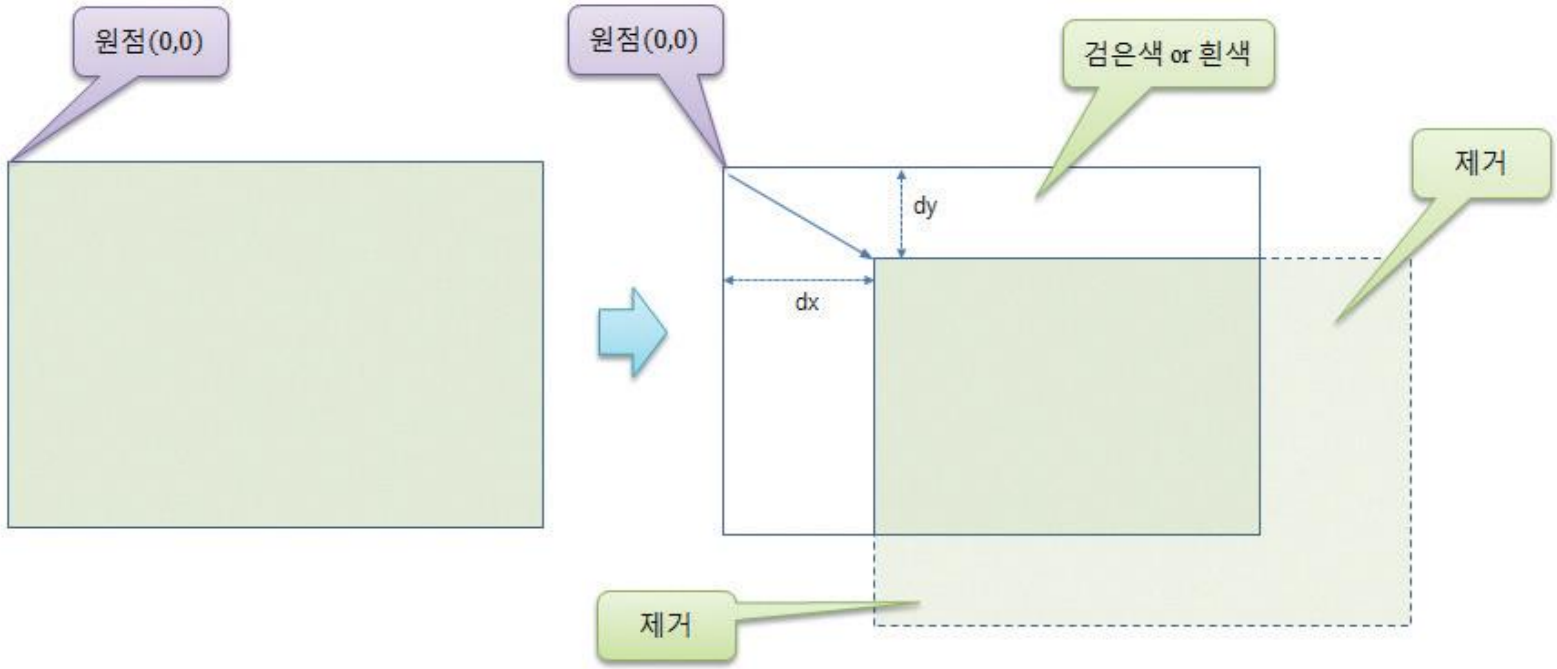
```
import cv2
import numpy as np
import math
from tkinter.filedialog import askopenfilename

def main():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    # 확대축소 비율
    zoomfactor = 0.7

    ysize = img.shape[0] # 원영상의 높이
    xsize = img.shape[1] # 원영상의 너비

    new_ysize = math.floor(img.shape[0] * zoomfactor) # 줌영상의 높이
    new_xsize = math.floor(img.shape[1] * zoomfactor) # 줌영상의 너비

    zoomImg = np.zeros((new_ysize, new_xsize), dtype='uint8')
```

```
    # 역방향 매핑
    for y in range(new_ysize):
        for x in range(new_xsize):
            r_orgy = y / zoomfactor
            r_orgx = x / zoomfactor
            i_orgy = math.floor(r_orgy)
            i_orgx = math.floor(r_orgx)
            sy = r_orgy - i_orgy
            sx = r_orgx - i_orgx
            if i_orgy < 0 or i_orgy > ysize-2
                or i_orgx < 0 or i_orgx > xsize-2:
                continue # 영역을 벗어나는 경우
            I1 = img.item(i_orgy  , i_orgx  )
            I2 = img.item(i_orgy  , i_orgx+1)
            I3 = img.item(i_orgy+1, i_orgx+1)
            I4 = img.item(i_orgy+1, i_orgx  )
            newValue = I1*(1-sx)*(1-sy) + I2*sx*(1-sy)
                     + I3*sx*sy + I4*(1-sx)*sy
            zoomImg.itemset((y, x), newValue)
        # end-for x
    # end-for y
    cv2.imshow('result', zoomImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
main()
```



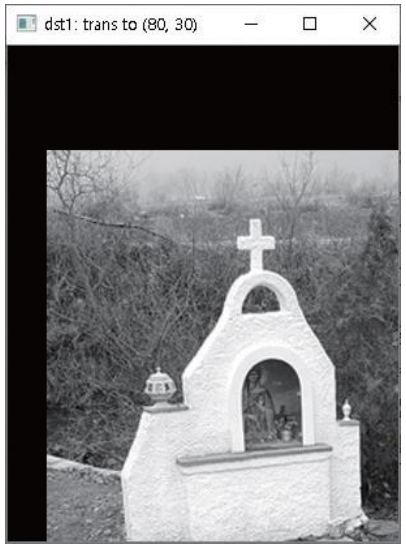
평행이동

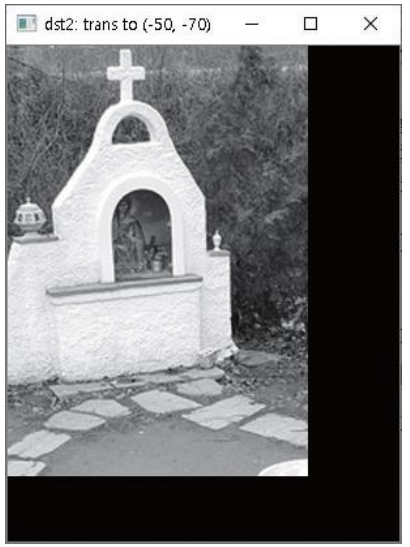
- 영상의 원점을 기준으로 모든 화소를 동일하게 가로방향과 세로 방향으로 옮기는 것
- 가로 방향으로 dx만큼, 세로 방향으로 dy만큼 전체 영상의 모든 화소 이동한 예



예제 8.4.1 영상 평행이동 – 04.translation.cpp

```
01  import numpy as np, cv2
02
03  def contain(p, shape):                          # 좌표(y,x)가 범위내 인지 검사
04      return 0<= p[0] < shape[0] and 0<= p[1] < shape[1]
05
06  def translate(img, pt):
07      dst = np.zeros(img.shape, img.dtype)        # 목적 영상 생성
08      for i in range(img.shape[0]):               # 목적 영상 순회– 역방향 사상
09          for j in range(img.shape[1]):
10              x, y = np.subtract((j, i) , pt)     # 좌표는 가로, 세로 순서
11              if contain((y, x), img.shape):      # 영상 범위 확인
12                  dst[i, j] = img[y, x]           # 행렬은 행, 열 순서
13      return dst
14
15  image = cv2.imread("images/translate.jpg", cv2.IMREAD_GRAYSCALE)
16  if image is None: raise Exception("영상파일 읽기 에러")
17
18  dst1 = translate(image, (30, 80))               # x=30, y=80 으로 평행이동
19  dst2 = translate(image, (-70, -50))
20
21  cv2.imshow("image", image)
22  cv2.imshow("dst1: transted to (30, 80)", dst1);
23  cv2.imshow("dst2: transted to (-70, -50)", dst2);
24  cv2.waitKey(0)
```

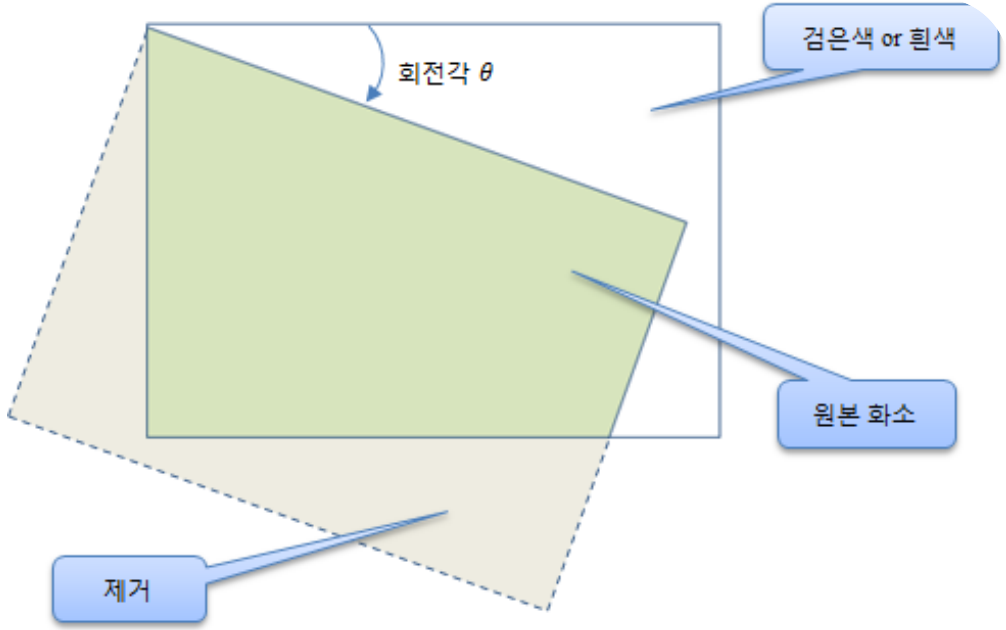




3

회전

❖ 입력영상의 모든 화소를 영상의 원점을 기준으로 원하는 각도만큼 모든 화소에 대해서 회전 변환을 시키는 것



순방향 사상

$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$

역방향사상

$$x = x' \cdot \cos\theta + y' \cdot \sin\theta$$
$$y = -x' \cdot \sin\theta + y' \cdot \cos\theta$$

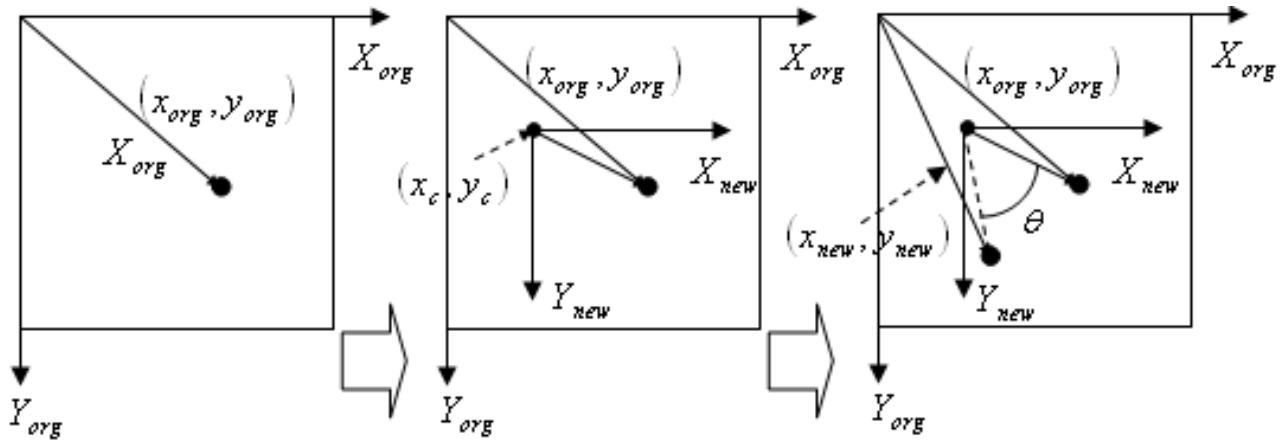
- 직교 좌표계 : 회전 변환은 반시계 방향으로 적용
- 영상 좌표계 : y 좌표가 하단으로 내려갈수록 증가하기 때문에 시계방향 회전

참고 :

❖ 회전변환

$$\begin{pmatrix} X_{new} \\ Y_{new} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_{org} - C_x \\ Y_{org} - C_y \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \end{pmatrix}$$

$$\begin{pmatrix} X_{org} \\ Y_{org} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_{new} - C_x \\ Y_{new} - C_y \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \end{pmatrix}$$



❖ 회전, 크기변환, 이동의 결합

$$\begin{pmatrix} X_{new} - C_x \\ Y_{new} - C_y \end{pmatrix} = Sc \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_{org} - C_x \\ Y_{org} - C_y \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \end{pmatrix}$$

$$\begin{pmatrix} X_{org} - C_x \\ Y_{org} - C_y \end{pmatrix} = \frac{1}{Sc} \cdot \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_{new} - C_x - T_x \\ Y_{new} - C_y - T_y \end{pmatrix}$$

$$\begin{pmatrix} X_{org} \\ Y_{org} \end{pmatrix} = \frac{1}{Sc} \cdot \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_{new} - C_x - T_x \\ Y_{new} - C_y - T_y \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \end{pmatrix}$$

예제 8.5.1 영상 회전 – 05.rotation.py

```
01  import numpy as np, math, cv2
02  from Common.interpolation import bilinear_value        # 양선형 보간 함수 임포트
03  from Common.functions import contain                   # 사각형으로 범위 확인 함수
04
05  def rotate(img, degree):                               # 원점 기준 회전 변환 함수
06      dst = np.zeros(img.shape[:2], img.dtype)           # 목적 영상 생성
07      radian = (degree/180) * np.pi                      # 회전 각도– 라디언
08      sin, cos = np.sin(radian), np.cos(radian)          # 사인, 코사인 값 미리 계산
09
10      for i in range(img.shape[0]):                      # 목적 영상 순회– 역방향 사상
11          for j in range(img.shape[1]):
12              y = -j * sin + i * cos
13              x = j * cos + i * sin                      # 회선 변환 수식
14              if contain((y, x), img.shape):             # 입력 영상의 범위 확인
15                  dst[i, j] = bilinear_value(img, [x, y])      # 화소값 양선형 보간
16      return dst
17
```

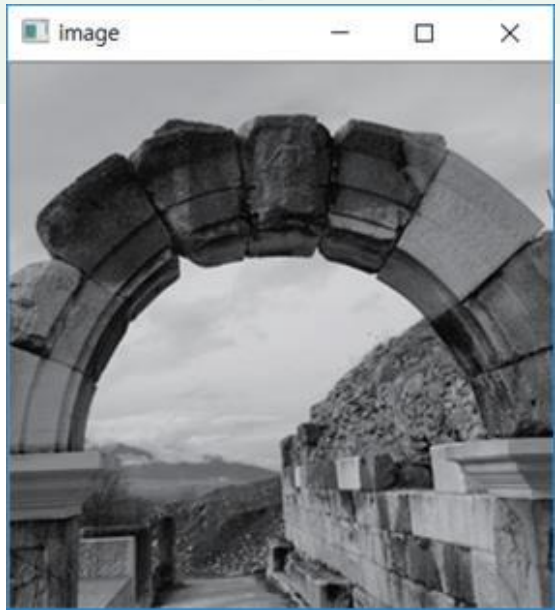
회전 기준점

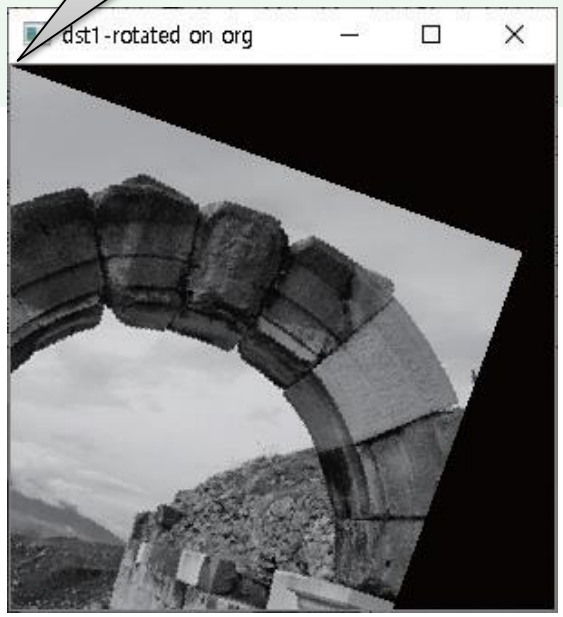
```
18  def rotate_pt(img, degree, pt):                    # pt 기준 회전 변환 함수
19      dst = np.zeros(img.shape[:2], img.dtype)       # 목적 영상 생성
20      radian = (degree/180) * np.pi                  # 회전 각도– 라디언
21      sin, cos = math.sin(radian), math.cos(radian)  # 사인, 코사인 값 미리 계산
22
23      for i in range(img.shape[0]):                  # 목적 영상 순회– 역방향 사상
24          for j in range(img.shape[1]):
                jj, ii = np.subtract((j, i), pt)       # 중심 좌표로 평행이동
26              y = -jj * sin + ii * cos               # 회선 변환 수식
27              x = jj * cos + ii * sin
                x, y = np.add((x, y), pt)              # 중심 좌표로 평행이동
29              if contain((y, x), img.shape):         # 입력 영상의 범위 확인
30                  dst[i, j] = bilinear_value(img, (x, y))    # 화소값 양선형 보간
31      return dst
32
33  image = cv2.imread("images/rotate.jpg", cv2.IMREAD_GRAYSCALE)
34  if image is None: raise Exception("영상파일 읽기 에러")
```

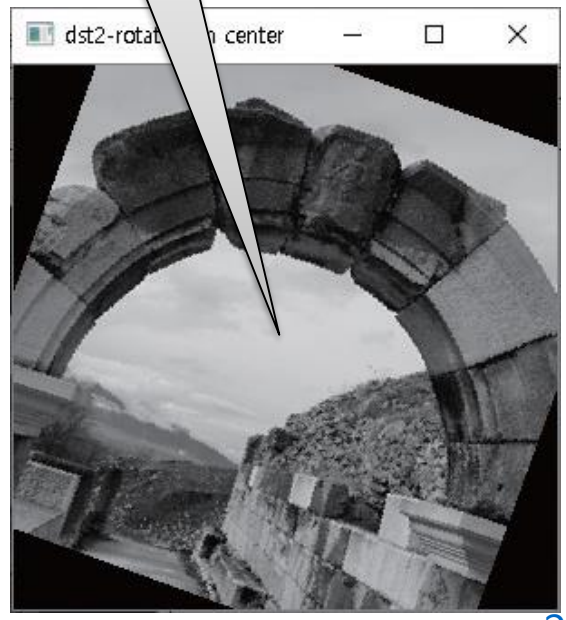
회전 기준점으로 평행이동

회전후 역 평행이동

```
33  image = cv2.imread("images/rotate.jpg", cv2.IMREAD_GRAYSCALE)
34  if image is None: raise Exception("영상파일 읽기 에러")
35
36  center = np.divmod(image.shape[::-1], 2)[0]          # 영상 크기로 중심 좌표 계산
37  dst1 = rotate(image, 20)                             # 원점 기준 회전 변환
38  dst2 = rotate_pt(image, 20, center)                  # center 기준 회전 변환
39
40  cv2.imshow("image", image)
41  cv2.imshow("dst1 : rotated on (0, 0)", dst1);
42  cv2.imshow("dst2 : rotated on center point", dst2);
43  cv2.waitKey(0)
```







4

Affine 변환

Affine 변환

- 회전

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- 크기변경

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- 평행이동

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

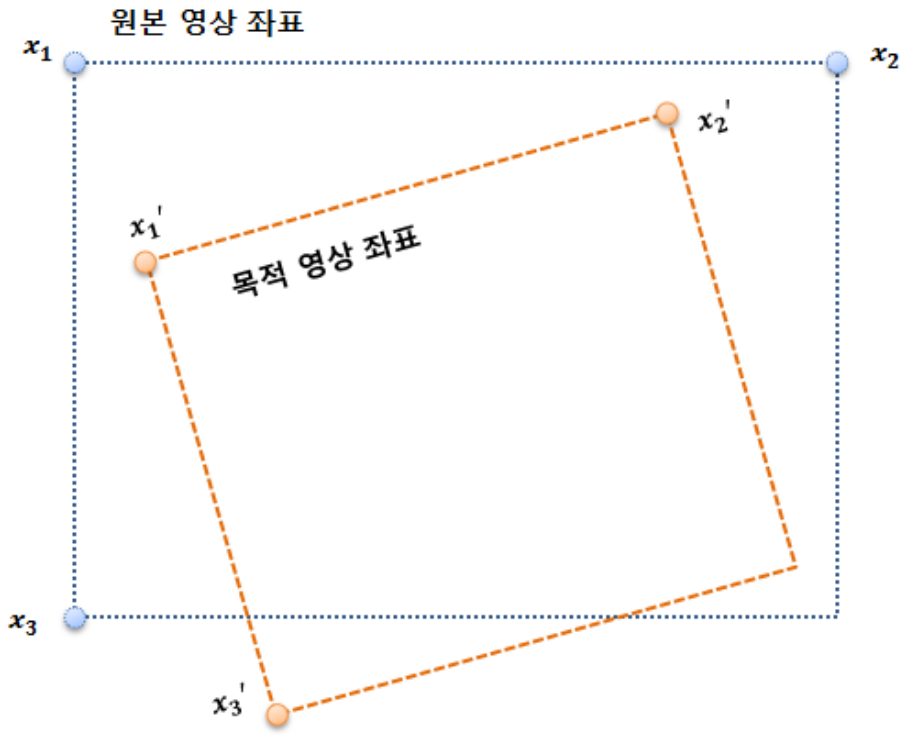
- 어파인 변환 수식

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 각 변환 행렬을 행렬곱으로 구성 → 최종 행렬곱 후에 마지막 행 삭제

$$\text{어파인 변환행렬} = \begin{bmatrix} \cos\theta & -sin'\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

❖ 입력영상의 좌표 3개(x_1 , x_2 , x_3)와 목적영상에 상응하는 좌표 3개 (x_1, x_2, x_3)를 알면 → 어파인 행렬 구성 가능



cv2.warpAffine

❖ **입력영상에 Affine변환을 수행한 결과를 반환**

❖ **cv2.warpAffine(src, M, dsize, dst=None, flags=None, borderMode=None, borderValue=None) -> dst**

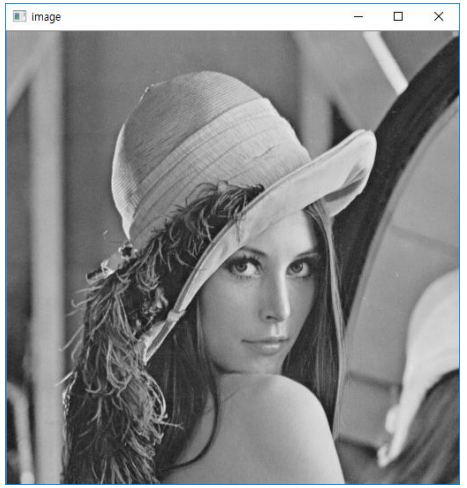
- **src** : 입력 영상
- **M** : 2x3 어파인 변환 행렬. 실수형.
- **dsize** : 결과 영상 크기. (w, h) 튜플. (0, 0)이면 src와 같은 크기로 설정
- **dst** : 출력 영상
- **flags** : 보간법. 기본값은 cv2.INTER_LINEAR
- **borderMode** : 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_CONSTANT
- **borderValue** : cv2.BORDER_CONSTANT일 때 사용할 상수 값. 기본값은 0(검정색)

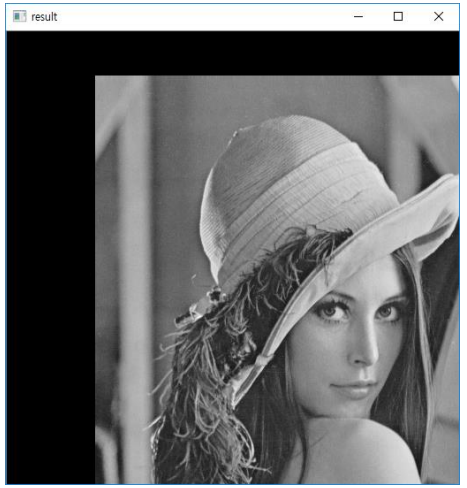
continue…

❖ $M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$: (t_x, t_y)만큼 이동

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

def geometricTransform():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape[0:2]

    M = np.array([[1, 0, 100],\
                  [0, 1,  50]], np.float32)
    result = cv2.warpAffine(img, M, (width,height))

    cv2.imshow('image', img)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

geometricTransform()
```

결과이미지의 크기





Affine 변환행렬 얻기

❖ cv2.getAffineTransform(src, dst) -> retval

- **src** : 3개의 원본 좌표점. numpy.ndarray. shape=(3, 2)
  - (예) np.array([[x1 , y1], [x2 , y2], [x3 , y3]], np.float32)
- **dst** : 3개의 결과 좌표점. numpy.ndarray. shape=(3, 2)

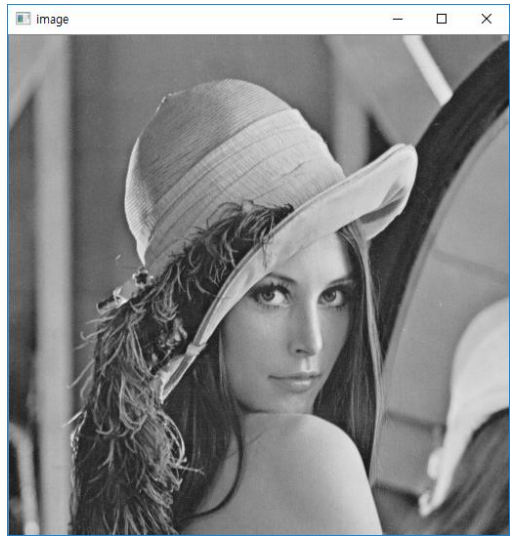
❖ cv2.getRotationMatrix2D(center, angle, scale) -> retval

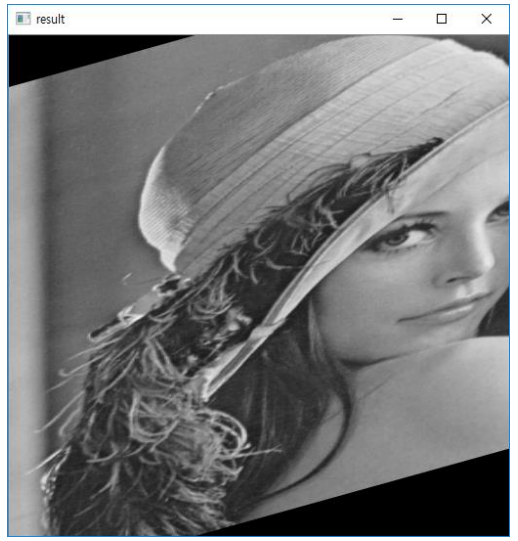
- **center** : 회전 중심 좌표. (x, y) 튜플
- **angle** : (반시계 방향) 회전 각도(degree). 음수는 시계 방향
- **scale** : 추가적인 확대 비율
- **retval** : 2x3 affine변환 행렬. 실수형

❖ cv2.invertAffineTransform(M, [,iM]) -> iM

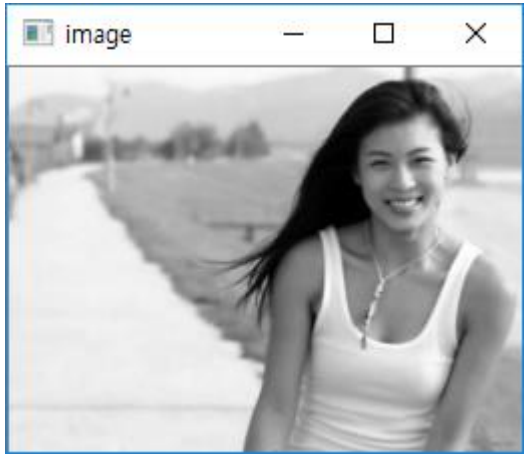
- **M** : Affine 변환 행렬
- **iM** : Affine 역변환 행렬

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

def geometricTransform():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape[0:2]

    pts1 = np.float32([[50,50], [50,100], [100,75]])
    pts2 = np.float32([[10,100], [10,150], [100,100]])
    M = cv2.getAffineTransform(pts1, pts2)
    result = cv2.warpAffine(img, M, (width,height))

    cv2.imshow('image', img)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

geometricTransform()
```





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

def geometricTransform():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape[0:2]

    M = cv2.getRotationMatrix2D((width/2, height/2), -45, 1)

    result = cv2.warpAffine(img, M, (width,height))

    cv2.imshow('image', img)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

geometricTransform()
```



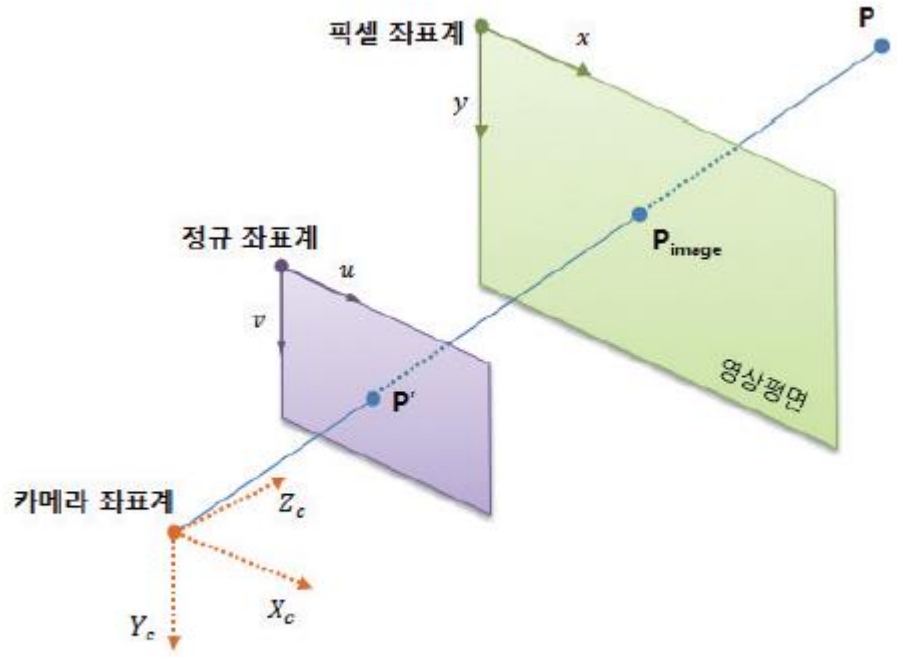
5

원근투시변환

원근 투시 변환

❖ 원근 투시 변환(perspective projection transformation)

- 원근법을 영상 좌표계에서 표현하는 것
- 3차원의 실세계 좌표를 투영 스크린상의 2차원 좌표로 표현할 수 있도록 변환해 주는것



〈그림 8.7.2〉 3차원의 실세계 좌표를 2차원 좌표계에 표현 방법

$$w \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

참고: 동차좌표와 동차행렬

❖ 동차 좌표와 동차 행렬

- 동차 좌표
  - $\dot{\mathbf{x}} = (y\ x\ 1)$
  - (3,5) → (3,5,1), (6,10,2), (0.3,0.5,0.1), …
- 동차 행렬

변환	동차 행렬 $\dot{\mathbf{H}}$	설명
이동	$T(t_y, t_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_y & t_x & 1 \end{pmatrix}$	y방향으로 t_y, x방향으로 t_x만큼 이동
회전	$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$	원점을 중심으로 시계방향으로 θ만큼 회전
크기	$S(s_y, s_x) = \begin{pmatrix} s_y & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & 1 \end{pmatrix}$	y방향으로 s_y, x방향으로 s_x만큼 확대
기울임	$Sh_y(h_y) = \begin{pmatrix} 1 & 0 & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $Sh_x(h_x) = \begin{pmatrix} 1 & h_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	Sh_y : y방향으로 h_y만큼 기울임 Sh_x : x방향으로 h_x만큼 기울임

❖ 왜 동차 좌표를 사용하나?

- 복합 변환을 이용한 계산 효율
  - (예) 이동 후 회전은 두 번의 행렬 곱셈, 그러나 복합 변환을 이용하면 한 번의 곱셈으로 가능

cv2.warpPerspective

❖ **cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) -> dst**

- **src** : 입력 영상
- **M** : 2x3 투시변환 행렬. 실수형.
- **dsize** : 결과 영상 크기. (w, h) 튜플. (0, 0)이면 src와 같은 크기로 설정
- **dst** : 출력 영상
- **flags** : 보간법. 기본값은 cv2.INTER_LINEAR
- **borderMode** : 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_CONSTANT
- **borderValue** : cv2.BORDER_CONSTANT일 때 사용할 상수 값. 기본값은 0(검정색)

❖ **cv2.getPerspectiveTransform(src, dst, solveMethod=None) -> retval**

- src : 4개의 원본 좌표점. numpy.ndarray. shape=(4, 2)
  - (예) np.array([[x1 , y1], [x2 , y2], [x3 , y3], [x4 , y4]], np.float32)
- dst : 4개의 결과 좌표점. numpy.ndarray. shape=(4, 2)
- retval : 3x3 투시 변환 행렬

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

def geometricTransform():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    height, width = img.shape[0:2]

    pts1 = np.float32([[0,0], [300,0], [0, 300], [300,300]])
    pts2 = np.float32([[50,50], [350,50], [30, 380], [390,390]])
    M = cv2.getPerspectiveTransform(pts1, pts2)
    result = cv2.warpPerspective(img, M, (width,height))

    cv2.imshow('image', img)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

geometricTransform()
```