CHAPTER 08
기하학 처리

PART 02 영상 처리와 OpenCV 함수 활용

contents

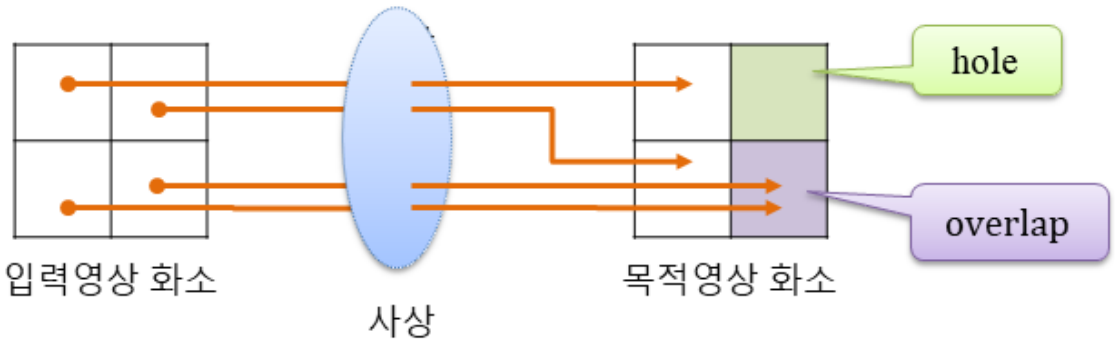
- 8.1 사상
- 8.2 크기 변경(확대/축소)
- 8.3 보간
- 8.4 평행이동
- 8.5 회전
- 8.6 행렬 연산을 통한 기하학 변환 - 어파인 변환
- 8.7 원근 투시(투영) 변환

8.1 사상(mapping)

- 기하학적 처리의 기본
  - 화소들의 배치 변경 →사상의 의미 이해

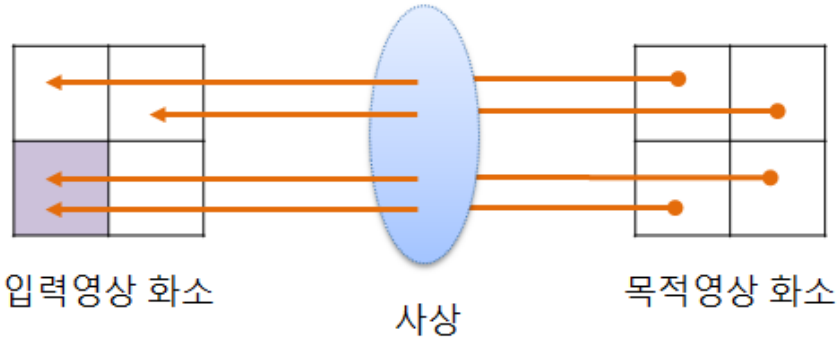
- 사상
  - 화소들의 배치를 변경할 때, 입력영상의 좌표에 해당하는 해당 목적영상의 좌표를 찾아서 화소값을 옮기는 과정

- 순방향 사상(forward mapping)
  - 원본영상의 좌표를 중심으로 목적영상의 좌표를 계산하여 화소의 위치를 변환하는 방식
  - 홀(hole)이나 오버랩(overlap)의 문제가 발생



8.1 사상

- 역방향 사상 (backward mapping)
  - 목적영상의 좌표를 중심으로 역변환을 계산하여 해당하는 입력 영상의 좌표를 찾아서 화소값을 가져오는 방식
  - 홀이나 오버랩은 발생하지 않음
  - 입력영상의 한 화소를 목적영상의 여러 화소에서 사용하게 되면 결과 영상의 품질 저하



8.2 크기 변경 (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

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

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

8.2 크기 변경 (확대/축소)

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
```

변경될 목적영상의 크기를 이용해서 크기 변경을 수행하는 함수

목적영상 좌표

원본영상 좌표

8.2 크기 변경 (확대/축소)

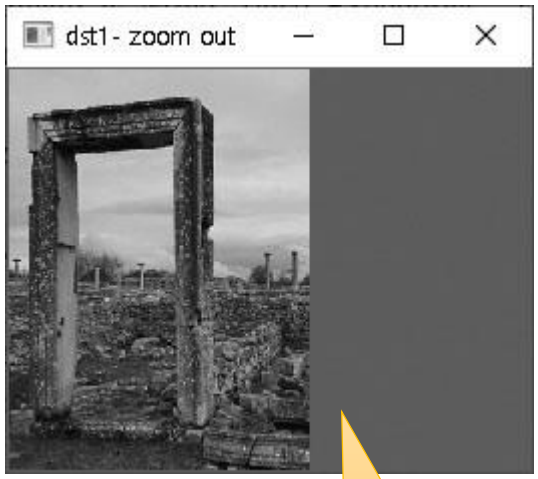
```
22  def time_check(func, image, size, title):              # 수행시간 체크 함수
23      start_time = time.perf_counter()
24      ret_img = func(image, size)
25      elapsed = (time.perf_counter() - start_time) * 1000
26      print(title, "수행시간 = %0.2f ms" % elapsed)
27      return ret_img
28
29  image = 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)
```

수행시간 체크할 함수명

영상이 작아서 윈도우 크기 확대

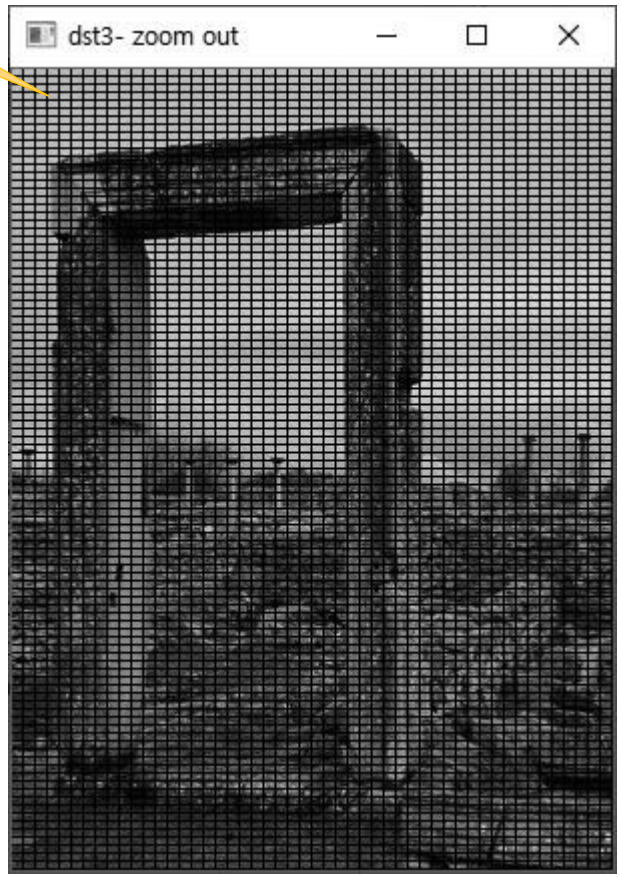
8.2 크기 변경 (확대/축소)

- 실행결과



윈도우 확대

홀의 발생으로 결과영상 품질 저하

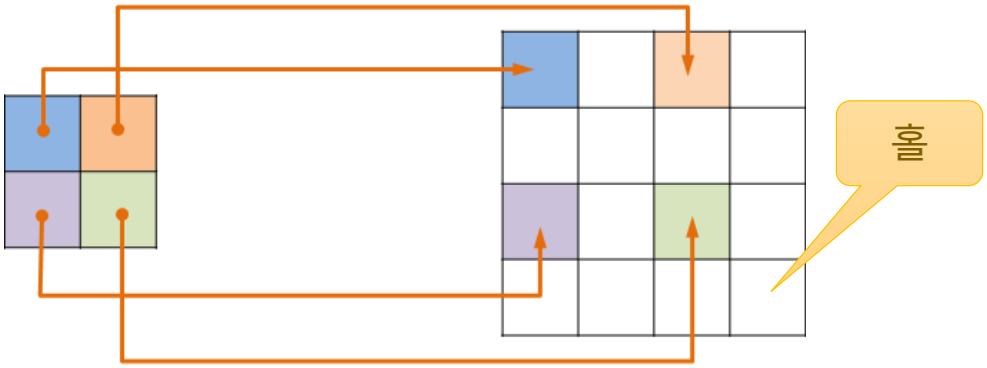


8.3 보간

- 8.3.1 최근접 이웃 보간법
  8.3.2 양선형 보간법

8.3 보간(interpolation)

- 순방향 사상으로 영상 확대 - 홀이 많이 발생



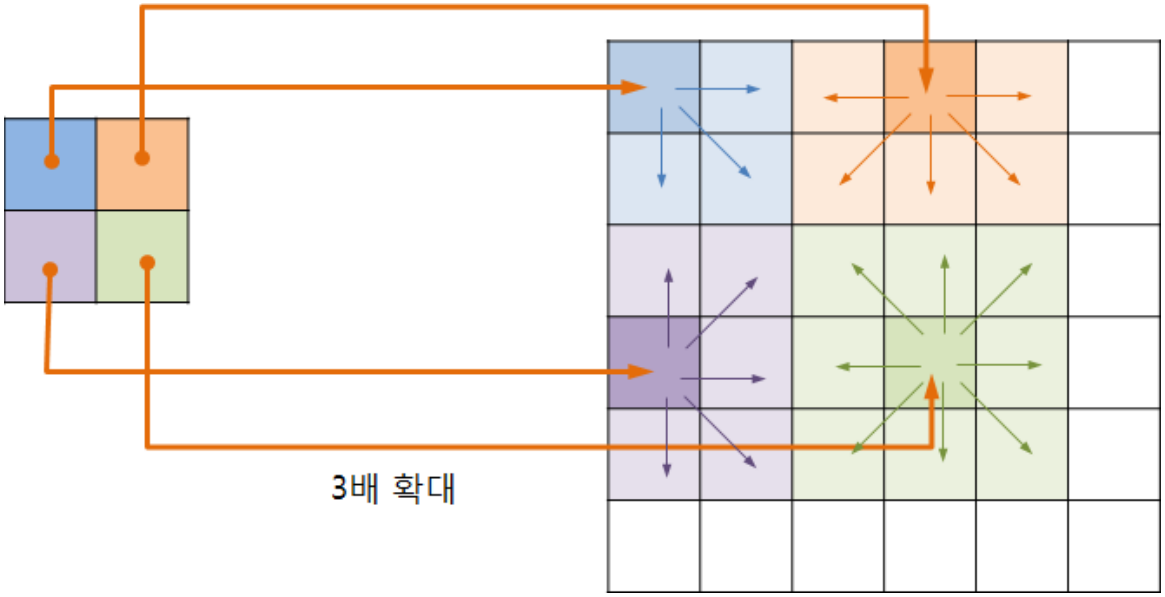
2배 확대

- 역방향 사상을 통해서 홀의 화소들을 입력영상에서 찾음
  - 영상을 축소할 때에는 오버랩의 문제가 발생

- 보간법 필요
  - 목적영상에서 홀의 화소들을 채우고, 오버랩이 되지 않게 화소들을 배치하여 목적영상을 만드는 기법

8.3.1 최근접 이웃 보간법(nearest neighbor interpolation)

- 목적영상을 만드는 과정에서 홀이 되어 할당 받지 못하는 화소들의 값을 찾을 때, 목적영상의 화소에 가장 가깝게 이웃한 입력영상의 화소값을 가져오는 방법



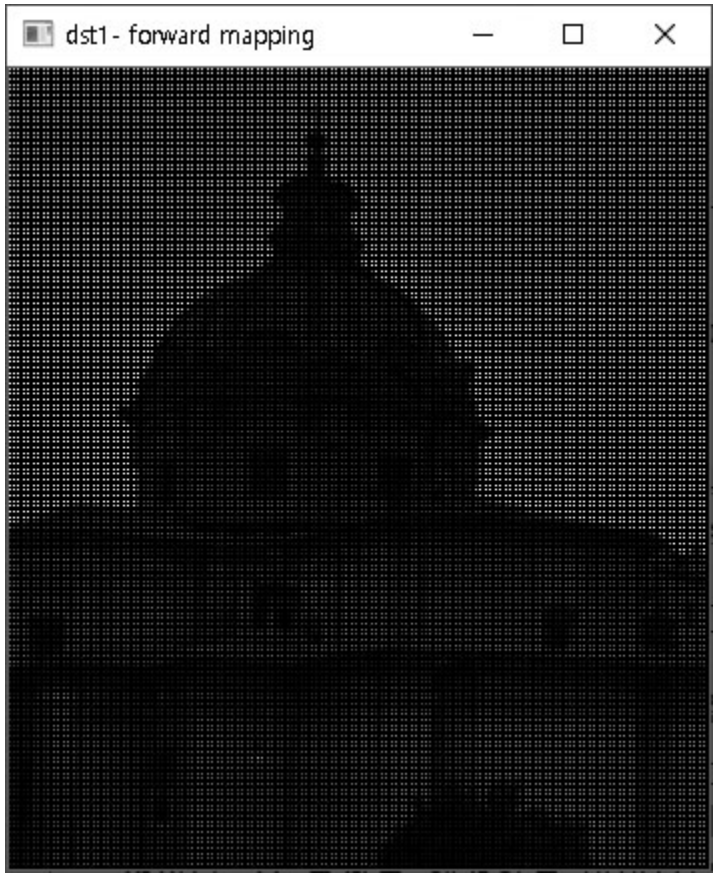
$$\begin{aligned} y' &= y \cdot ratio\ Y \\ x' &= x \cdot ratio\ X \end{aligned} \quad \Rightarrow \quad y = \frac{y'}{ratio\ Y},\ x = \frac{x'}{ratio\ X}$$

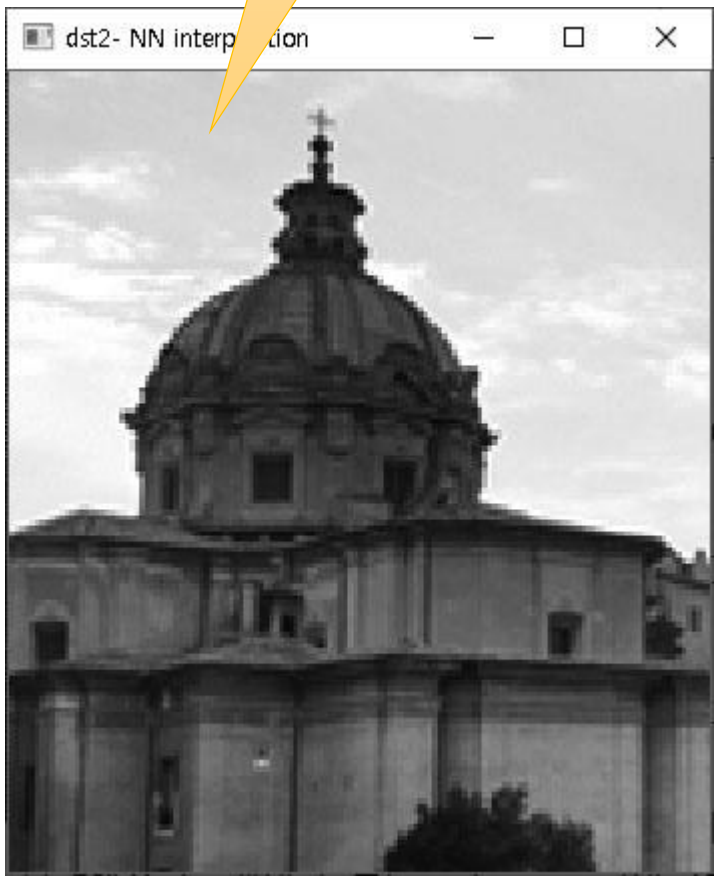
8.3.1 최근접 이웃 보간법

예제 8.3.1 크기변경 & 최근접 이웃 보간- 02.scaling_nearset.cpp

```
01  import numpy as np, cv2
02  from Common.interpolation import scaling     # interpolation 모듈의 scaling() 함수 임포트
03
04  def scaling_nearest(img, size):                          # 크기 변경 함수3
05      dst = np.zeros(size[::-1], img.dtype)                # 행렬과 크기는 원소가 역순
06      ratioY, ratioX = np.divide(size[::-1], img.shape[:2]) # 변경 크기 비율
07      i = np.arange(0, size[1], 1)                         # 목적 영상 세로(i) 좌표 생성
08      j = np.arange(0, size[0], 1)                         # 목적 영상 가로(j) 좌표 생성
09      i, j = np.meshgrid(i, j)
10      y, x = np.int32(i / ratioY), np.int32(j / ratioX)    # 입력 영상 좌표
11      dst[i, j] = img[y, x]                                # 역방향 사상 → 입력 영상 좌표 계산
12
13      return dst
14
15  image = cv2.imread("images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
16  if image is None: raise Exception("영상파일 읽기 에러")
17
18  dst1 = scaling(image, (350, 400))                        # 크기 변경- 기본
19  dst2 = scaling_nearest(image, (350, 400))                # 크기 변경- 최근접 이웃 보간
20
21  cv2.imshow("image", image)
22  cv2.imshow("dst1- forward mapping", dst1)                # 순방향 사상
23  cv2.imshow("dst2- NN interpolation", dst2)               # 최근접 이웃 보간
24  cv2.waitKey(0)
```

8.3.1 최근접 이웃 보간법

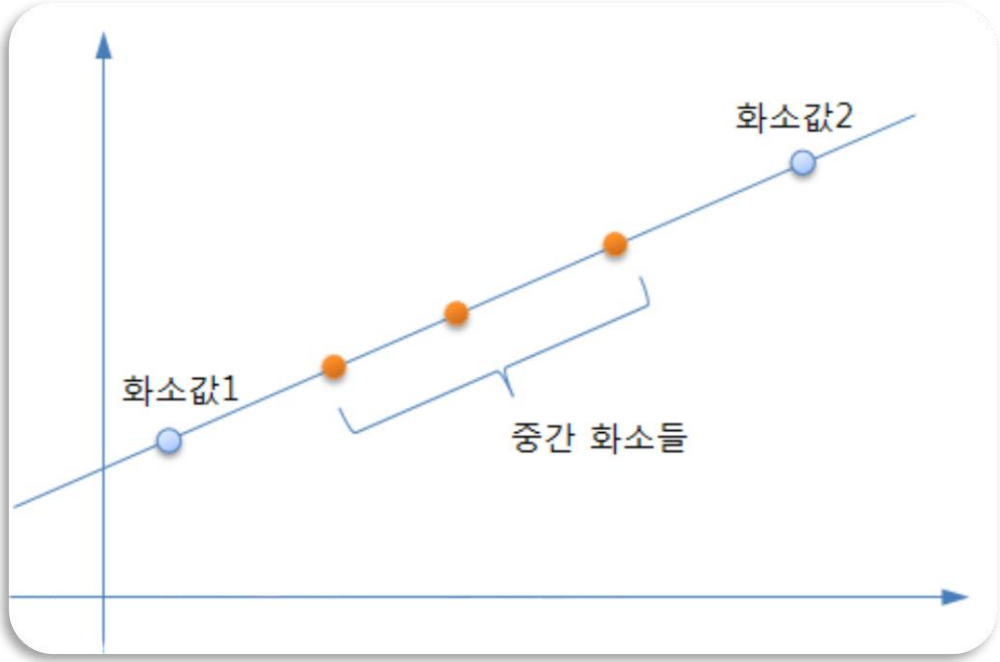




8.3.2 양선형 보간법 (bilinear interpolation)

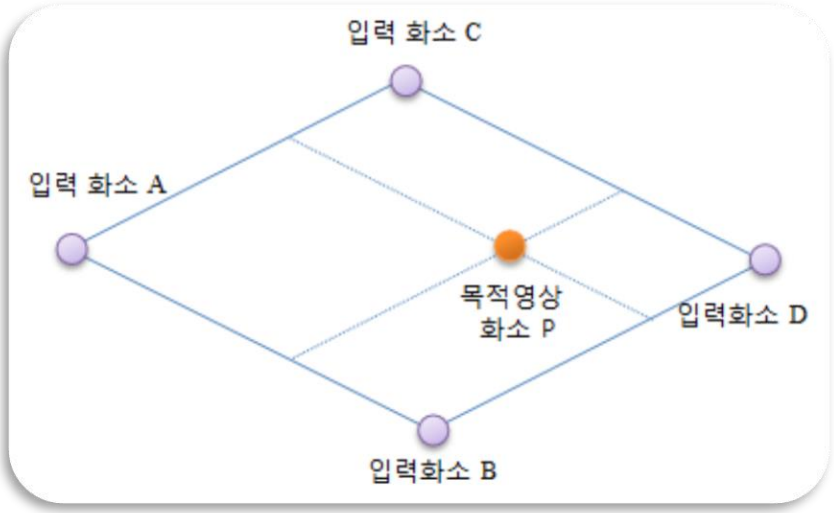
- 최근접 이웃 보간법
  - 확대비율이 커지면, 모자이크 현상 혹은 경계부분에서 계단현상 발생

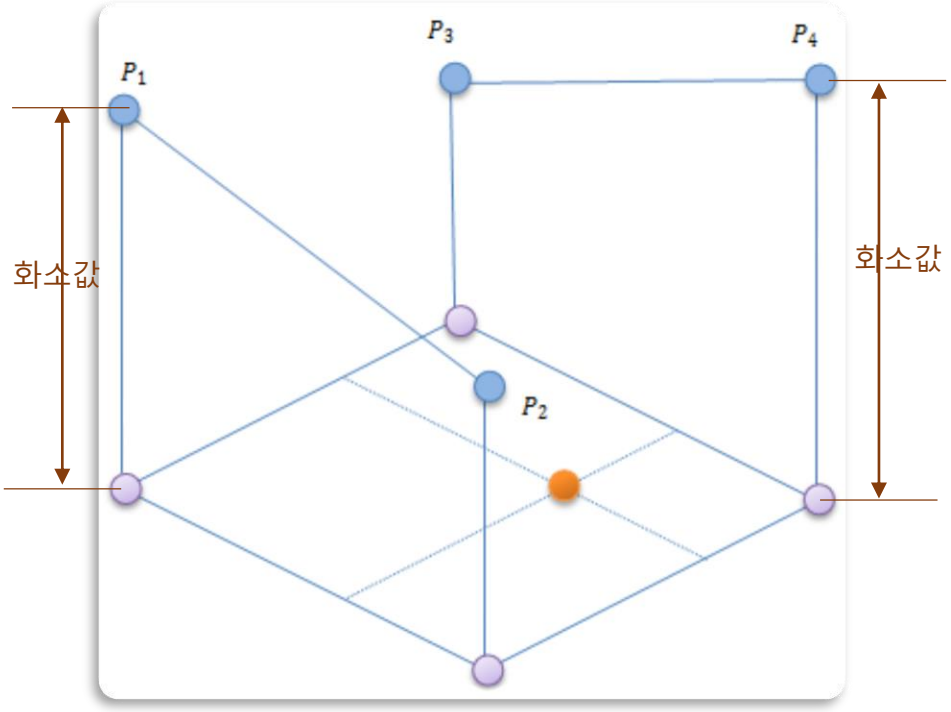
- 양선형 보간으로 해결
  - 직선의 선상에 위치한 중간 화소들의 값은 직선의 수식을 이용해서 쉽게 계산



8.3.2 양선형 보간법 (bilinear interpolation)

- 양선형 보간법 - 선형 보간을 두 번에 걸쳐서 수행하기에 붙여진 이름
  - 목적영상 화소(P)를 역변환으로 계산하여 가장 가까운 위치에 있는 입력영상의 4개 화소(A, B, C, D) 값 가져옴
  - 4개 화소를 두 개씩(AB, CD) 묶어서 두 화소를 화소값(P_1, P_2, P_3, P_4)으로 잇는 직선 구성

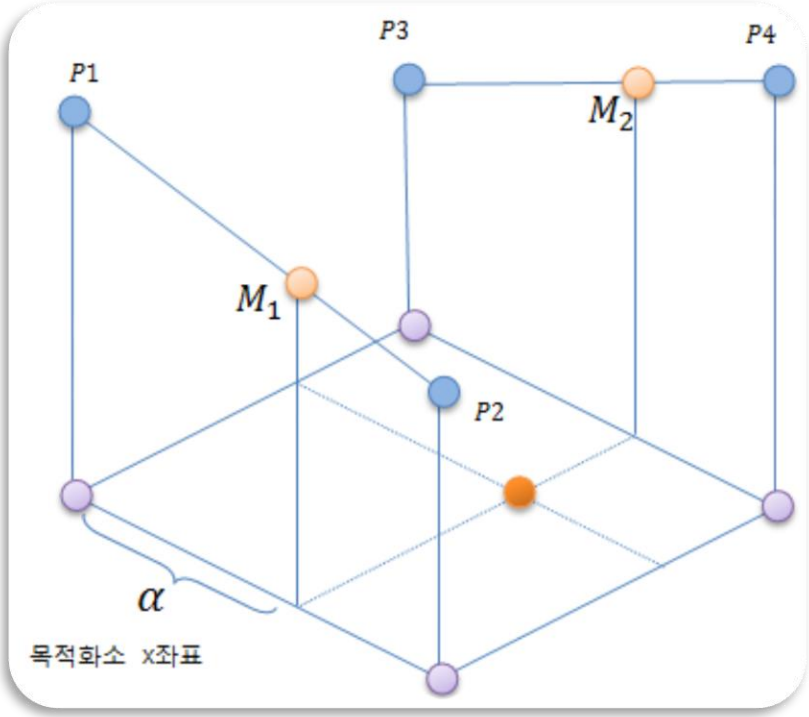


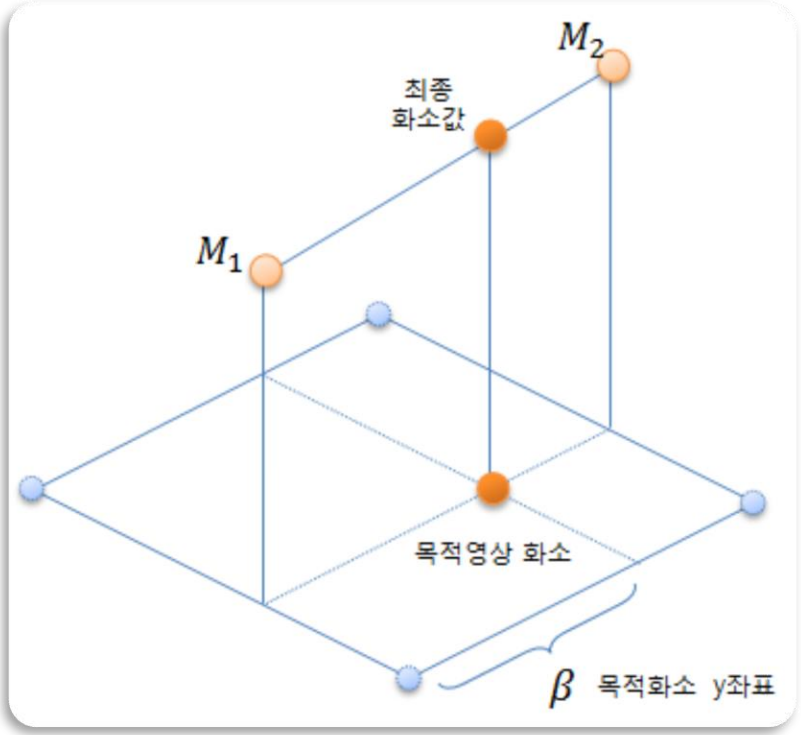


8.3.2 양선형 보간법 (bilinear interpolation)

- 직선상에서 목적영상 화소의 좌표로 중간 위치 찾음
  - 중간 위치는 거리 비율(α , 1- α)로 찾음
- 중간 위치 화소값(M_1 , M_2) 계산
  - 입력 화소값과 거리 비율(α , 1- α)로 직선 수식 이용해 계산

- 두 개의 중간 화소값(M_1 , M_2)을 잇는 직선 다시 구성
- 두 개의 중간 화소값과 거리 비율(b , 1-b)로 직선 수식을 이용해서 최종 화소값 계산





8.3.2 양선형 보간법

- 수식으로 정리

$$M_1 = \alpha \cdot B + (1-\alpha) \cdot A = A + \alpha \cdot (B-A)$$
$$M_2 = \alpha \cdot D + (1-\alpha) \cdot C = C + \alpha \cdot (D-C)$$

$$P = \beta \cdot M_2 + (1-\beta) \cdot M_1 = M_1 + \beta \cdot (M_2 - M_1)$$

- OpenCV 보간 옵션
  - cv::resize(), cv::remap(), cv::warpAffine(), cv::warpPerspective() 등의 함수에서 사용

〈표 8.3.1〉 보간 방법에 대한 flag 옵션

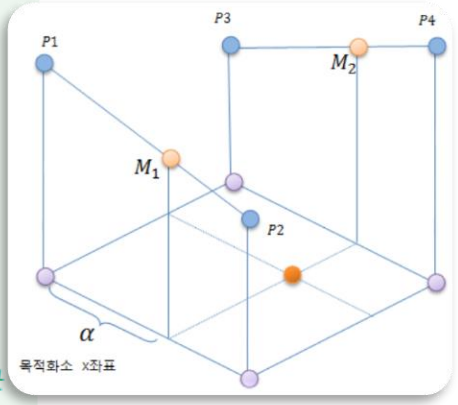
옵션 상수	값	설명
INTER_NEAREST	0	최근접 이웃 보간
INTER_LINEAR	1	양선형 보간 (기본값)
INTER_CUBIC	2	바이큐빅 보간 – 4x4 이웃 화소 이용
INTER_AREA	3	픽셀 영역의 관계로 리샘플링
INTER_LANCZOS4	4	Lanczos 보간 – 8x8 이웃 화소 이용

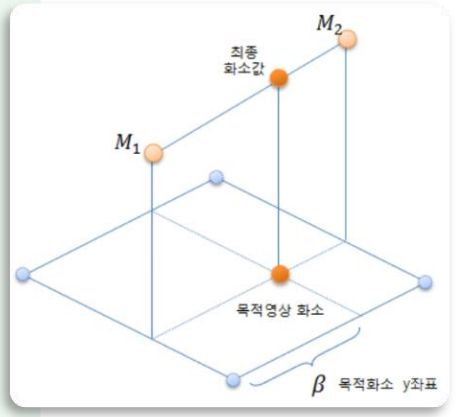
8.3.2 양선형 보간법

예제 8.3.2 크기변경 & 양선형 보간 – 03.scaling_bilinear.cpp

```
01  import numpy as np, cv2
02  from Common.interpolation import scaling_nearest      # 최근접 이웃 보간 함수 임포트
03
04  def bilinear_value(img, pt):                          # 단일 화소 양선형 보간 수행 함수
05      x, y = np.int32(pt)
06      if x >= img.shape[1]-1: x = x - 1                 # 영상 범위 벗어남 처리
07      if y >= img.shape[0]-1: y = y - 1
08
09      P1, P2, P3, P4 = np.float32(img[y:y+2,x:x+2].flatten())  # 4개 화소-관심 영역으로 접근
10      ## 4개 화소 - 화소 직접 접근
11      # P1 = float(img[y, x] )                          # 좌상단 화소
12      # P2 = float(img[y + 0, x + 1])                   # 우상단 화소
13      # P3 = float(img[y + 1, x + 0])                   # 좌하단 화소
14      # P4 = float(img[y + 1, x + 1])                   # 우하단 화소
15
16      alpha, beta = pt[1] - y, pt[0] - x                # 거리 비율
17      M1 = P1 + alpha * (P3 - P1)                       # 1차 보간
18      M2 = P2 + alpha * (P4 - P2)
19      P  = M1 + beta * (M2 - M1)                        # 2차 보간
20      return np.clip(P, 0, 255)                         # 화소값 saturation 후 반환
21
22  def scaling_bilinear(img, size):                      # 양선형 보간
23      ratioY, ratioX = np.divide(size[::-1], img.shape[:2]) # 변경 크기 비율-
24
25      dst = [[ bilinear_value(img, (j/ratioX, i/ratioY))  # 리스트 생성
26                  for j in range(size[0])]
27                for i in range(size[1])]
28      return np.array(dst, img.dtype)
```

단일 화소 양선형 보간





$$M_1 = \alpha \cdot B + (1-\alpha) \cdot A = A + \alpha \cdot (B-A)$$
$$M_2 = \alpha \cdot D + (1-\alpha) \cdot C = C + \alpha \cdot (D-C)$$

$$P = \beta \cdot M_2 + (1-\beta) \cdot M_1 = M_1 + \beta \cdot (M_2 - M_1)$$

8.3.2 양선형 보간법

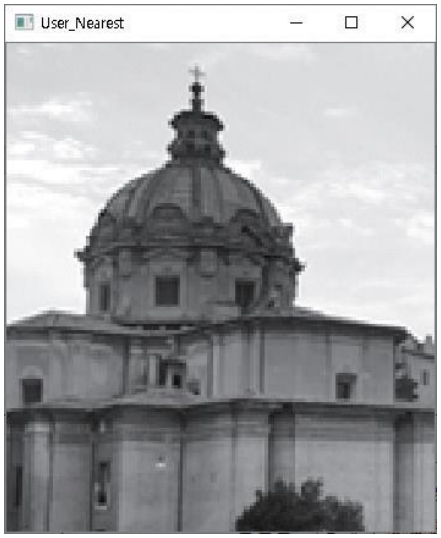
```
30  image = cv2.imread("images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
31  if image is None: raise Exception("영상파일 읽기 에러")
32
33  size = (350, 400)
34  dst1 = scaling_bilinear(image, size)                    # 크기 변경– 양선형 보간
35  dst2 = scaling_nearest(image, size)                     # 크기 변경– 최근접 이웃 보간
36  dst3 = cv2.resize(image, size, 0, 0, cv2.INTER_LINEAR)  # OpenCV 함수 – 양선형
37  dst4 = cv2.resize(image, size, 0, 0, cv2.INTER_NEAREST) # OpenCV 함수 – 최근접
38
39  cv2.imshow("image", image)
40  cv2.imshow("User_bilinear", dst1);
41  cv2.imshow("User_Nearest", dst2)
42  cv2.imshow("OpenCV_bilinear", dst3);
43  cv2.imshow("OpenCV_Nearest", dst4)
44  cv2.waitKey(0)
```

8.3.2 양선형 보간법

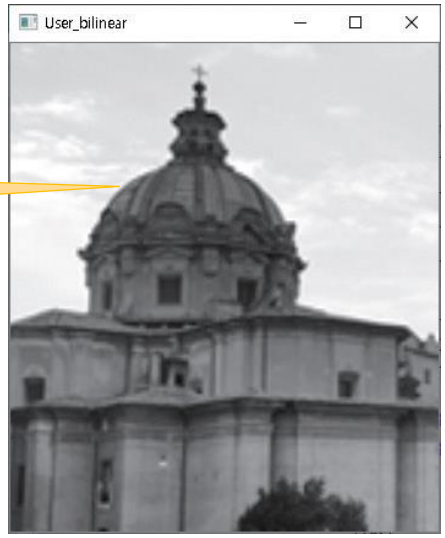
- 실행결과

입력영상

확대

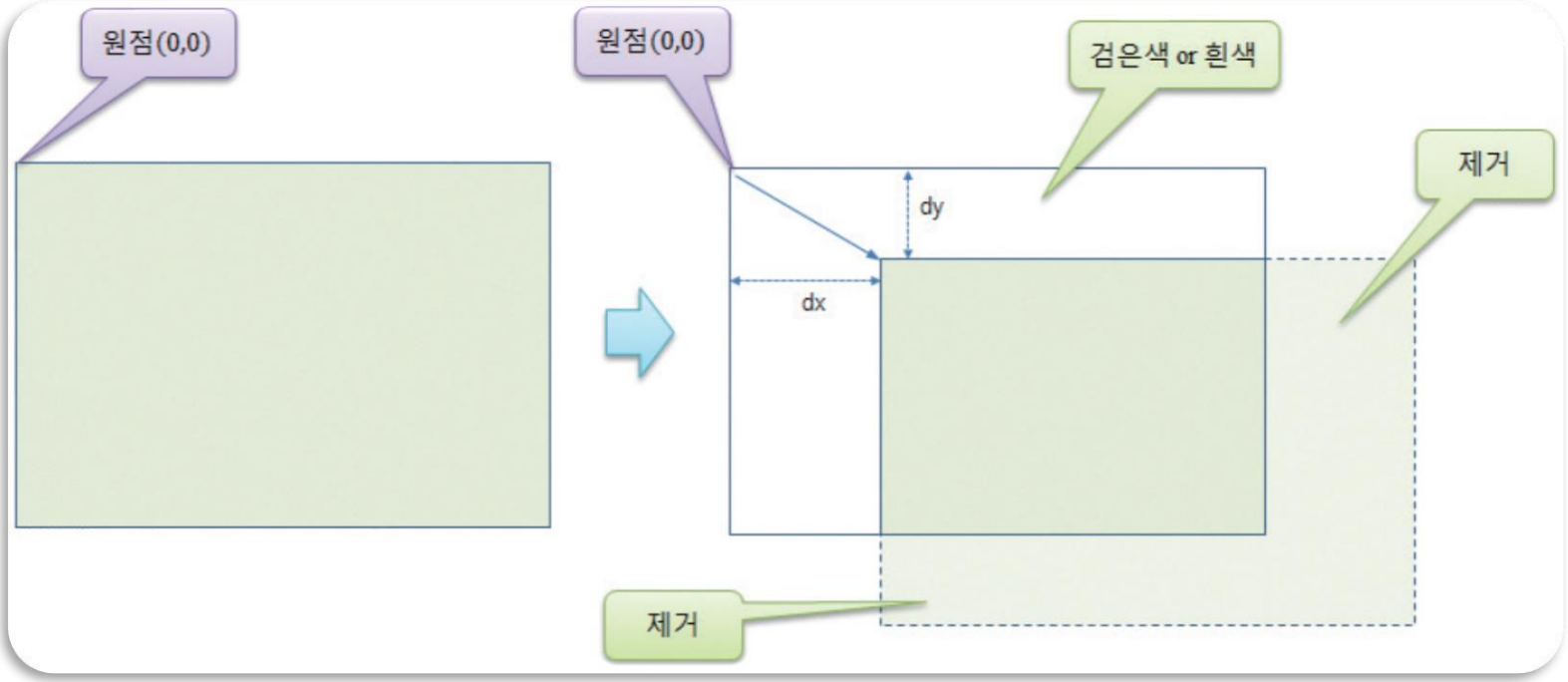


계산 현상의 감소



8.4 평행이동

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



순방향 사상

$$x' = x + dx$$
$$y' = y + dy$$

역방향사상

$$x = x' - dx$$
$$y = y' - dy$$

8.4 평행이동

예제 8.4.1 영상 평행이동 – 04.translation.cpp

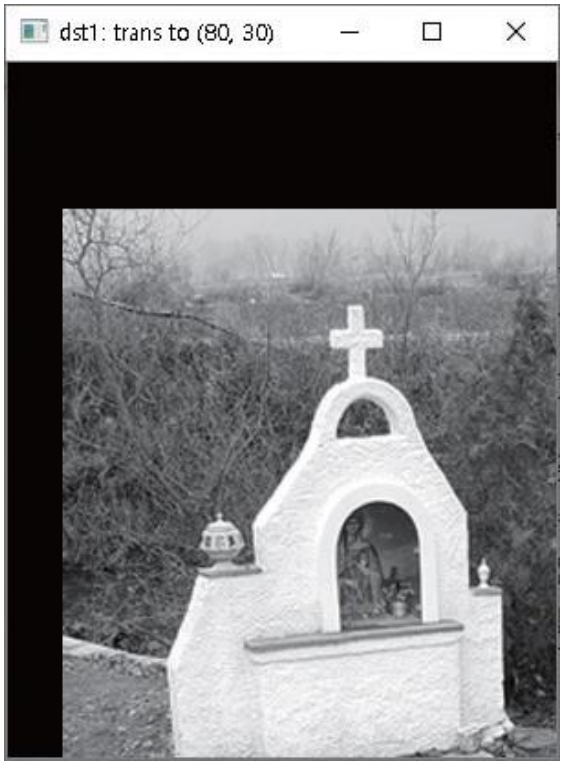
```
01  import numpy as np, cv2
02
03  def contain(p, shape):                          # 좌표(y,x)가 범위내 인지 검사
04      return 0<= p[0] < shape[0] and 0<= p[1] < shape[1]
05
06  def translate(img, pt):
07      dst = np.zeros(img.shape, img.dtype)        # 목적 영상 생성
08      for i in range(img.shape[0]):               # 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)
```

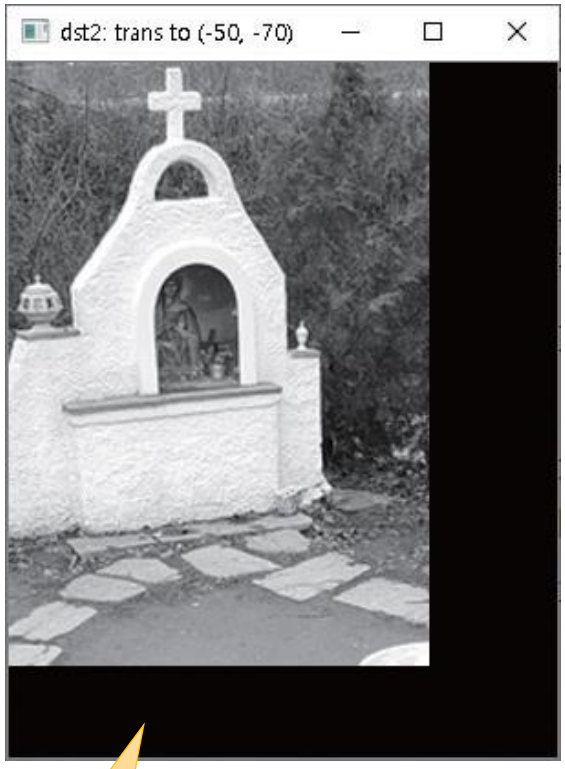
입력영상 좌표 계산

계산 화소가 입력영상 범위내에 있어야 됨

8.4 평행이동

- 실행결과





입력영상 범위 벗어나면 검은색 처리

8.5 회전

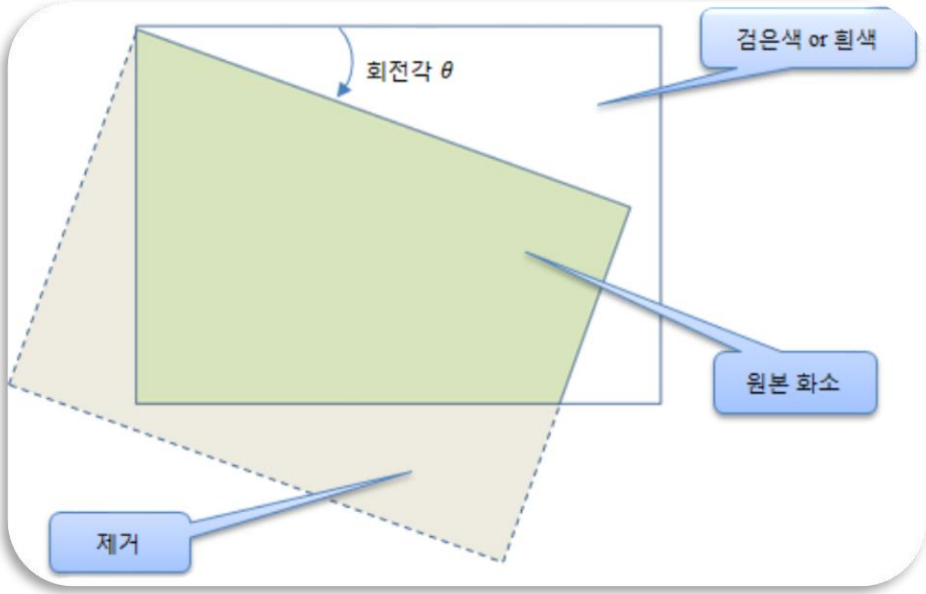
- 입력영상의 모든 화소를 영상의 원점을 기준으로 원하는 각도만큼 모든 화소에 대해서 회전 변환을 시키는 것
- 회전 변환 수식

순방향 사상

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

역방향사상

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



- 원점으로부터 시계 방향으로 정해진 각도만큼 회전된 영상 생성
  - 직교 좌표계에서 회전 변환은 반시계 방향으로 적용
  - 영상 좌표계에서는 y 좌표가 하단으로 내려갈수록 증가하기 때문에 시계방향 회전

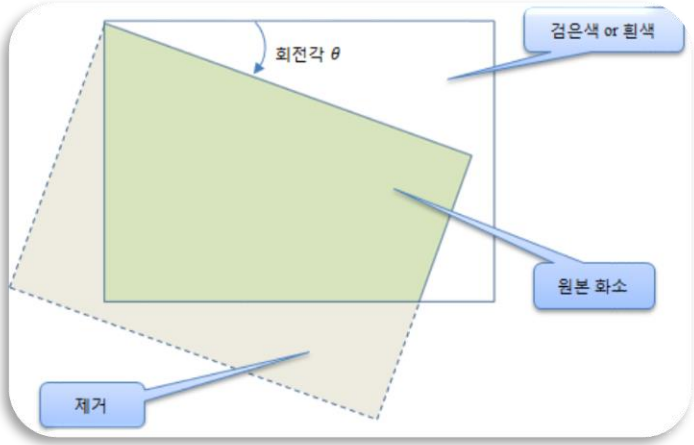
8.5 회전

- 특정 좌표에서(center X, center Y) 회전하는 경우
  - 회전의 기준점으로 영상을 이동시킨 후, 회전 수행
  - 다시 원점 좌표로 이동

$$x=(x'-center\ X)\cos\theta+(y'-center\ Y)\sin\theta+center\ X$$
$$y=-(x'-center\ X)\sin\theta+(y'-center\ Y)\cos\theta+center\ Y$$

기준점 이동

원점 재이동



8.5 회전

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
        sin, cos = np.sin(radian), np.cos(radian)          # 사인, 코사인 값 미리 계산
10      for i in range(img.shape[0]):                      # 목적 영상 순회– 역방향 사상
11          for j in range(img.shape[1]):
                y = -j * sin + i * cos
                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
```

반복문 내에서 계산시 속도 저하

역방향 사상 수식

8.5 회전

회전 기준점

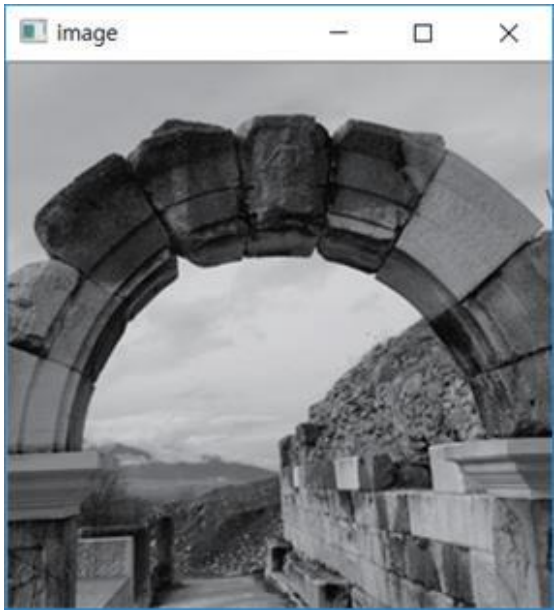
```
18  def rotate_pt(img, degree, pt):                    # pt 기준 회전 변환 함수
19      dst = np.zeros(img.shape[:2], img.dtype)       # 목적 영상 생성
20      radian = (degree/180) * np.pi                  # 회전 각도– 라디언
21      sin, cos = math.sin(radian), math.cos(radian)  # 사인, 코사인 값 미리 계산
22
23      for i in range(img.shape[0]):                  # 목적 영상 순회– 역방향 사상
24          for j in range(img.shape[1]):
25              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
28              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")
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)
```

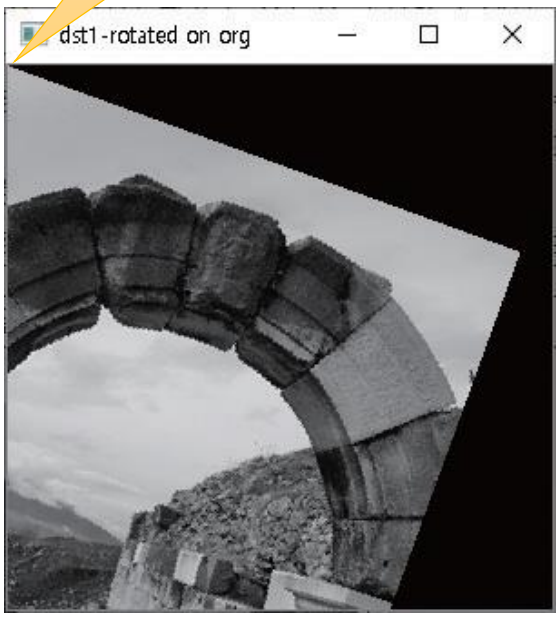
회전 기준점으로 평행이동

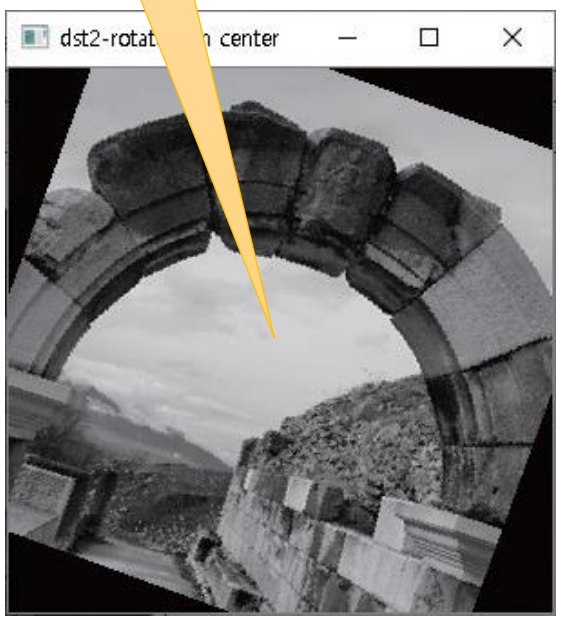
회전후 역 평행이동

8.5 회전

- 실행결과







8.6 행렬 연산을 통한 기하학 변환–어파인 변환

- 기하학 변환 수식이 행렬의 곱으로 표현 가능
  - 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

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

  - 평행이동

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

  - 어파인 변환 수식

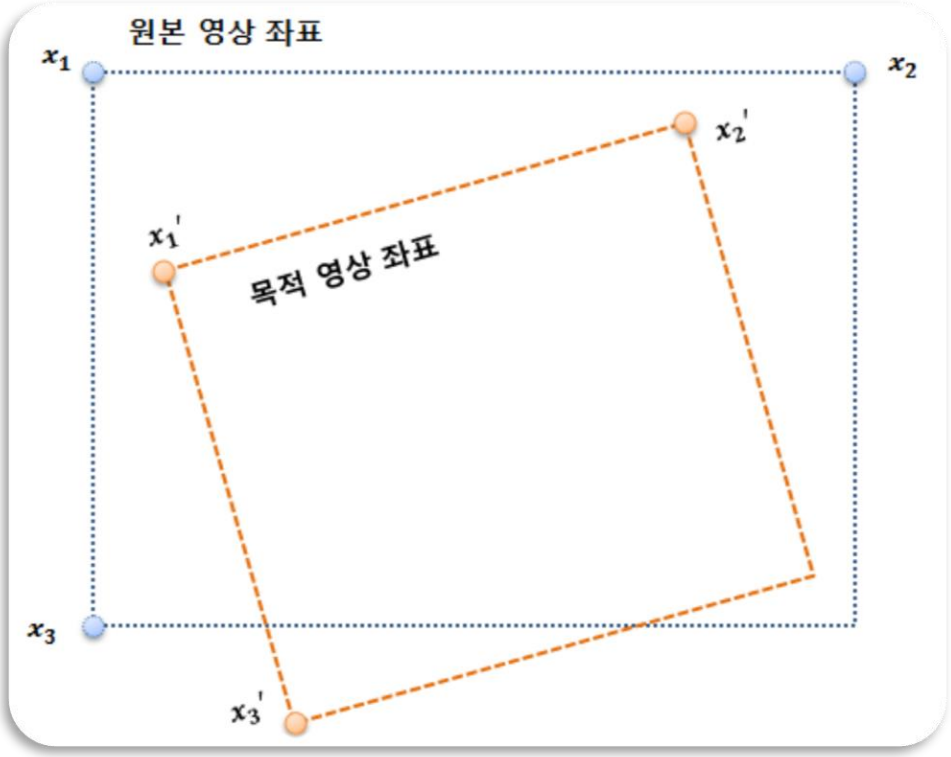
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  - 각 변환 행렬을 행렬 곱으로 구성가능 → 최종 행렬곱 후에 마지막 행 삭제

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

8.6 행렬 연산을 통한 기하학 변환–어파인 변환

- 입력영상의 좌표 3개(x_1 , x_2 , x_3)와 변환이 완료된 목적영상에서 상응하는 좌표 3개(x_1, x_2, x_3)를 알면 → 어파인 행렬 구성 가능



8.6 행렬 연산을 통한 기하학 변환–어파인 변환

- OpenCV에서도 어파인 변환 수행 함수 제공

함수 및 인수 구조		
cv2.warpAffine (src, M, dsize [, dst [, flags [, borderMode [, borderValue]]]]) → dst ■ 설명: 입력 영상에 어파인 변환을 수행해서 반환한다.		
인수 설명	■ src	입력 영상
	■ dst	반환 영상
	■ M	어파인 변환 행렬
	■ dsize	반환 영상의 크기
	■ flags	보간 방법
	■ borderMode	경계지정 방법
cv2.getAffineTransform(src, dst) → retval ■ 설명: 3개의 좌표쌍을 입력하면 어파인 변환 행렬을 반환한다.		
인수 설명	■ src	입력 영상 좌표 3개 (행렬로 구성)
	■ dst	목적 영상 좌표 3개 (행렬로 구성)
cv2.getRotationMatrix2D(center, angle, scale) → retval ■ 설명: 회전 변환과 크기 변경을 수행할 수 있는 어파인 행렬을 반환한다.		
인수 설명	■ center	회전의 중심점
	■ angle	회전각도, 양수 각도가 반시계 방향 회전 수행
	■ scale	변경할 크기
cv2.invertAffineTransform(M [, iM]) → iM ■ 설명: 어파인 변환 행렬의 역 행렬을 반환한다.		
인수 설명	■ M	어파인 변환 행렬
	■ iM	어파인 역변환 행렬

8.6 행렬 연산을 통한 기하학 변환–어파인 변환

심화예제 8.6.2 어파인 변환의 연결 – 07.affine_combination.py

```
01  import numpy as np, math, cv2
02  from Common.interpolation import affine_transform     # 저자 구현 어파인변환 함수 임포트
03
04  def getAffineMat(center, degree, fx=1, fy=1, translate=(0,0)):    # 변환 행렬 합성 함수
05      scale_mat = np.eye(3, dtype=np.float32)            # 크기 변경 행렬
06      cen_trans = np.eye(3, dtype=np.float32)            # 중점 평행 이동
07      org_trans = np.eye(3, dtype=np.float32)            # 원점 평행 이동
08      trans_mat = np.eye(3, dtype=np.float32)            # 좌표 평행 이동
09      rot_mat   = np.eye(3, dtype=np.float32)            # 회전 변환 행렬
10
11      radian = degree / 180 * np.pi                      # 회전 각도– 라디언 계산
12      rot_mat[0] = [ np.cos(radian), np.sin(radian), 0]  # 회전행렬 0행
13      rot_mat[1] = [-np.sin(radian), np.cos(radian), 0]  # 회전행렬 1행
14
15      cen_trans[:2, 2] = center                          # 중심 좌표 이동
16      org_trans[:2, 2] = -center[0], -center[1]          # 원점으로 이동
17      trans_mat[:2, 2] = translate                       # 평행 이동 행렬의 원소 지정
18      scale_mat[0, 0], scale_mat[1, 1] = fx, fy          # 크기 변경 행렬의 원소 지정
19
20      ret_mat = cen_trans.dot(rot_mat.dot(trans_mat.dot(scale_mat.dot(org_trans))))
21      # ret_mat = cen_trans.dot(rot_mat.dot(scale_mat.dot(trans_mat.dot(org_trans))))
22      return np.delete(ret_mat, 2, axis=0)               # 마지막행 제거 ret_mat[0:2,:]
23
```

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

각 변환 행렬의 곱으로 최종 변환 행렬 계산

행렬 곱의 순서에 따라 최종 목적행렬 달라짐

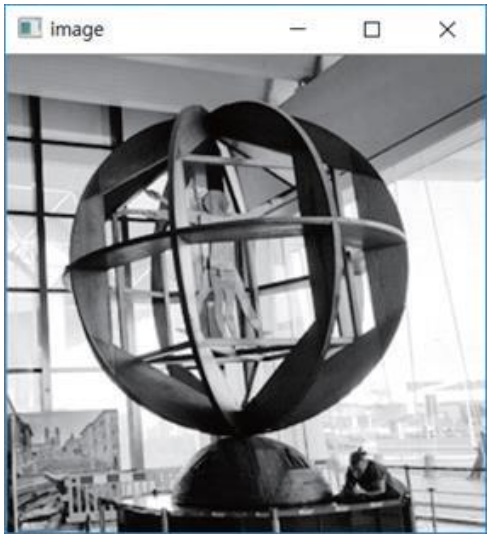
3행, 3열 행렬에서 2행, 3열 어파인 행렬

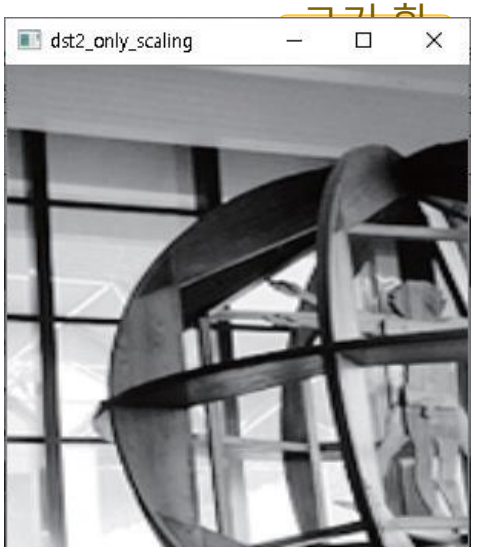
8.6 행렬 연산을 통한 기하학 변환–어파인 변환

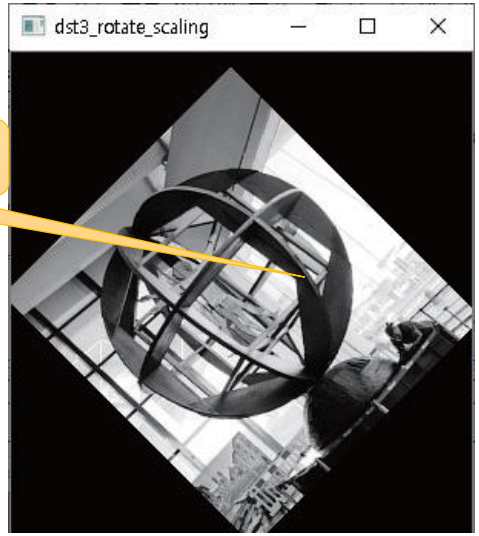
```
24  image = cv2.imread("images/affine2.jpg", cv2.IMREAD_GRAYSCALE)
25  if image is None: raise Exception("영상파일 읽기 에러")
26
27  size = image.shape[::-1]
28  center = np.divmod(size, 2)[0]                          # 회전 중심 좌표
29  angle, tr = 45, (200, 0)                                # 각도와 평행이동 값 지정
30
31  aff_mat1 = getAffineMat(center, angle)                  # 중심 좌표 기준 회전
32  aff_mat2 = getAffineMat((0,0), 0, 2.0, 1.5)             # 크기 변경– 확대
33  aff_mat3 = getAffineMat(center, angle, 0.7, 0.7)        # 회전 및 축소
34  aff_mat4 = getAffineMat(center, angle, 0.7, 0.7, tr)    # 복합 변환
35
36  dst1 = cv2.warpAffine(image, aff_mat1, size)            # OpenCV 함수
37  dst2 = cv2.warpAffine(image, aff_mat2, size)
38  dst3 = affine_transform(image, aff_mat3)      # 사용자 정의 함수
39  dst4 = affine_transform(image, aff_mat4)
40
41  cv2.imshow("image", image)
42  cv2.imshow("dst1_only_rotate", dst1)
43  cv2.imshow("dst2_only_scaling", dst2)
44  cv2.imshow("dst3_rotate_scaling", dst3)
45  cv2.imshow("dst4_rotate_scaling_translate", dst4)
46  cv2.waitKey(0)
```

8.6 행렬 연산을 통한 기하학 변환–어파인 변환

- 실행결과







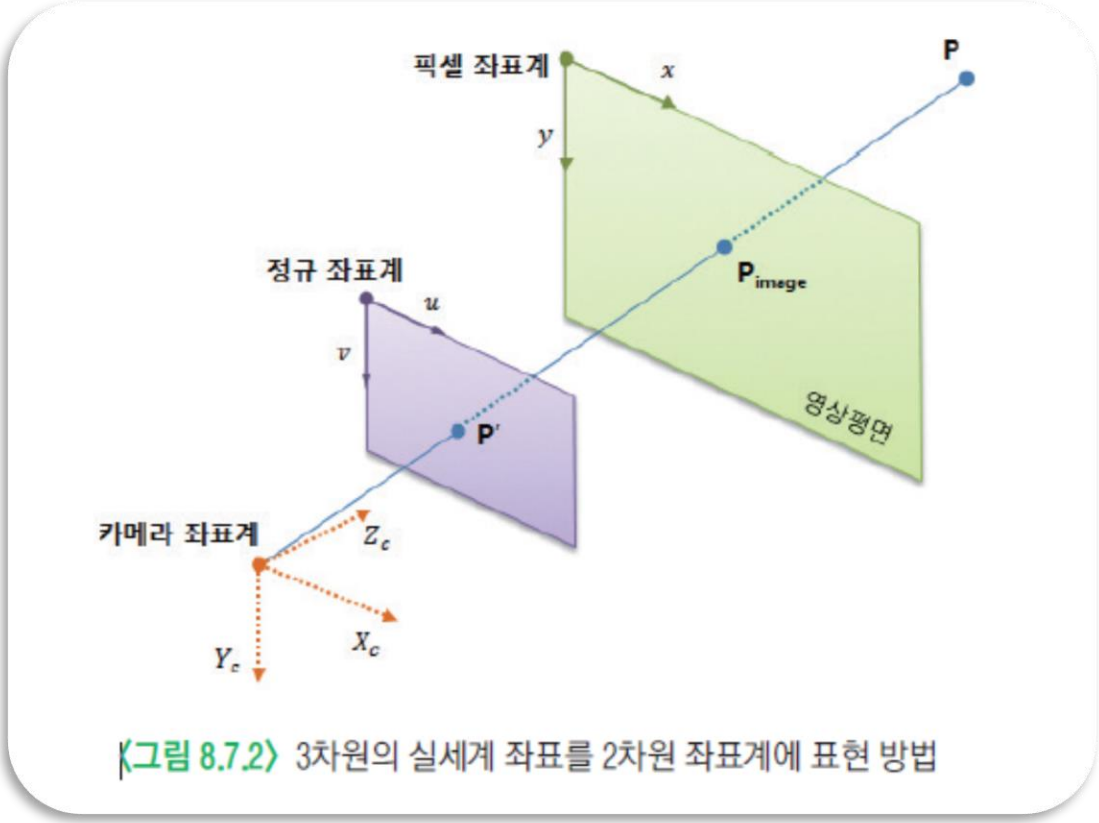
8.7 원근 투시(투영) 변환

- 아테네 학당
  - 라파엘로 산치오 작품 / 바티칸 사도궁전의 방들 중에서 서명실에 그린 벽화
  - 벽면에 그려진 그림에서 이렇게 입체감을 느끼는 이유 → 원근법

- 원근법
  - 눈에 보이는 3차원의 세계를 2차원의 그림(평면)으로 옮길 때에 관찰자가 보는 것 그대로 사물과의 거리를 반영하여 그리는 방법

8.7 원근 투시(투영) 변환

- 원근 투시 변환(perspective projection transformation)
  - 이 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

8.7 원근 투시(투영) 변환

- 동차 좌표계(homogeneous coordinates)
  - 모든 항의 차수가 동일하기 때문에 붙여진 이름으로서 n차원의 투영 공간을 n+1개의 좌표로 나타내는 좌표계
    - 직교 좌표인 (x, y)를 (x, y, 1)로 표현하는 것
    - 일반화해서 0이 아닌 상수 w에 대해 (x, y)를 (wx, wy, w)로 표현
    - 상수 w가 무한히 많기 때문에 (x, y)에 대한 동차 좌표 표현은 무한히 많이 존재
  - 동차 좌표계에서 한 점(wx, wy, w)을 직교 좌표로 나타내면
    - 각 원소를 w로 나우어 어서 (x/w, y/w)가 됨
    - 예, 동차 좌표계에서 한 점(5, 7, 5) → 직교 좌표에서(5/5, 7/5) 즉, (1 , 1.4)

8.7 원근 투시(투영) 변환

- 원근 변환을 수행하는 행렬

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

- OpenCV 함수
  - cv::getPerspectiveTransform() 함수
    - 4개의 좌표쌍으로부터 원근변환 행렬을 계산
  - cv::warpPerspective() 함수
    - 원근변환 행렬에 따라서 원근변환 수행
  - cv::transform() 함수
    - 입력영상의 4개 좌표와 원근 행렬을 인수로 입력하면 원근 변환된 좌표를 반환

8.7 원근 투시(투영) 변환

심화예제 8.7.1 원근 왜곡 보정 – 10.perspective_transform.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/perspective.jpg", cv2.IMREAD_COLOR)
04  if image is None: raise Exception("영상파일 읽기 에러")
05
06  pts1 = np.float32([(80, 40), (315, 133), (75, 300), (335, 300)])     # 입력 영상 4개 좌표
07  pts2 = np.float32([(50, 60), (340, 60), (50, 320), (340, 320)])      # 목적 영상 4개 좌표
08
09  perspect_mat = cv2.getPerspectiveTransform(pts1, pts2)               # 원근 변환 행렬
10  dst = cv2.warpPerspective(image, perspect_mat, image.shape[1::-1], cv2.INTER_CUBIC)
11  print("[perspect_mat] = \n%s\n" % perspect_mat )
12
13  ## 변환 좌표 계산 – 행렬 내적 이용 방법
14  ones = np.ones((4,1), np.float64)
15  pts3 = np.append(pts1, ones, axis=1)                    # 원본 좌표→동차 좌표 저장
16  pts4 = cv2.gemm(pts3, perspect_mat.T, 1, None, 1)       # 좌표 변환값 계산
17
18  ## 변환 좌표 계산 – cv2.transform() 함수 이용방법
```

원근 변환 행렬로 원근 변환 수행

8.7 원근 투시(투영) 변환

3차원 좌표 – 동차좌표로 저징

```
19  # pts3 = np.expand_dims(pts1, axis=0)                    # 차원 증가
20  # pts4 = cv2.transform(pts3, m=perspect_mat)
21  # pts4 = np.squeeze(pts4, axis=0)                         # 차원 감소
22  # pts3 = np.squeeze(pts3, axis=0)                         # 차원 감소 – 결과 표시 위해
23
24  print(" 원본 영상 좌표 \t 목적 영상 좌표 \t\t 동차 좌표 \t\t 변환 결과 좌표")
25  for i in range(len(pts4)):
        pts4[i] /= pts4[i][2] # 동차 좌표→직교 좌표
        print("%i : %-14s %-14s %-18s %-18s" % (i, pts1[i], pts2[i], pts3[i], pts4[i]))
        cv2.circle(image, tuple(pts1[i].astype(int)), 3, (0, 255, 0), -1)
29      cv2.circle(dst, tuple(pts2[i].astype(int)), 3, (0, 255, 0), -1)
30
31  cv2.imshow("image", image)
32  cv2.imshow("dst_perspective", dst)
33  cv2.waitKey(0)
```

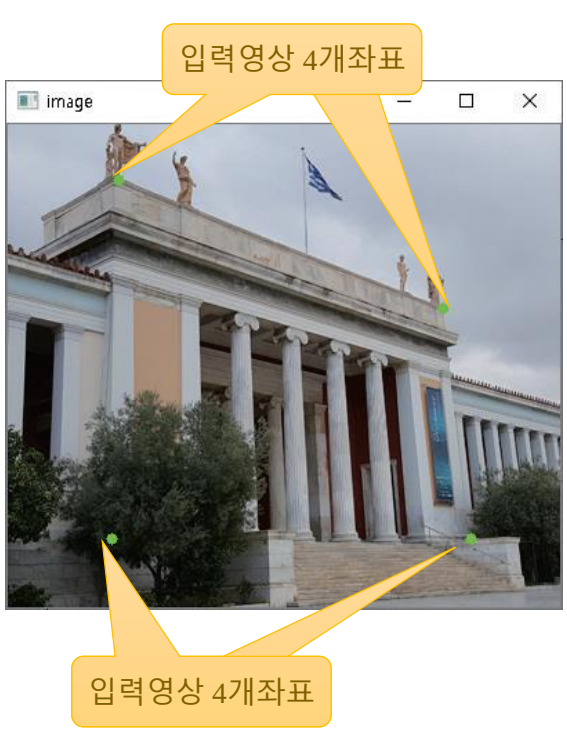
상수ω로 나누어서 동차좌표를 직교좌표로 변환

8.7 원근 투시(투영) 변환

- 실행결과

```
Run: 10.perspective_transform
C:\Python\python.exe D:/source/chap08/10.perspective_transform.py
[perspect_mat] =
[[ 6.25789284e-01  3.98298577e-02 -6.88839366e+00]
 [-5.02676539e-01  1.06358288e+00  5.13923399e+01]
 [-1.57086418e-03  5.25700042e-04  1.00000000e+00]]

   원본 영상 좌표   목적 영상 좌표     동차 좌표        변환 결과 좌표
0 : [80.  40.]     [50.  60.]     [80.  40.   1.]     [50.  60.   1.]
1 : [315.  133.]   [340.  60.]    [315.  133.   1.]   [340.  60.   1.]
2 : [ 75.  300.]   [ 50.  320.]   [ 75.  300.   1.]   [ 50.  320.   1.]
3 : [335.  300.]   [340.  320.]   [335.  300.   1.]   [340.  320.   1.]
```



8.7 원근 투시(투영) 변환 – 심화예제

- 마우스 드래그로 선택된 영역에 원근 변환 제거하기

심화예제 8.7.2 마우스 이벤트로 원근 왜곡 보정 – 11.perspective_event.py

```
01  import numpy as np, cv2
02  from Common.functions import imread, contain          # 좌표로 범위 확인 함수 임포트
03
04  def draw_rect(img):                                   # 좌표 사각형 그리기 함수
05      rois = [(p - small, small * 2) for p in pts1]     # 좌표 사각형 관심 영역
06      for (x, y), (w, h) in np.int32(rois):
07          roi = img[y:y+h, x:x+w]                       # 좌표 사각형 범위 가져오기
08          val = np.full(roi.shape, 80, np.uint8)        # 컬러(3차원) 행렬 생성
09          cv2.add(roi, val, roi)                        # 관심영역 밝기 증가
10          cv2.rectangle(img, (x, y, w, h), (0, 255, 0), 1)
11      cv2.polylines(img, [pts1.astype(int)], True, (0, 255, 0), 1)  # 4개 좌표 잇기
12      cv2.imshow("select rect", img)
13
14  def warp(img):                                        # 원근 변환 수행 함수
15      perspect_mat = cv2.getPerspectiveTransform(pts1, pts2)
16      dst = cv2.warpPerspective(img, perspect_mat, (350, 400), cv2.INTER_CUBIC) # 원근 변환
17      cv2.imshow("perspective transform", dst)
18
```

4개좌표 잇는 직선

4개 좌표 - 작은 사각형으로 표시

마우스 드래그로 선택된 4개 좌표와 목적영상 4개 좌표로 원근 변환 행렬 계산

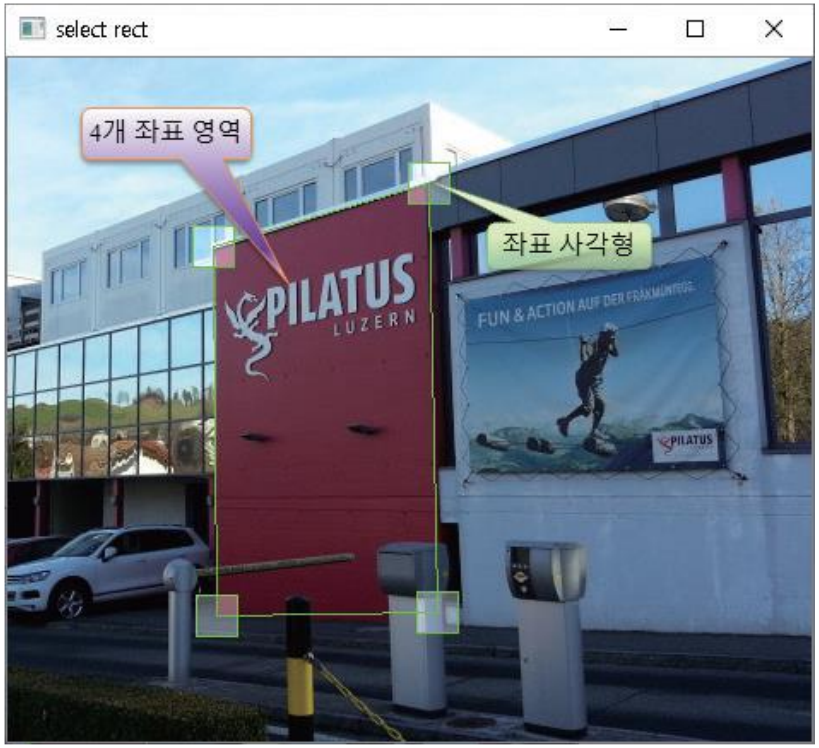
8.7 원근 투시(투영) 변환

```
19  def onMouse(event, x, y, flags, param):                  # 마우스 이벤트 처리 함수
20      global check
21      if event == cv2.EVENT_LBUTTONDOWN:
22          for i, p in enumerate(pts1):
23              p1, p2 = p - small, p + small                # p 좌표의 우상단, 좌하단 좌표생성
24              if contain((x,y), p1, p2): check = i         # 클릭 좌표로 좌표 사각형 선택
25
26      if event == cv2.EVENT_LBUTTONUP:  check = -1         # 마우스 업시 좌표번호 초기화
27
28      if check >= 0 :                                      # 좌표 사각형 선택 시
29          pts1[check] = (x, y)
30          draw_rect(np.copy(image))
31          warp(np.copy(image))
32
33  image = cv2.imread('images/perspective2.jpg', cv2.IMREAD_COLOR)
34  if image is None: raise Exception("영상파일 읽기 에러")
35
36  small = np.array([12, 12])                               # 좌표 사각형 크기
37  check = -1                                               # 선택 좌표 사각형 번호 초기화
38  pts1 = np.float32([(100, 100), (300, 100), (300, 300), (100, 300)])  # 4개 좌표 초기화
39  pts2 = np.float32([(0, 0), (400, 0), (400, 350), (0, 350)])          # 목적 영상 4개 좌표
40
41  draw_rect(np.copy(image))
42  cv2.setMouseCallback("select rect", onMouse, 0)
43  cv2.waitKey(0)
```

4개 사각형 중 선택된 사각형의 번호 체크

8.7 원근 투시(투영) 변환

- 실행결과



단원 요약

- 사상(mapping)은 화소들의 배치를 변경할 때, 입력영상의 좌표가 새롭게 배치될 해당 목적영상의 좌표를 찾아서 화소값을 옮기는 과정을 말한다.

- 목적영상에서 홀의 화소들을 채우고, 오버랩이 되지 않게 화소들을 배치하여 목적영상을 만드는 기법을 보간법(interpolation)이라 하며, 그 종류에는 최근접 이웃 보간법, 양선형 보간법, 3차 회선 보간법 등 다양한 방법이 있다.

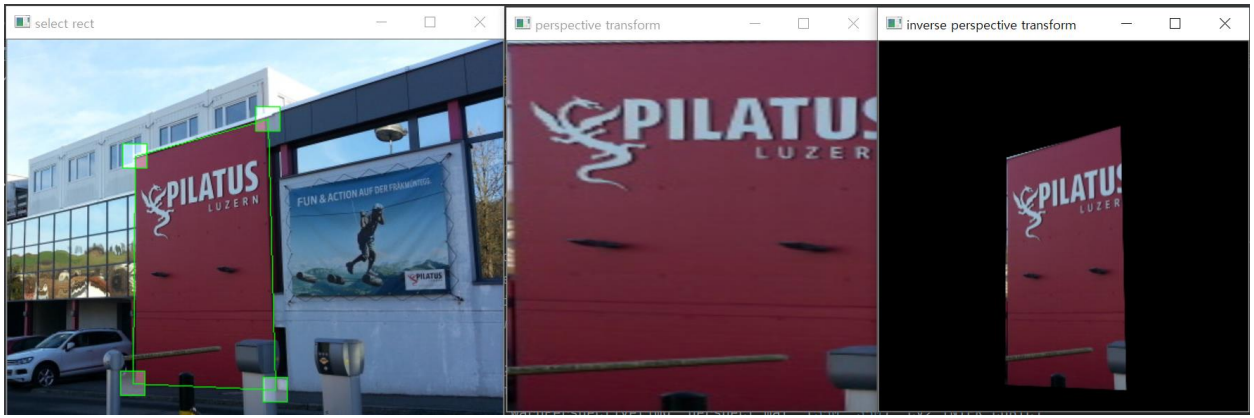
- 2×3 크기의 어파인 변환 행렬을 이용해서 회전, 크기변경, 평행이동 등을 복합적으로 수행할 수 있다.

- 원근법은 눈에 보이는 3차원의 세계를 2차원의 평면으로 옮길 때에 관찰자가 보는 것 그대로 사물과의 거리를 반영하여 그리는 방법을 말한다.

8. 실습 과제

- (과제) 연습문제 10. (p.402)
  - 원본 영상에 (50, 60) 좌표만큼 평행이동을 수행하는 프로그램을 작성하시오. 직접 translate() 함수를 작성한 결과와 OpenCV 함수를 사용한 결과를 모두 표시하시오.
  - 심화예제 8.6.2를 타이핑한 이후, getAffineMat함수의 코드에 대해 자세히 설명하시오.

- (보너스)
  - 심화 예제 8.7.2를 수정하여, 역 원근 변환을 수행하는 창을 추가로 띄우시오.
    - 역행렬 계산 함수: np.linalg.inv(행렬)
    - 역행렬 계산 함수를 사용하지 않고도 구현해 보시오.



8. 실습 규칙

- 실습 과제는 실습 시간내로 해결해야 합니다.
  - 해결 못한경우 실습 포인트를 얻지 못합니다.
  - -> 집에서 미리 예습하고 오길 권장합니다.

- 코드 공유/보여주기 금지. 의논 가능.

- 보너스문제까지 해결한 학생은 조기 퇴실 가능