영상처리프로그래밍
영상의 기하학적 변환

한림대학교 소프트웨어융합대학
박섭형

2022년 1학기

1 영상의 기하학적 변환

- 영상 내에 있는 화소들을 공간적으로 재배치하는 변환
- 영상의 크기 변환
- 영상의 회전 변환
- 영상의 대칭 변환
- 영상의 이동 변환
- 영상 내 객체의 모양 변경 등

[2]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

영상의 평행 이동 (translation)

영상의 모든 화소를 x축 방향으로 dx만큼, y축 방향으로 dy만큼 이동

$$(x, y) \rightarrow (x', y')$$

여기에서 (x, y)는 입력 영상의 화소 위치이고, (x', y')는 목적 영상의 화소 위치

- 순방향 사상 (forward mapping)

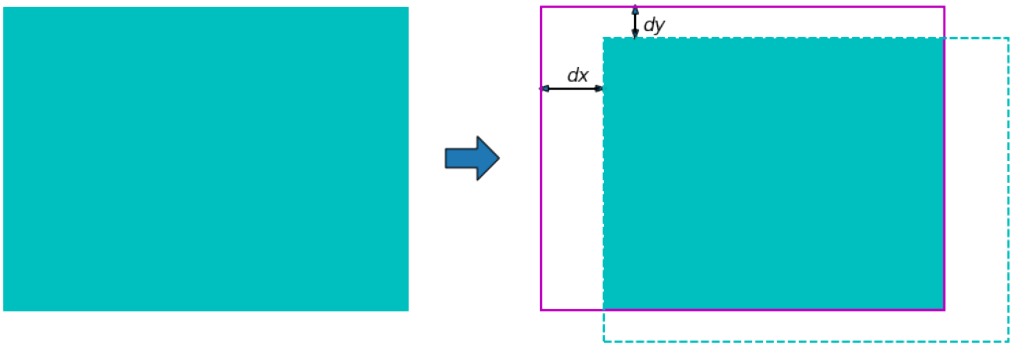
$$x' = x + dx$$

$$y' = y + dy$$

- 역방향 사상 (backward mapping)

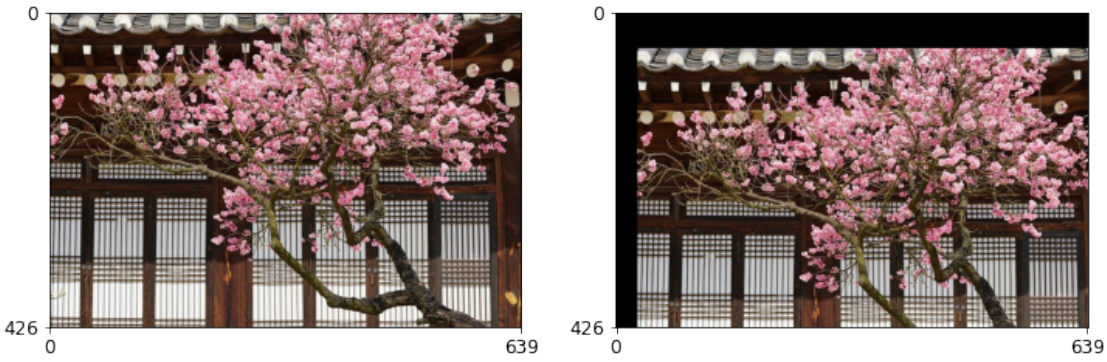
$$x = x' - dx$$

$$y = y' - dy$$

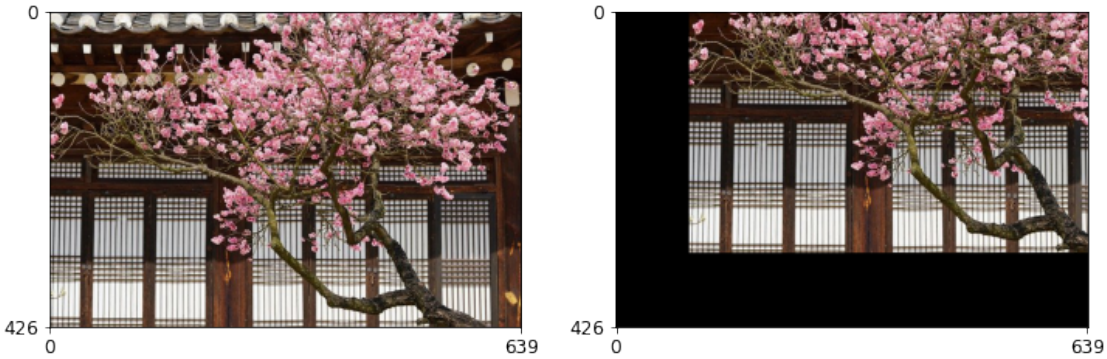


[4]:
```
import cv2
import numpy as np

def translate_forward(img, dx, dy):
    if type(dx) == int and type(dy) == int:
        dst = np.zeros_like(img)
        for y in range(img.shape[0]):
            for x in range(img.shape[1]):
                x_new = x + dx
                y_new = y + dy
                if (0 <= x_new < dst.shape[1]) and (0 <= y_new < dst.shape[0]):
                    dst[y_new, x_new] = img[y, x]
        return dst
```

[5]:
```
img = cv2.imread('plum_640.jpg')
dst = translate_forward(img, 30, 50)
ipp.show_two_images(img[...,::-1], dst[...,::-1])
```



[6]:
```
dst = translate_forward(img, 100, -100)
ipp.show_two_images(img[...,::-1], dst[...,::-1])
```

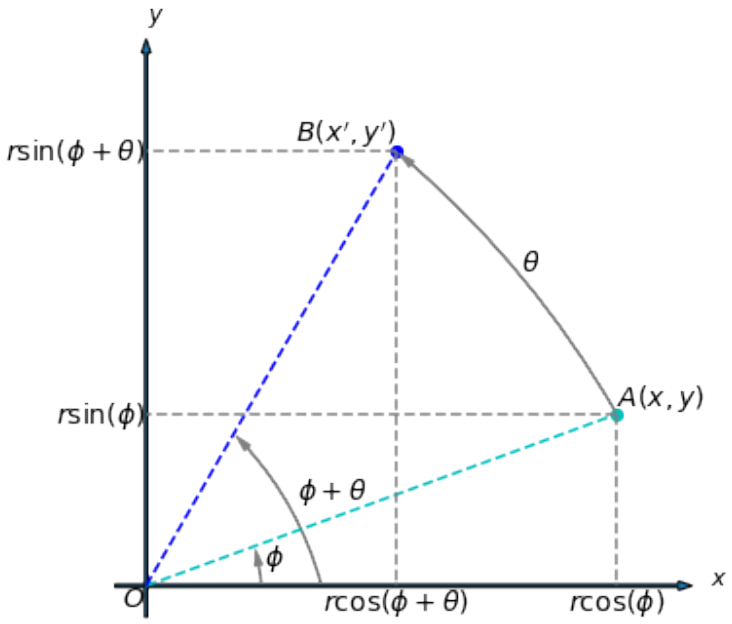


[7]:
```
img.shape
```

[7]:
```
(427, 640, 3)
```

영상의 회전

- 유클리드 공간에서 한 점 (x, y)를 x 축의 양의 방향을 기준으로 반시계 방향으로 θ 만큼 회전시킨 점의 좌표를 (x', y')이라고 하자.



$$r = \sqrt{x^2 + y^2}$$

$$x' = r\cos(\phi + \theta) = r(\cos\phi\cos\theta - \sin\phi\sin\theta) = x\cos\theta - y\sin\theta$$

$$y' = r\sin(\phi + \theta) = r(\sin\phi\cos\theta + \cos\phi\sin\theta) = y\cos\theta + x\sin\theta$$

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

- 유클리드 공간에서 한 점을 x 축의 양의 방향을 기준으로 반시계 방향으로 θ 만큼 회전시키는 변환을 수행하는 변환은 다음과 같이 표현되는 회전 행렬로 나타낼 수 있다.

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

- x 축의 양의 방향을 기준으로 반시계 방향으로 θ 만큼 회전시키는 변환을 수행하는 변환의 행렬
  - $\cos(-\theta) = \cos\theta, \ \ \sin(-\theta) = -\sin\theta$

$$R(-\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

- 유클리드 공간에서 한 점 (x, y)를 θ 만큼 회전시키는 변환은 (x, y)를 열벡터 $\mathbf{a} = \begin{bmatrix} x \\ y \end{bmatrix}$ 로 표현한 다음에 회전 행렬과 $\mathbf{a}$를 곱하는 것으로 표현된다.

$$R\mathbf{a} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}.$$

따라서 점 (x, y)를 θ 만큼 회전시킨 점의 위치는 $(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$가 된다.

- 예를 들어서, 점 $(1, 1)$을 $45°$ 회전시킨 점의 좌표는 다음과 같이 구할 수 있다.

$$\begin{bmatrix} \cos 45° & -\sin 45° \\ \sin 45° & \cos 45° \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}.$$

$$\left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}\right) = \left(0, \frac{2}{\sqrt{2}}\right) = \left(0, \sqrt{2}\right)$$

[9]:
```
def rotation_matrix(theta, degree=True):
    if degree:
        theta = np.deg2rad(theta)
    return np.array([[np.cos(theta), -np.sin(theta)],
                     [np.sin(theta), np.cos(theta)]])
```

- 예: 점 $A(1, 1)$을 $45°$도 회전시키는 경우

[10]:
```
A = np.array((1, 1)).reshape((2,1))
theta = 45.
R = rotation_matrix(theta)
B = R @ A

print(A)
print(R)
print(B)
```

```
[[1]
 [1]]
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[[0.        ]
 [1.41421356]]
```

[11]:
```
np.sqrt(2)
```

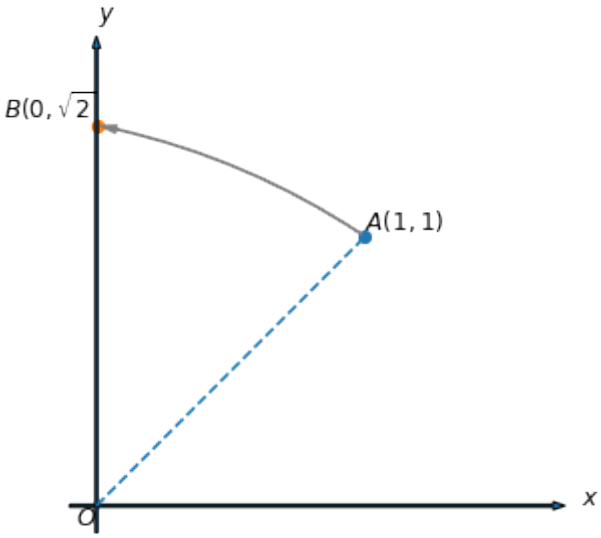
[11]:
```
1.4142135623730951
```

[12]:
```
A = np.array((1, 1))
B = R @ A
B
```

[12]:
```
array([0.        , 1.41421356])
```

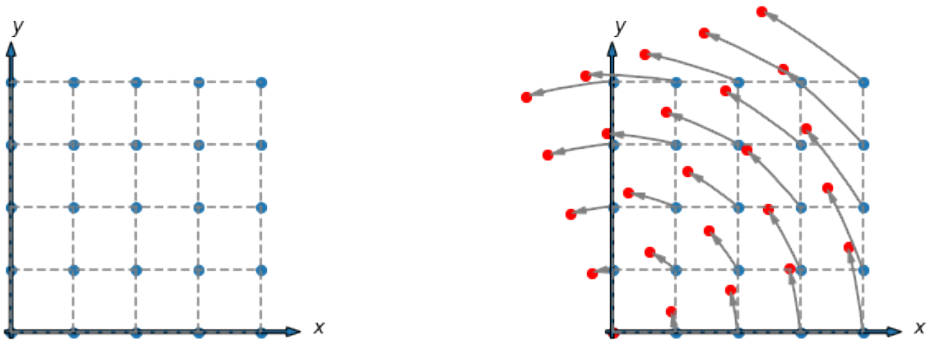
[13]:
```
A.shape
```

[13]:
```
(2,)
```

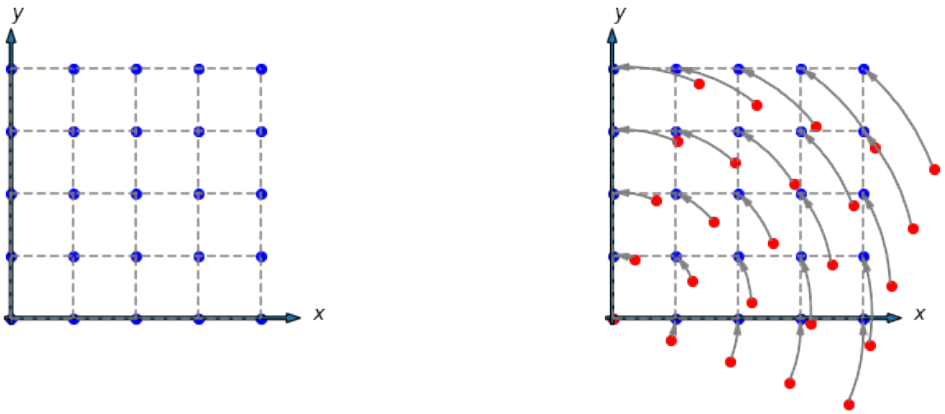


예: 5×5 영상을 반시계 방향으로 20도 회전시키기

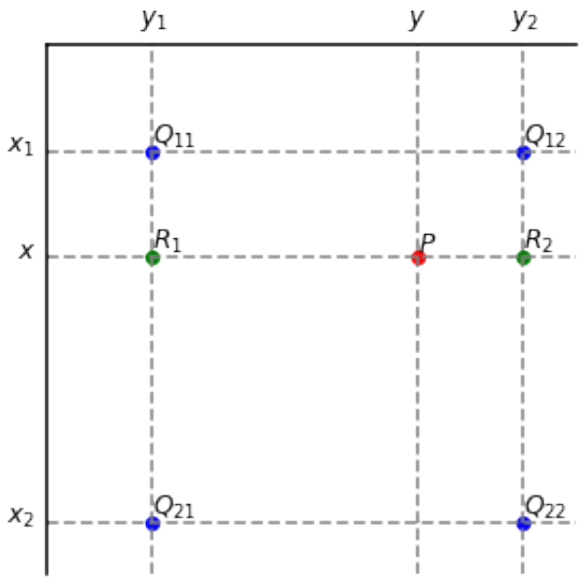
Forward Mapping



Backward Mapping



- 양방향 보간(bilinear interpolation) 방법



- $Q_{11}(x_1, y_1)$, $Q_{12}(x_1, y_2)$, $Q_{21}(x_2, y_1)$, $Q_{22}(x_2, y_2)$

$$f(R_1) = f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) = f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(R) = f(x, y) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

- 회전의 기준을 영상의 중심으로 하는 경우

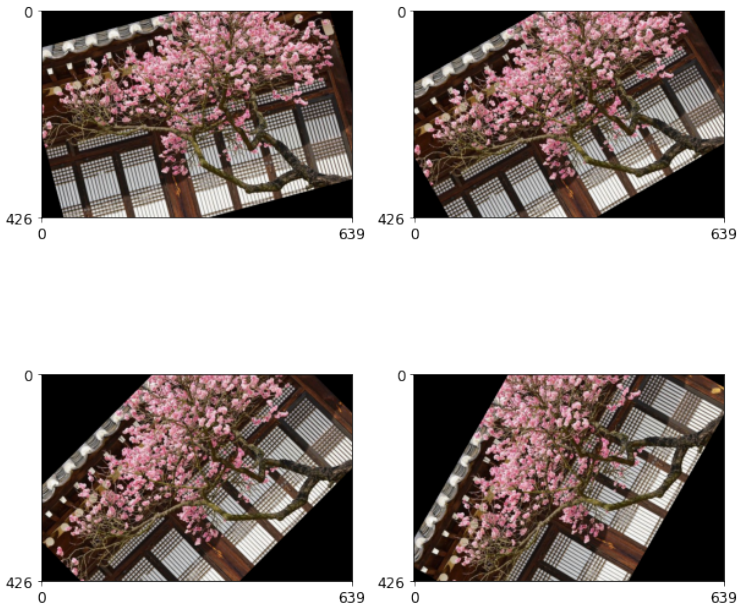
[18]:
```
# image rotation
# 아래 방향: x 축의 양의 방향
# 오른쪽 방향: y 축의 양의 방향
import numpy as np

def bilinear_interpolation(img, p):
    x, y = p
    x1 = int(x)
    x2 = x1 + 1
```

```
    y1 = int(y)
    y2 = y1 + 1

    img_R1 = (x2 - x) *img[x1, y1] + (x - x1) *img[x2, y1]
    img_R2 = (x2 - x) *img[x1, y2] + (x - x1) *img[x2, y2]

    return (y2 - y) *img_R1 + (y - y1) *img_R2


def rotate_an_image(img, theta, center=(0,0), degree=True):

    dst = np.zeros_like(img)
    if degree:
        theta = np.deg2rad(theta)
    SIN = np.sin(-theta)
    COS = np.cos(-theta)

    for x in range(dst.shape[0]):
        for y in range(dst.shape[1]):
            x1, y1 = x - center[0], y - center[1]
            x_new = x1 * COS - y1 * SIN
            y_new = x1 * SIN + y1 * COS
            x1, y1 = x_new + center[0], y_new + center[1]
            if (0 <= x1 <= img.shape[0] - 1) and (0 <= y1 <= img.shape[1] - 1):
                dst[x,y] = bilinear_interpolation(img, (x1, y1)).round(0)
    return dst
```

[19]:
```
img = cv2.imread('plum_640.jpg')
dst_15 = rotate_an_image(img[...,::-1], 15, (img.shape[0]//2, img.shape[1]//2))
dst_30 = rotate_an_image(img[...,::-1], 30, (img.shape[0]//2, img.shape[1]//2))
dst_45 = rotate_an_image(img[...,::-1], 45, (img.shape[0]//2, img.shape[1]//2))
dst_60 = rotate_an_image(img[...,::-1], 60, (img.shape[0]//2, img.shape[1]//2))
```

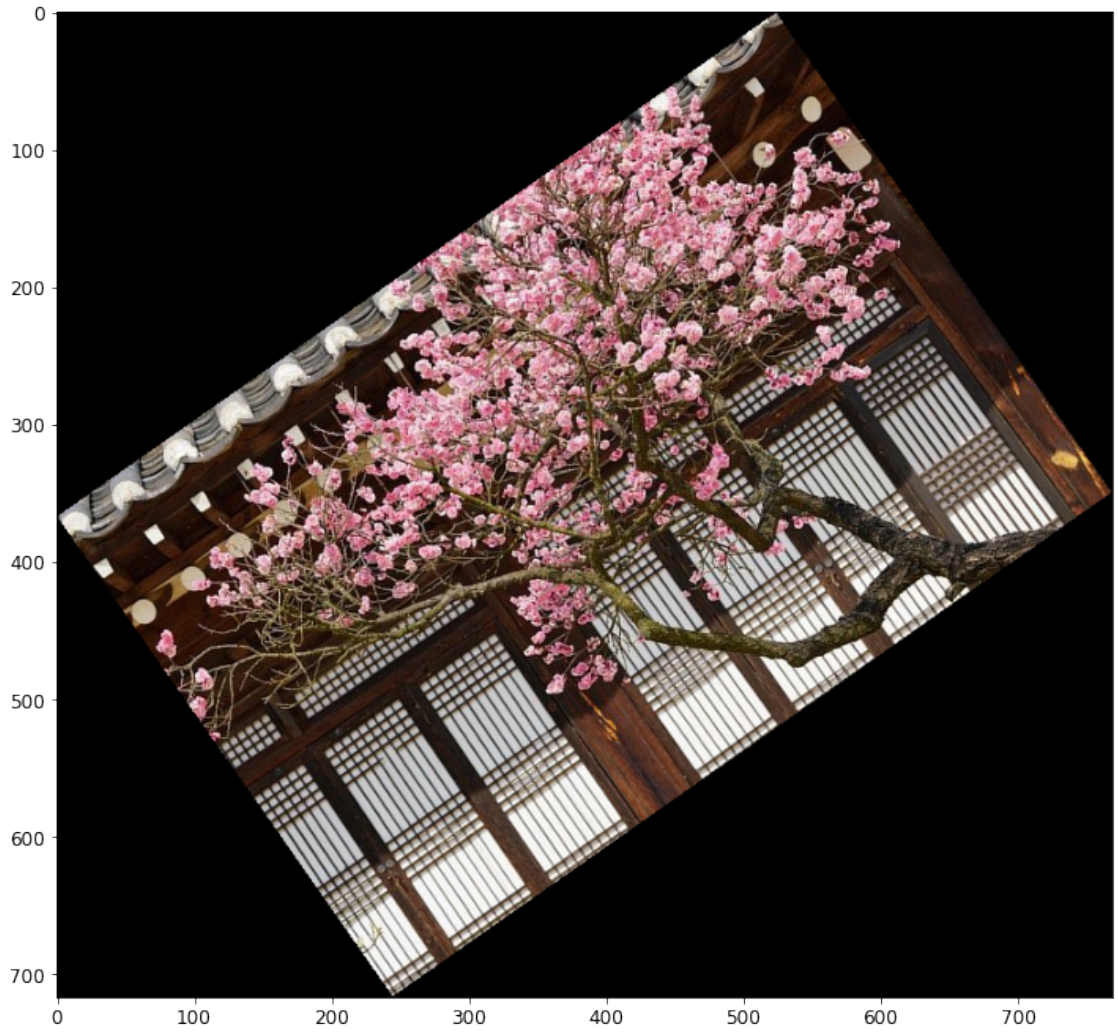
[20]:
```
plt.figure(figsize=(12,8))
plt.imshow(dst_15)
plt.show()
```

[21]:
```
ipp.show_images(dst_15, dst_30, dst_45, dst_60)
```



[22]:
```
from scipy import ndimage

#rotation angle in degree
dst_35 = ndimage.rotate(img[...,::-1], 35)
```

[23]:
```
plt.figure(figsize=(12,12))
plt.imshow(dst_35)
plt.show()
```



영상의 어파인 변환 (Affine Transformation)

- Affine 변환
  - 한 영상을 변환했을 때, 변환 전에 공선점들이 변환된 영상에서도 공선점이 되고, 점과 점 사이의 거리의 비율도 유지되는 변환
  - 공선점 (collinear points): 같은 직선 상에 있는 점들

- 영상의 크기 변환

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

- 예:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 3y \end{bmatrix}.$$

- 회전 변환

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

- 영상의 평행 이동

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}.$$

- 예:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 5 \\ 10 \end{bmatrix}.$$

- 대칭 변환 (Reflection)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

- 가로 엇갈림 변환 (Shear transformation)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & c_x \\ c_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

확대 행렬 (또는 첨가 행렬, augmented matrix**)과 확대 벡터**(augmented vector)**를 이용한 변환 식의 표현**

- 영상의 평행 이동 식의 변형

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \end{bmatrix}.$$

$$\mathbf{y} = A\mathbf{x} + \mathbf{b}$$

확대 행렬과 확대 벡터를 이용하면 위 식을 다음과 같이 표현할 수 있다.

$$\begin{bmatrix} \mathbf{y} \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & A & & \mathbf{b} \\ 0 & \cdots & 0 & 1 \end{array}\right] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- 회전 변환

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- 영상의 크기 변환

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- 대칭 변환 (Reflection)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- 가로 엇갈림 변환 (Shear transformation)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & c_x \\ c_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & c_x & 0 \\ c_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Affine 변환 예 1: **점의 평행 이동 변환 후 회전 변환**

- 유클리드 공간에서 좌표가 (x, y) 인 점을 x 축 방향으로 0.5, y 축 방향으로 -0.3 이동한 후에 45° 도 회전하는 변환

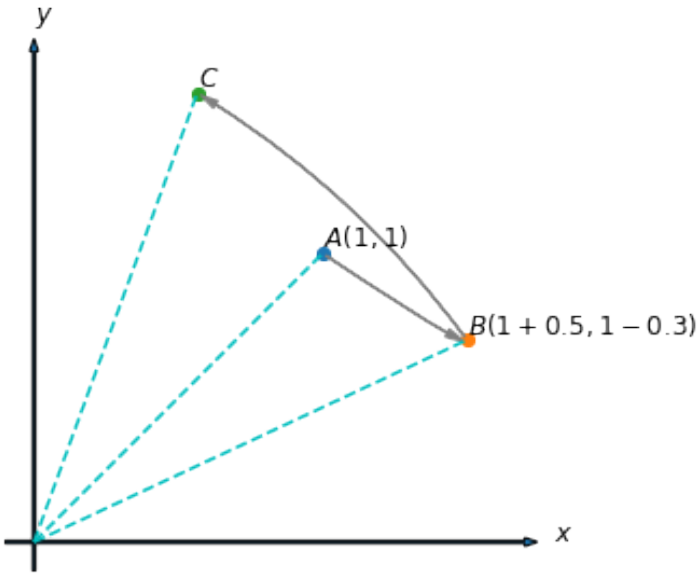
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0.5 \\ 0 & 1 & -0.3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

- 개별 변환의 단계적 적용

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0.5 \\ 0 & 1 & -0.3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \end{bmatrix} \begin{bmatrix} x + 0.5 \\ y - 0.3 \\ 1 \end{bmatrix}.$$

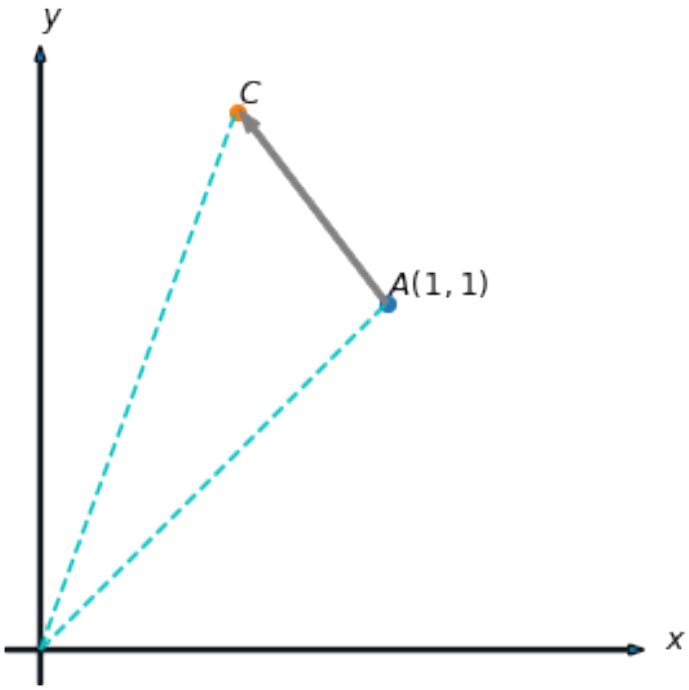
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos 45^\circ + 0.5\cos 45^\circ - y\sin 45^\circ + 0.3\sin 45^\circ \\ x\sin 45^\circ + 0.5\sin 45^\circ + y\cos 45^\circ - 0.3\cos 45^\circ \end{bmatrix}.$$



- 통합 변환 적용

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0.5\cos 45^\circ + 0.3\sin 45^\circ \\ \sin 45^\circ & \cos 45^\circ & 0.5\sin 45^\circ - 0.3\cos 45^\circ \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

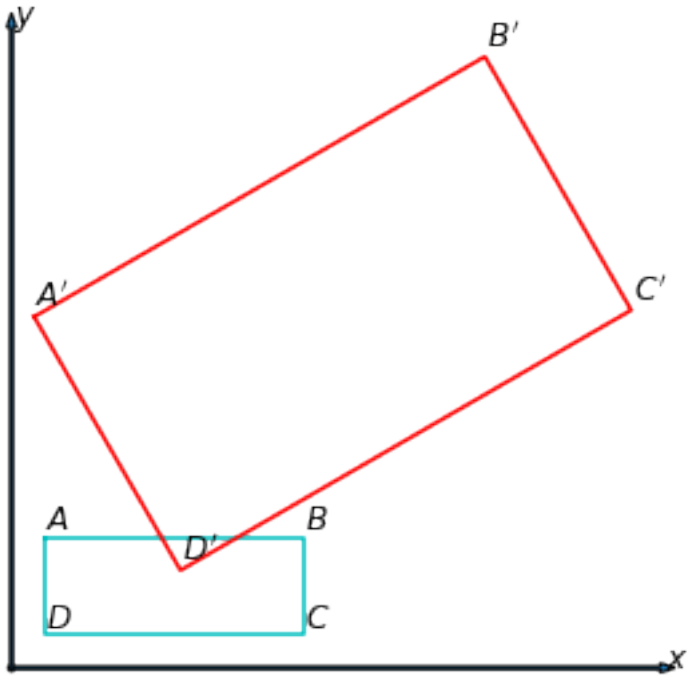
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x\cos 45^\circ + 0.5\cos 45^\circ - y\sin 45^\circ + 0.3\sin 45^\circ \\ x\sin 45^\circ + 0.5\sin 45^\circ + y\cos 45^\circ - 0.3\cos 45^\circ \end{bmatrix}.$$



Affine 변환 예 2: : **영상의 평행 이동 변환 후 크기 변경 변환 후 회전 변환**

- 다각형의 affine 변환

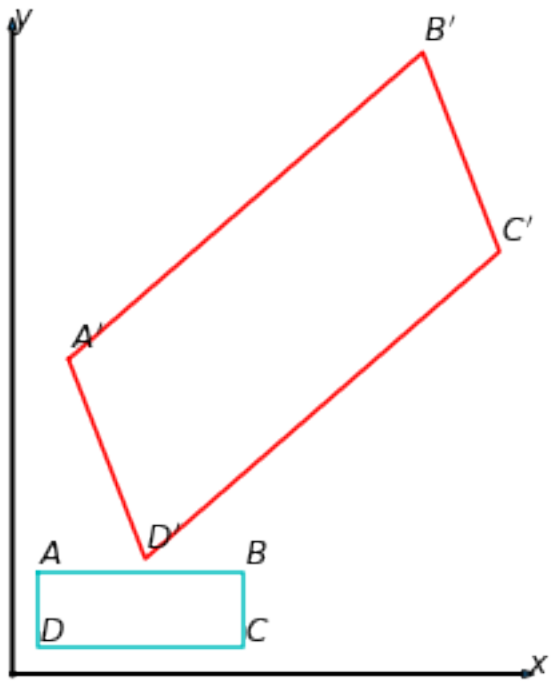
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$



Affine 변환 예 3 : 영상의 평행 이동 변환 후 회전 변환 후 크기 변경 변환

- 다각형의 affine 변환

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$



cv2.warpAffine() 함수 이용하기

$$\texttt{dst}(x,y) = \texttt{src}(\texttt{M}_{11}x + \texttt{M}_{12}y + \texttt{M}_{13}, \texttt{M}_{21}x + \texttt{M}_{22}y + \texttt{M}_{23})$$

[29]:
```
help(cv2.warpAffine)
```

```
Help on built-in function warpAffine:

warpAffine(...)
    warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) ->
dst
    .   @brief Applies an affine transformation to an image.
    .
    .   The function warpAffine transforms the source image using the specified
matrix:
    .
    .   \f[\texttt{dst} (x,y) =  \texttt{src} ( \texttt{M} _{11} x +  \texttt{M}
```

```
_{12} y +  \texttt{M} _{13}, \texttt{M} _{21} x +  \texttt{M} _{22} y +
\texttt{M} _{23})\f]
    .
    .   when the flag #WARP_INVERSE_MAP is set. Otherwise, the transformation is
first inverted
    .   with #invertAffineTransform and then put in the formula above instead of
M. The function cannot
    .   operate in-place.
    .
    .   @param src input image.
    .   @param dst output image that has the size dsize and the same type as src
.
    .   @param M \f$2\times 3\f$ transformation matrix.
    .   @param dsize size of the output image.
    .   @param flags combination of interpolation methods (see
#InterpolationFlags) and the optional
    .   flag #WARP_INVERSE_MAP that means that M is the inverse transformation (
    .   \f$\texttt{dst}\rightarrow\texttt{src}\f$ ).
    .   @param borderMode pixel extrapolation method (see #BorderTypes); when
    .   borderMode=#BORDER_TRANSPARENT, it means that the pixels in the
destination image corresponding to
    .   the "outliers" in the source image are not modified by the function.
    .   @param borderValue value used in case of a constant border; by default,
it is 0.
    .
    .   @sa  warpPerspective, resize, remap, getRectSubPix, transform
```

- cv2.warpAffine()로 영상에 회전 변환 적용하기
  - 유클리드 공간에서 회전 행렬

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{bmatrix}$$

- cv2.getRotationMatrix2D(center, angle, scale) 함수
  - center: 입력 영상에서 회전 중심
  - angle: 회전 각 (degrees). 양수는 반시계 방향
  - scale: 등방성 배율

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \texttt{center.x} - \beta \cdot \texttt{center.y} \\ -\beta & \alpha & \beta \cdot \texttt{center.x} + (1-\alpha)\cdot \texttt{center.y} \end{bmatrix}$$

where

$$\alpha = \texttt{scale} \cdot \cos \texttt{angle},$$
$$\beta = \texttt{scale} \cdot \sin \texttt{angle}$$

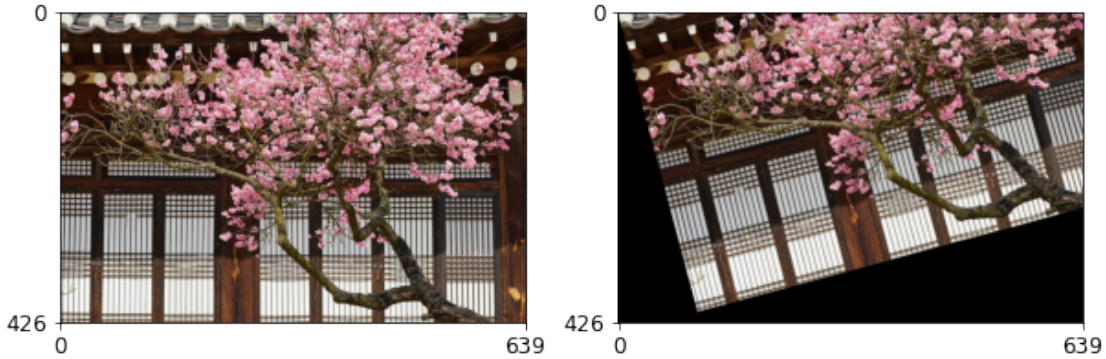
[30]:
```
cv2.getRotationMatrix2D((0,0), 45, 1.0)
```

[30]:
```
array([[ 0.70710678,  0.70710678,  0.        ],
       [-0.70710678,  0.70710678,  0.        ]])
```

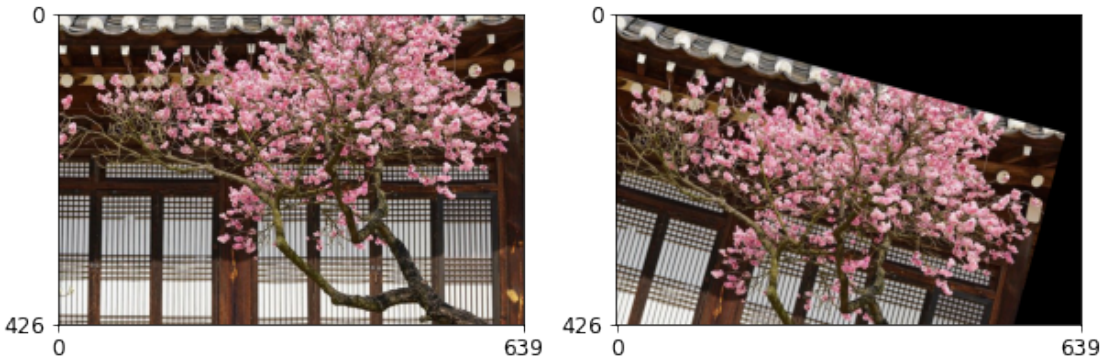
[31]:
```
rotation_matrix_2x3(45)
```

[31]:
```
array([[ 0.70710678, -0.70710678,  0.        ],
       [ 0.70710678,  0.70710678,  0.        ]])
```

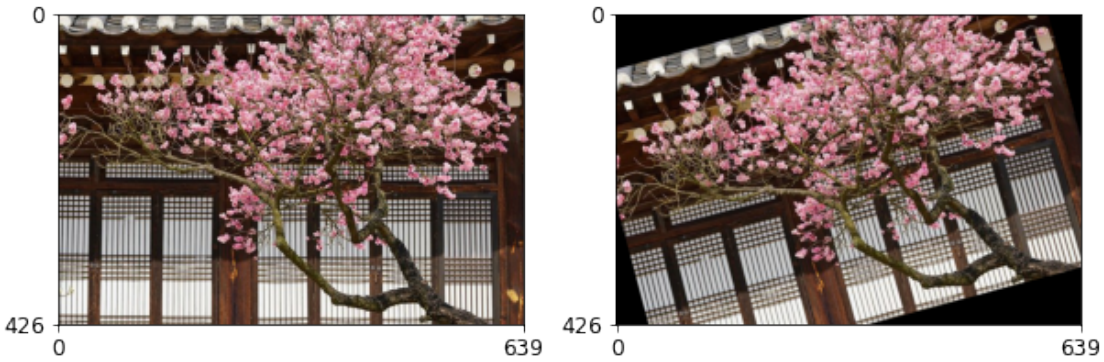
[32]:
```
A = cv2.getRotationMatrix2D((0,0), 15, 1.0)
dst = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], dst[...,::-1])
```



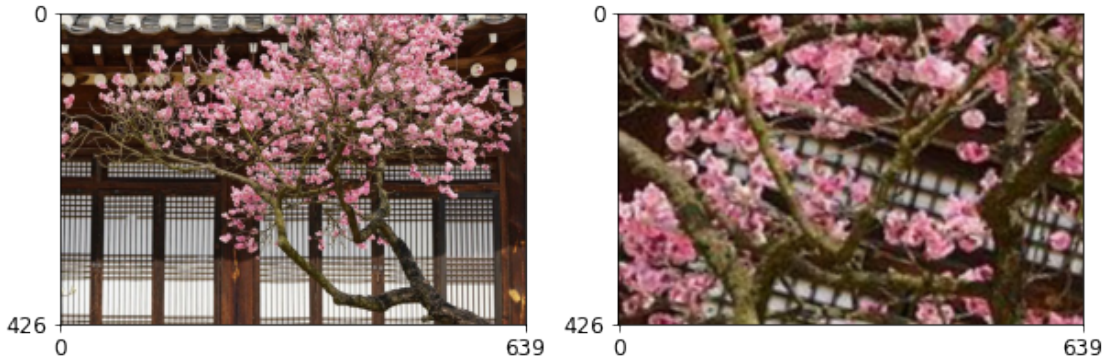
[33]:
```
A = cv2.getRotationMatrix2D((0,0), -15, 1.0)
dst = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], dst[...,::-1])
```



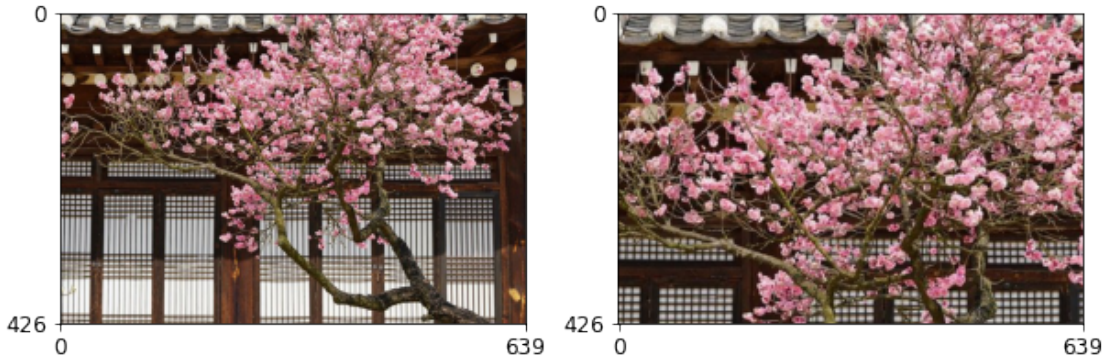
[34]:
```
A = cv2.getRotationMatrix2D((img.shape[1]//2, img.shape[0]//2), 15, 1.0)
dst = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], dst[...,::-1])
```



[35]:
```
A = cv2.getRotationMatrix2D((img.shape[1]//2, img.shape[0]//2), -15, 4.0)
dst = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], dst[...,::-1])
```



[36]:
```
A = cv2.getRotationMatrix2D((img.shape[1]//2, 0), 0, 1.5)
dst = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], dst[...,::-1])
```

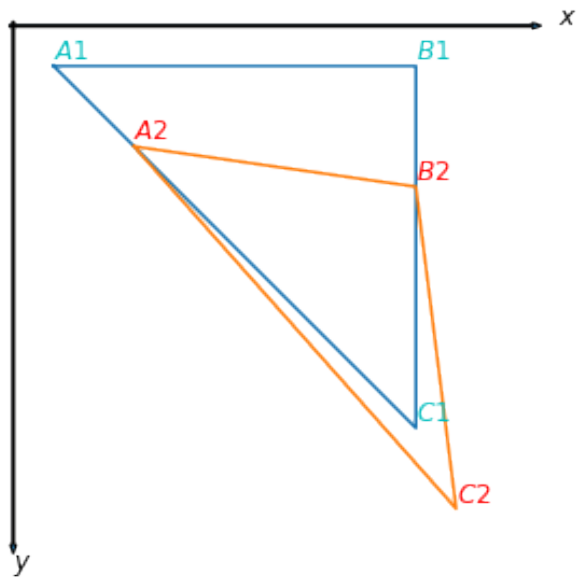


- cv2.getAffineTransform() 함수 이용하기

[37]:
```
help(cv2.getAffineTransform)
```

```
Help on built-in function getAffineTransform:

getAffineTransform(...)
    getAffineTransform(src, dst) -> retval
    .   @overload
```

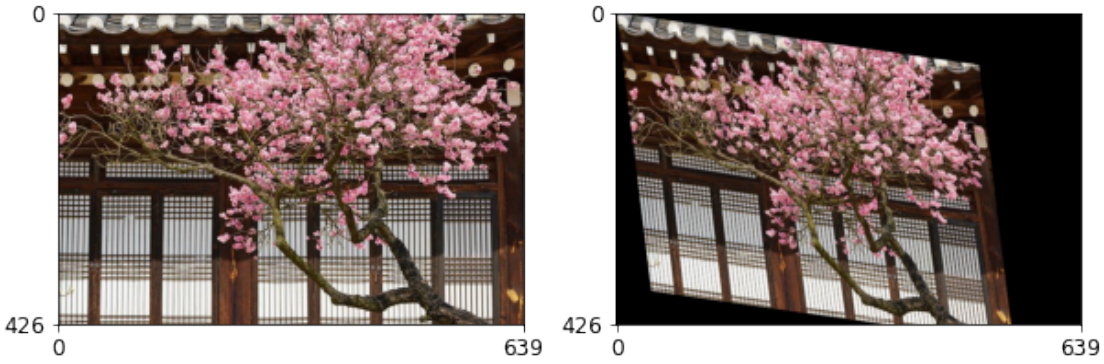
[38]:
```
A1 = np.float32((1, 1))
B1 = np.float32((10, 1))
C1 = np.float32((10, 10))

A2 = np.float32((3, 3))
B2 = np.float32((10, 4))
C2 = np.float32((11, 12))
```



[40]:
```
src = np.float32([A1, B1, C1])
dst = np.array([A2, B2, C2], np.float32)
A = cv2.getAffineTransform(src, dst)
A
```

[40]:
```
array([[0.77777778, 0.11111111, 2.11111111],
       [0.11111111, 0.88888889, 2.        ]])
```

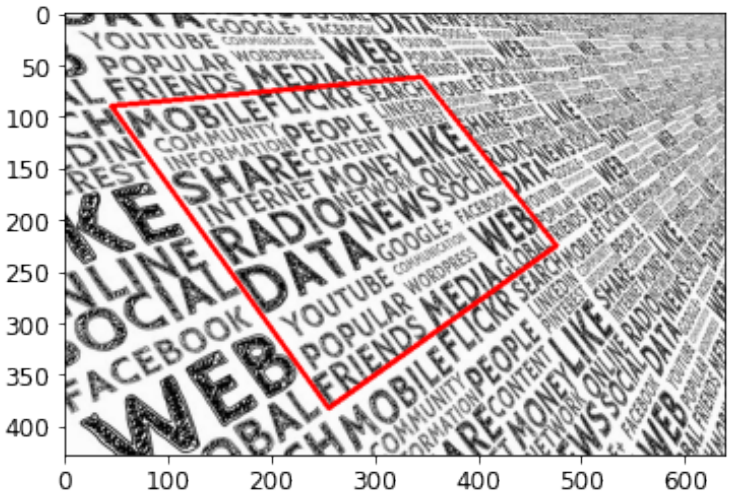
[41]:
```
transformed = cv2.warpAffine(img, A, (img.shape[1], img.shape[0]))
ipp.show_images(img[...,::-1], transformed[...,::-1])
```



원근 투영(투시) 변환 (Perspective projection transformation)

- 3차원의 실세계의 좌표를 투영 스크린 상의 2차원 좌표로 표현하는 변환

[42]:
```
img = cv2.imread('social-media_640.jpg')
img2 = img.copy()
A = (45,90)
B = (345,62)
C = (475,225)
D = (255,382)
RED = (0, 0, 255)
pts1 = [A, B, C, D, A]
for n in range(4):
    cv2.line(img2, pts1[n], pts1[n+1], RED, 3)

plt.imshow(img2[...,::-1])
plt.show()
```



[43]:
```
A2 = (10,10)
B2 = (410,10)
C2 = (410,410)
D2 = (10,410)

src = np.float32([A, B, C, D])
dst = np.float32([A2, B2, C2, D2])

perspective_mat = cv2.getPerspectiveTransform(src, dst)
transformed = cv2.warpPerspective(img, perspective_mat, (640, 428), cv2.
↪INTER_CUBIC)
```

[44]:
```
ipp.show_images(img2[...,::-1], transformed[...,::-1])
```