CHAPTER 7
영역 처리

김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

열림 연산: 입력 -> 침식 -> 팽창

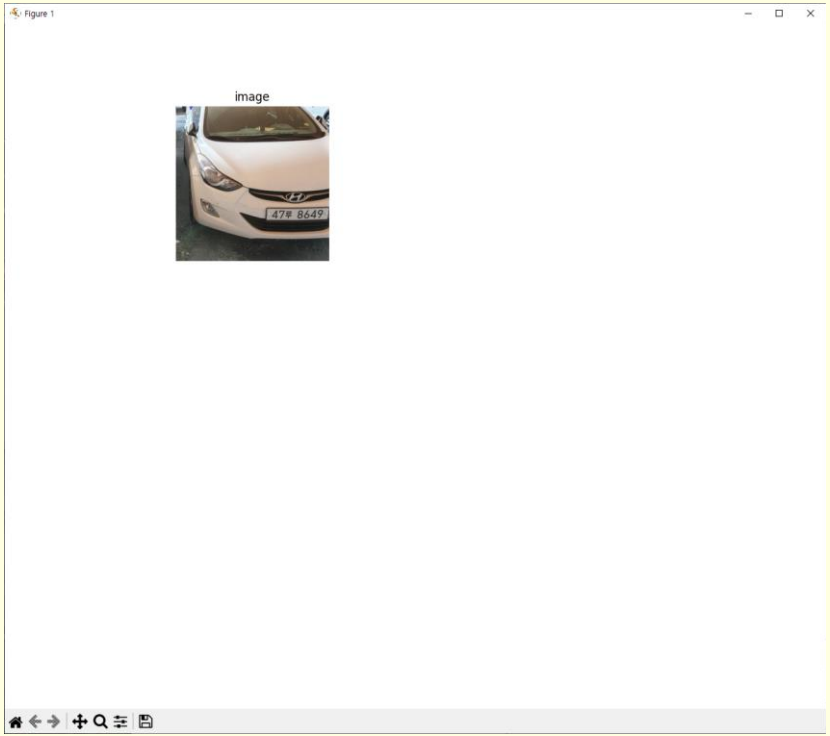
닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

7.4 모폴로지

7.4.4 닫힘 연산을 이용한 번호판 검출

```
import numpy as np, cv2
import matplotlib.pyplot as plt

image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)

titles = ['image']

plt.figure(figsize=(12, 10))

for idx, title in enumerate(titles):
    plt.subplot(3, 2, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title), cmap="gray")

plt.show()
```



열림 연산: 입력 -> 침식 -> 팽창

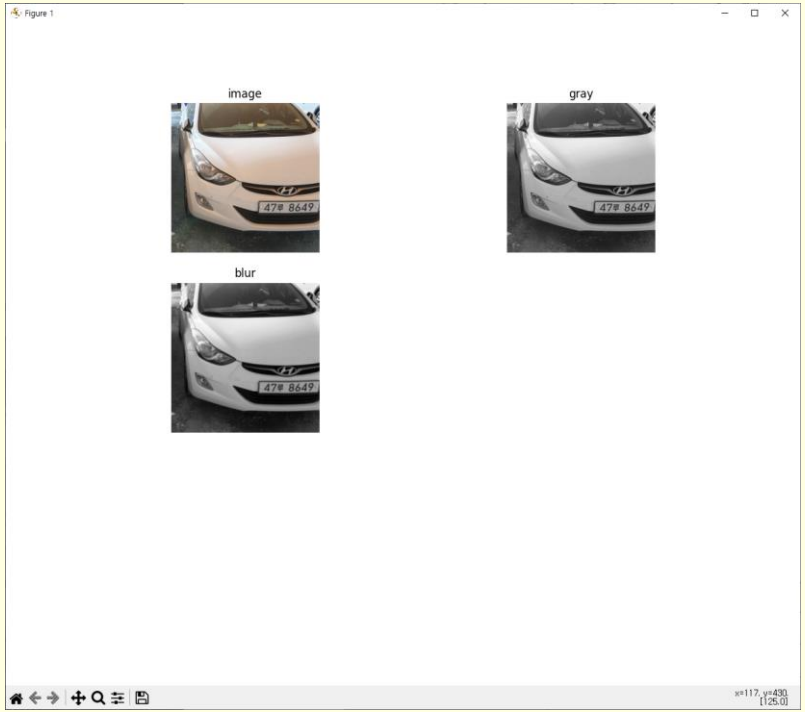
닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

7.4 모폴로지

7.4.4 닫힘 연산을 이용한 번호판 검출

```
...
image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)

mask = np.ones((5, 17), np.uint8)
번호판의 가로 세로 비율: 약 17:5
번호판을 검출하기 위해 17x5 크기의 마스크를 생성

Numpy는 (y, x)이기 때문에 1로 채워진 (5, 17)로 생성

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
원본 이미지를 그레이 스케일로 변환

blur = cv2.blur(gray, (5, 5))
잡음 제거를 위한 블러링 처리
여기서 5, 5는 5 x 5범위 내 이웃 픽셀을 평균으로, 블러링 처리하는 것이다.

titles = [‘image’, ‘blur’]
...
```



열림 연산: 입력 -> 침식 -> 팽창

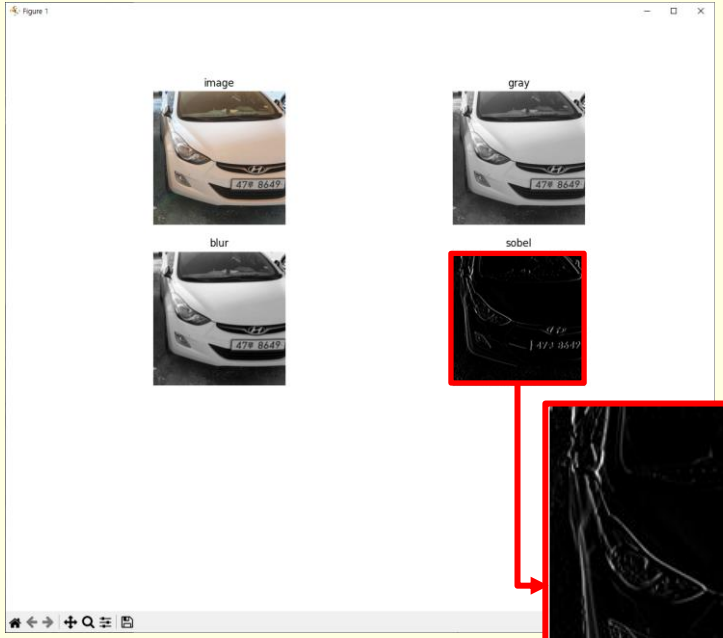
닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

■ 7.4 모폴로지

◆ 7.4.4 닫힘 연산을 이용한 번호판 검출

```
...
blur = cv2.blur(gray, (5, 5))

sobel = cv2.Sobel(blur, cv2.CV_8U, 1, 0, 5)

Sobel 함수를 이용하여 수직 방향의 에지를 검출

cv2.Sobel(img, type, dx, dy, ksize)

1. gray는 입력 이미지
2. cv2.CV_8U는 8비트 부호 없는 정수형(Unsigned int)을 의미
3. 1은 dx로, x방향 미분 차수를 의미
4. 0은 dy로, y방향 미분 차수를 의미
5. 5는 ksize로, 소벨 커널의 크기를 의미

cv2.Sobel(gray, cv2.CV_8U, 1, 0, 5) = 수직 방향의 에지를 검출
cv2.Sobel(gray, cv2.CV_8U, 0, 1, 5) = 수평 방향의 에지를 검출

수직 방향의 에지를 검출한 후, 모폴로지 닫힘 연산을 진행하면
주변 픽셀을 참조하여 픽셀의 값을 결정함.
그 결과는 이미지를 보면 쉽게 이해할 수 있다.

titles = [‘image’, ‘blur’]
...
```



열림 연산: 입력 -> 침식 -> 팽창

닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

7.4 모폴로지

◆ 7.4.4 닫힘 연산을 이용한 번호판 검출

```
...
sobel = cv2.Sobel(blur, cv2.CV_8U, 1, 0, 5)

ret, thresh = cv2.threshold(sobel, 120, 255, cv2.THRESH_BINARY)
```

ret = 임계값, thresh = 임계값을 적용한 결과

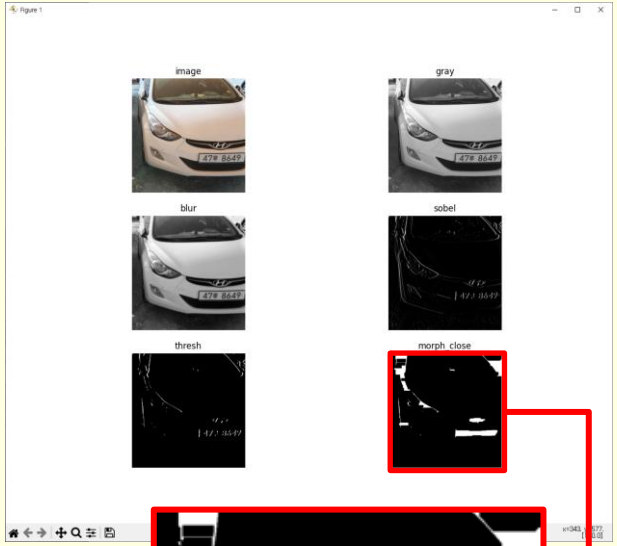
1. sobel은 입력 이미지
2. 120은 임계 값
3. 255는 최대 값 (이미지는 0 ~ 255로 이루어져 있기 때문에 최대값을 255로 주면 된다.)
4. cv2.THRESH_BINARY는 임계값을 넘으면 최대값으로, 넘지 못하면 0으로 처리

```
morph_close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, mask, iterations=3)
```

모폴로지 닫힘 연산

1. thresh는 입력 이미지
2. cv2.MORPH_CLOSE는 모폴로지 연산 종류(열림 또는 닫힘)
3. mask는 앞에서 설정한 마스킹 범위 (번호판에 해당하는 예상 범위)
4. iterations는 반복 횟수 (너무 적어도, 너무 많아도 이상하게 나온다.)

```
titles = ['image', 'gray', 'blur', 'sobel', 'thresh', 'morph_close']
...
```



열림 연산: 입력 -> 침식 -> 팽창

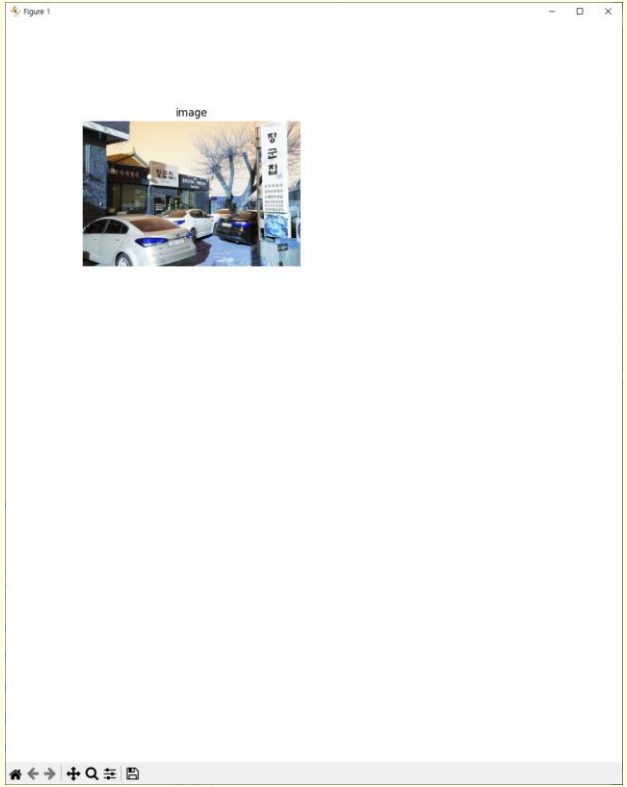
닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

7.4 모폴로지

◆ 7.4.5 닫힘 연산을 이용한 간판 검출

```
import numpy as np, cv2
import matplotlib.pyplot as plt

image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)

titles = ['image']

plt.figure(figsize=(10, 12))

for idx, title in enumerate(titles):
    plt.subplot(3, 2, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title), cmap="gray")

plt.show()
```



열림 연산: 입력 -> 침식 -> 팽창

닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

7.4 모폴로지

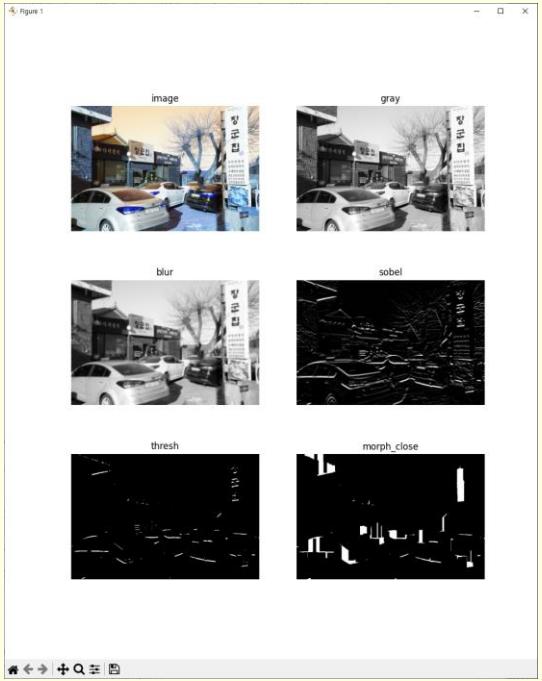
7.4.5 닫힘 연산을 이용한 간판 검출

```
...
image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

blur = cv2.blur(gray, (5, 5))

sobel = cv2.Sobel(blur, cv2.CV_8U, 0, 1, 5)

ret, thresh = cv2.threshold(sobel, 200, 255, cv2.THRESH_BINARY)

mask = np.ones((20, 5), np.uint8)

morph_close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, mask, iterations=3)

titles = ['image', 'gray', 'blur', 'sobel', 'thresh', 'morph_close']
...
```

수평 방향 검출

구조물이 많아 임계값 200으로 조정

검출하려는 간판이 약 5 x 20 크기



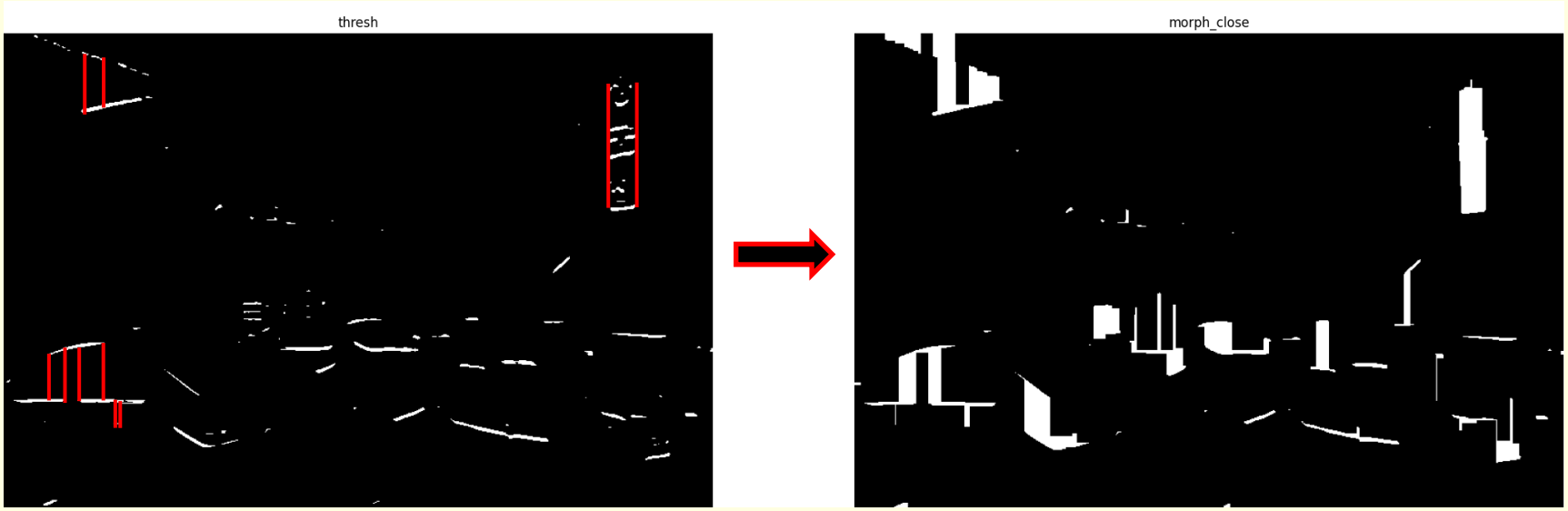
열림 연산: 입력 -> 침식 -> 팽창

닫힘 연산: 입력-> 팽창 -> 침식

7. 영역 처리

■ 7.4 모폴로지

◆ 7.4.5 닫힘 연산을 이용한 간판 검출



모폴로지 닫힘 연산을 통해 지정한 마스크 범위 만큼의 인접 픽셀들끼리 연결되어
간판 글자를 대략 검출할 수 있다.

물론 여기서 끝이 아니라, 더 고도화 하는 방법이 있으나 그건 나중에 다루도록 한다.

CHAPTER 8
기하학 처리

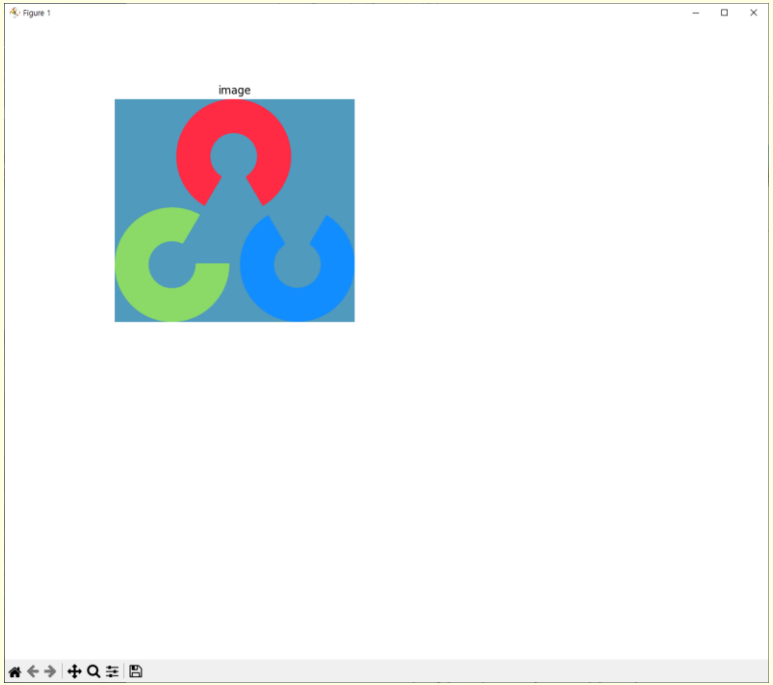
김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

8. 기하학 처리

8.6 어파인 변환

8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

```
import numpy as np, cv2
import matplotlib.pyplot as plt

image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

titles = ['image']
plt.figure(figsize=(12, 10))
for idx, title in enumerate(titles):
    plt.subplot(2, 2, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title))
plt.show()
```



어파인 변환이라고도 하고, 아핀 변환이라고도 함

비례를 유지하는 기하학적 함수

어파인 변환은 크기가 2x3 이어야 함

8. 기하학 처리

8.6 어파인 변환

8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

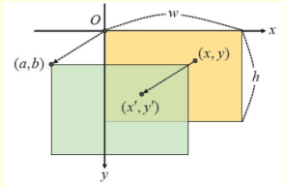
```
...
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
h, w, c = image.shape

a, b = -100, 200
# x축으로 -100, y축으로 200 이동
mov_M = np.array([[1, 0, a],
                  [0, 1, b]], dtype=np.float32)

image_move = cv2.warpAffine(image, mov_M, (w, h))
```

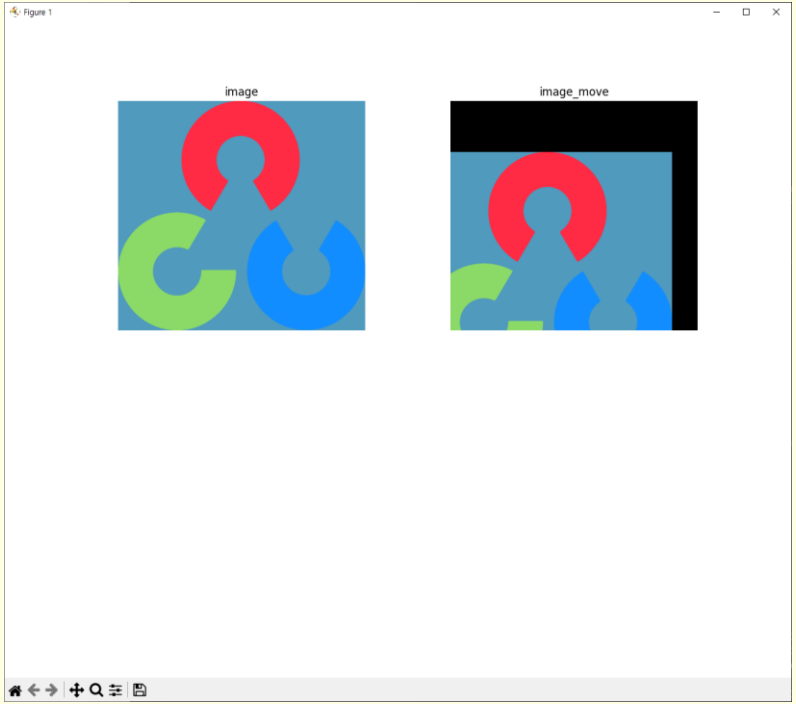
cv2.warpAffine(src, M, dsize, ...)
src = 입력 영상
M = 어파인 변환 행렬
dsize = 반환 영상의 크기

```
titles = ['image', 'image_move']
...
```

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \end{bmatrix}$$



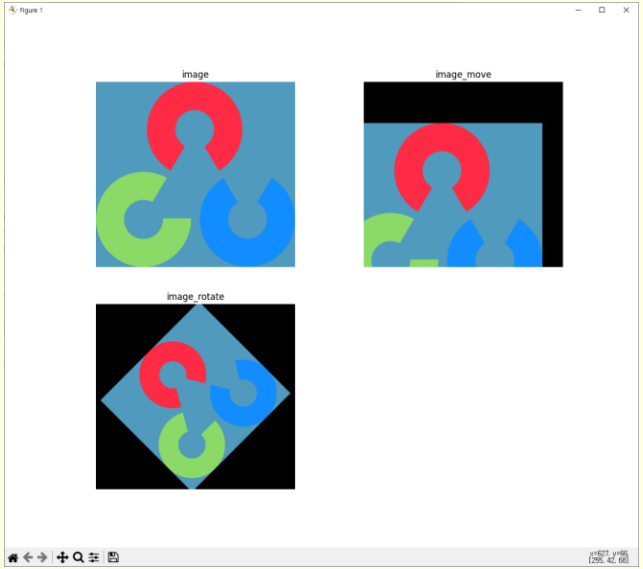


8. 기하학 처리

8.6 어파인 변환

8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

```
...
image_move = cv2.warpAffine(image, mov_M, (w, h))

center = (w // 2, h // 2)
scale = 0.7
angle = 45
theta = np.radians(45)
alpha = scale * np.cos(theta)
beta = scale * np.sin(theta)
rot_M = np.array([[alpha, beta, (1-alpha)*center[0] - beta*center[1]],
                  [-beta, alpha, beta*center[0] + (1-alpha)*center[1]]],
                  dtype=np.float32)
image_rotate = cv2.warpAffine(image, rot_M, (w, h))

titles = ['image', 'image_move', 'image_rotate']
...
```



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

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

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

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

8. 기하학 처리

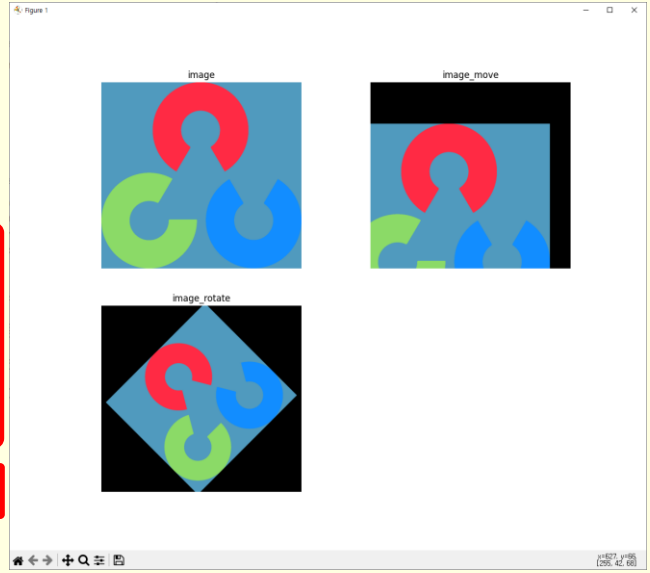
8.6 어파인 변환

8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

```
...
image_move = cv2.warpAffine(image, mov_M, (w, h))

center = (w // 2, h // 2)
scale = 0.7
angle = 45
theta = np.radians(45)
alpha = scale * np.cos(theta)
beta = scale * np.sin(theta)
rot_M = np.array([[alpha, beta, (1-alpha)*center[0] - beta*center[1]],
                  [-beta, alpha, beta*center[0] + (1-alpha)*center[1]]],
                  dtype=np.float32)
=
rot_M = cv2.getRotationMatrix2D(center, angle, scale)
```

OpenCv에서 더 간단하게 사용할 수 있는 함수를 제공

```
image_rotate = cv2.warpAffine(image, rot_M, (w, h))

titles = ['image', 'image_move', 'image_rotate']
...
```



8. 기하학 처리

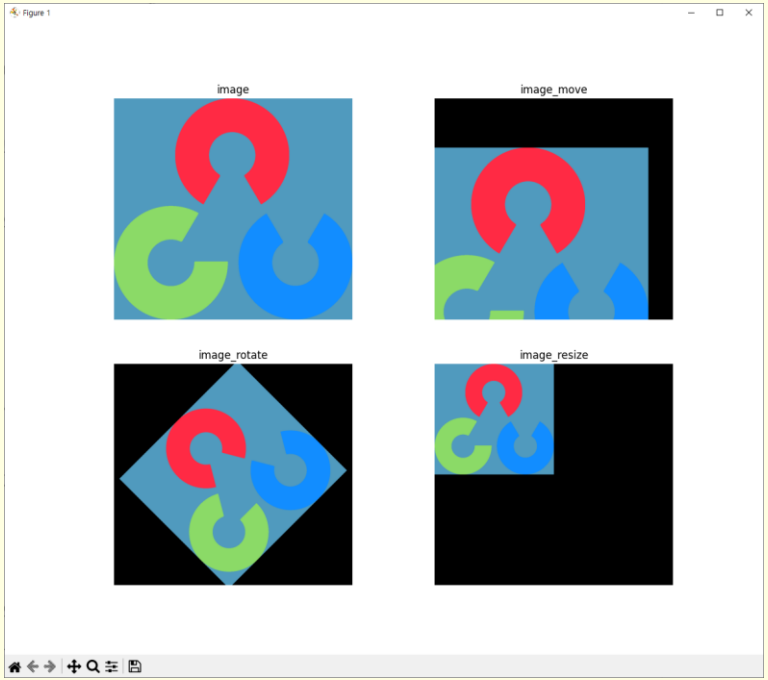
8.6 어파인 변환

8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

```
...
image_rotate = cv2.warpAffine(image, rot_M, (w, h))

s_x, s_y = 0.5, 0.5
resize_M = np.array([[s_x, 0, 0],
                     [0, s_y, 0]], dtype=np.float32)
image_resize = cv2.warpAffine(image, resize_M, (w, h))

titles = ['image', 'image_move', 'image_rotate', 'image_resize']
...
```

$$\begin{cases} x' = s_x x \\ y' = s_y y \end{cases} \quad or \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & x_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$$



8. 기하학 처리

■ 8.6 어파인 변환

◆ 8.6.1 어파인 변환을 이용한 이동, 회전, 크기 변환

```
...
image_rotate = cv2.warpAffine(image, rot_M, (w, h))

s_x, s_y = 0.5, 0.5
resize_M = np.array([[s_x, 0, 0],
                     [0, s_y, 0]], dtype=np.float32)
image_resize = cv2.warpAffine(image, resize_M, (w, h))
```

1. 직접 수식을 지정하는 방법

Scale 지정

```
resize_M = cv2.getRotationMatrix2D(center, 0, 0.5)
image_resize = cv2.warpAffine(image, resize_M, (w, h))
```

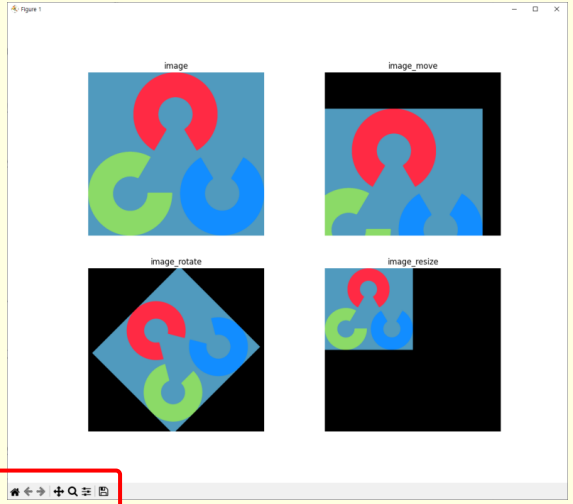
2. 회전변환 함수에서 지정하는 방법

```
image_resize = np.zeros((h, w, c), dtype=np.uint8)
resize_M = cv2.resize(image, (0, 0), fx=0.5, fy=0.5, interpolation=cv2.INTER_LINEAR)
image_resize[:resize_M.shape[0], :resize_M.shape[1]] = resize_M
```

선형 보간법 사용

3. 어파인 변환을 안쓰고, resize로 구현하는 방법

```
titles = ['image', 'image_move', 'image_rotate', 'image_resize']
...
```

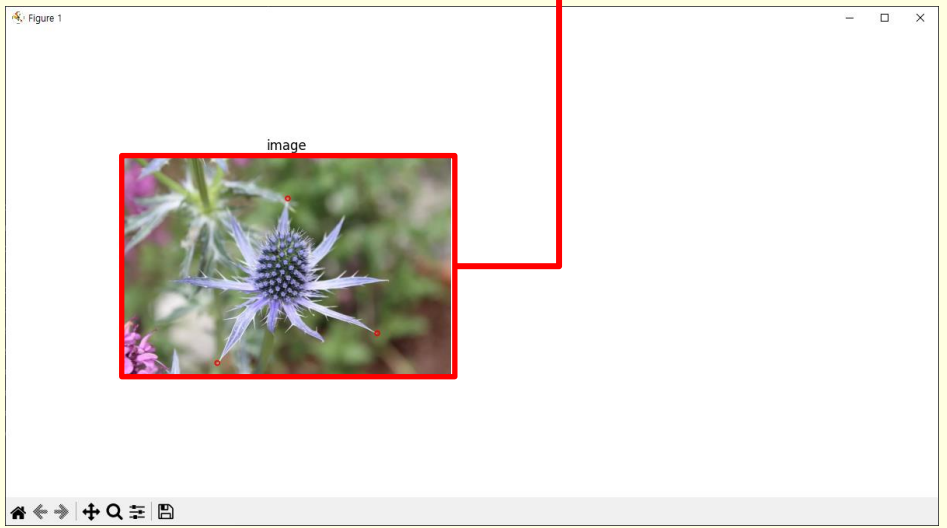


8. 기하학 처리

8.6 어파인 변환

8.6.2 일반화된 어파인 변환

```
import numpy as np, cv2
import matplotlib.pyplot as plt

image = cv2.imread("이미지 경로", cv2.IMREAD_COLOR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
h, w, c = image.shape

titles = ["image"]
plt.figure(figsize=(12, 6))
for idx, title in enumerate(titles):
    plt.subplot(1, 2, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title))
plt.show()
```



8. 기하학 처리

■ 8.6 어파인 변환

◆ 8.6.2 일반화된 어파인 변환


```
...
h, w, c = image.shape

pt1 = np.array([[772, 196], [1190, 824], [443, 962]], np.float32)
# 원본 이미지에서 이미지를 변환할 좌표 설정
pt2 = np.array([[275, 503], [1286, 128], [1332, 893]], np.float32)
# 변환된 이미지의 좌표 설정
for pt in pt1:
    cv2.circle(image, tuple(pt.astype(int)), 10, (255, 0, 0), 5)
    1. 입력 영상
    2. 원의 중심 좌표
    3. 원의 반지름
    4. 원의 색상
    5. 원의 두께

aff_mat = cv2.getAffineTransform(pt1, pt2)
affine = cv2.warpAffine(image, aff_mat, (w, h))

titles = [“image”, “affine”]
...
```

지정된 좌표에 따라 약 100도 회전
정확히 말하면 회전이 아닌 3점 어파인 변환