CHAPTER 8
기하학 처리

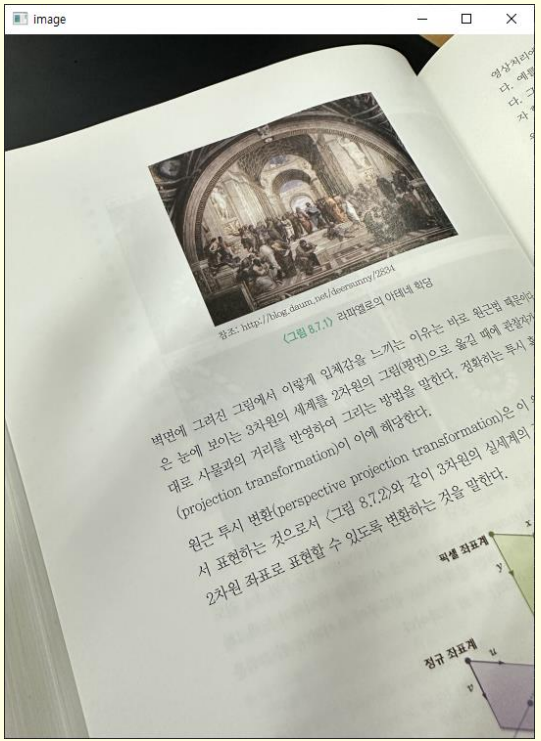
김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```python
import numpy as np, cv2

# 이미지 경로만 입력하면 기본 값으로 COLOR 이미지로 받아 옴
image = cv2.imread("이미지 경로")

# 이미지 크기 조정
image = cv2.resize(image, (image.shape[1]//2, image.shape[0]//2))

# 각 네모 박스의 크기 선언
small = np.array([12, 12])

# 마우스를 클릭했는지 안 했는지 검사하기 위한 변수
check = -1

# 각 네모 박스의 위치 초기 선언
pts1 = np.float32([(100, 100), (300, 100), (300, 300), (100, 300)])

# 이미지 정상 출력 확인
cv2.imshow("image", image)
cv2.waitKey(0)
```



8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```python
# 원본 이미지를 그리기 위한 함수
def draw_rect(img):
    # 1줄 for문
    # 현재 4개의 네모 박스들 위치에 small로 지정한 범위 적용
    rois = [(p - small, small * 2) for p in pts1]

    # 현재 4개의 네모 박스들 위치를 for each문으로 반복
    # np.int32를 준 이유는 좌표 값은 소수점이면 안되기 때문
    for (x, y), (w, h) in np.int32(rois):
        # 현재 좌표에 맞춰 네모 그리기
        cv2.rectangle(img, (x, y, w, h), (0, 255, 0), 1)


    # 각 네모 박스의 좌표를 선으로 이어 줌
    cv2.polylines(img, [pts1.astype(int)], True, (0, 255, 0), 1)


    # 이미지 출력
    cv2.imshow("select rect", img)
```

같은 기능

```python
    rois = []
    for p in pts1:
        rois.append((p-small, small * 2))
```

cv2.rectangle
1. 입력 영상
2. 사각형의 위치, 사각형의 크기
3. 색상
4. 두께

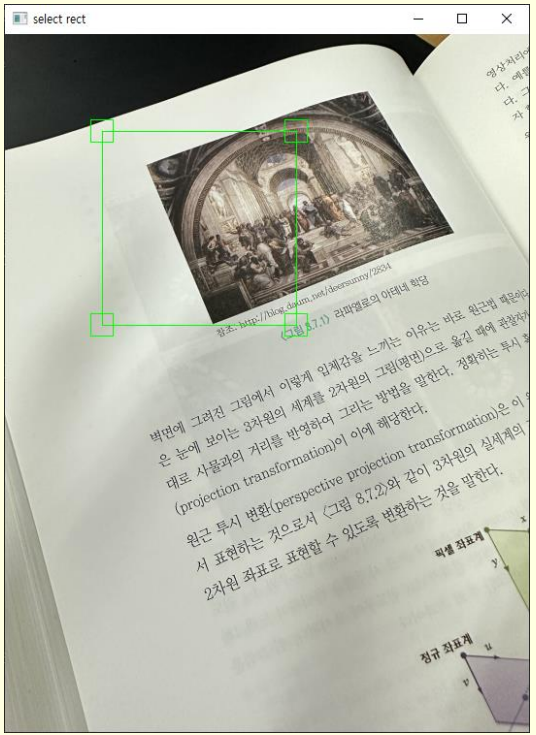
cv2.polylines
1. 입력 영상
2. 도형들의 위치 좌표
3. F = 열린 도형 / T = 닫힌 도형
4. 색상
5. 두께

8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```
...
# 각 네모 박스의 위치 초기 선언
pts1 = np.float32([(100, 100), (300, 100), (300, 300), (100, 300)])

# 원본 이미지를 띄우기 위한 함수 호출
draw_rect(np.copy(image))

# 이미지 정상 출력 확인용 이었으니, 제거
cv2.imshow("image", image)

# 키 입력 대기
cv2.waitKey(0)
```



8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```python
def onMouse(event, x, y, flags, param):
    global check

    # 만약 마우스 왼쪽 버튼을 눌렀다면
    if event == cv2.EVENT_LBUTTONDOWN:
        for i, p in enumerate(pts1):
            # 마우스를 클릭했을때 박스 안의 범위를 적용시켜서 현재위치 반환
            p1, p2 = p - small, p + small
            # 만약 현재 마우스 위치가 박스의 위치와 같다면
            if contain((x,y), p1, p2):
                # 몇 번째 박스가 클릭 됐는지 check에 저장
                check = i

    # Mouse Event에 Drag 이벤트는 지원하지 않으니, 이렇게 구현
    if check >= 0:
        # 클릭한 박스를 움직이기 위해 해당 좌표만 변경
        pts1[check] = (x, y)
        # 클릭한 박스를 현재 위치에 맞게 그리기 위해 생성
        draw_rect(np.copy(image))
```

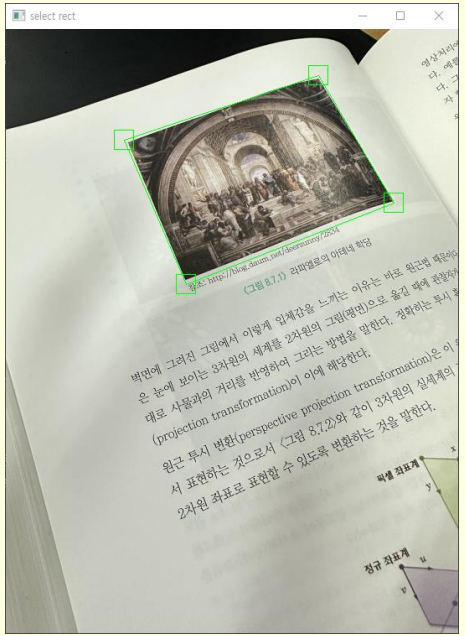
현재 박스 위치
좌표 반복 순서

8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```python
# 생성된 네모 박스를 잡아 끌 수 있도록 기능 모듈화
def contain(p, p1, p2):
    # x, y의 위치가 네모박스 안에 있는지 비교 문 사용
    return p1[0] <= p[0] < p2[0] and p1[1] <= p[1] < p2[1]

def onMouse(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        ...

    # 마우스에서 손 떼면 더이상 박스 안 따라다니게 변경
    if event == cv2.EVENT_LBUTTONUP:
        check = -1

    if check >= 0:
        ...
...
draw_rect(np.copy(image))

# 원본 이미지가 띄워져 있는 Window에 마우스 콜백함수 적용
cv2.setMouseCallback("select rect", onMouse, 0)
cv2.waitKey(0)
```



cv2.setMouseCallback
1. 이미 생성 되어있는 Window
2. 마우스 이벤트 발생 시 전달 함수
3. 마우스 이벤트와 함께 전달할 param

-> 이미지를 전달하기도 하는데
이번 실습에서는 좌표 값으로
투영 실습을 하기 때문에 사용 X

8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```python
def warp(img):
    # 이동 전 좌표, 이동 후 좌표를 주면 투시 변환 행렬을 반환하는 함수
    perspect_mat = cv2.getPerspectiveTransform(pts1, pts2)

    # 반환될 이미지
    dst = cv2.warpPerspective(img, perspect_mat, (400, 350), cv2.INTER_CUBIC)

def onMouse(event, x, y, flags, param):
    ...
    if check >= 0:
        ...
        draw_rect(np.copy(image))

        # 현재 박스(범위)에 따라 투영된 이미지를 출력
        warp(np.copy(image))
```

cv2.getPerspectiveTransform
1. 이동 전 좌표
2. 이동 후 좌표

cv2.warpPerspective
1. 입력 영상
2. 투시 변환 행렬
3. 결과 영상 크기
4. 보간법

cv2.INTER_LINEAR: 양선형 보간법
4개의 픽셀(2x2 이웃 픽셀)을 참조
처리속도 빠름, 퀄리티 보통

cv2.INTER_CUBIC: 3차 회선 보간법
16개의 픽셀(4x4 이웃 픽셀)을 참조
처리 속도 느림, 퀄리티 좋음

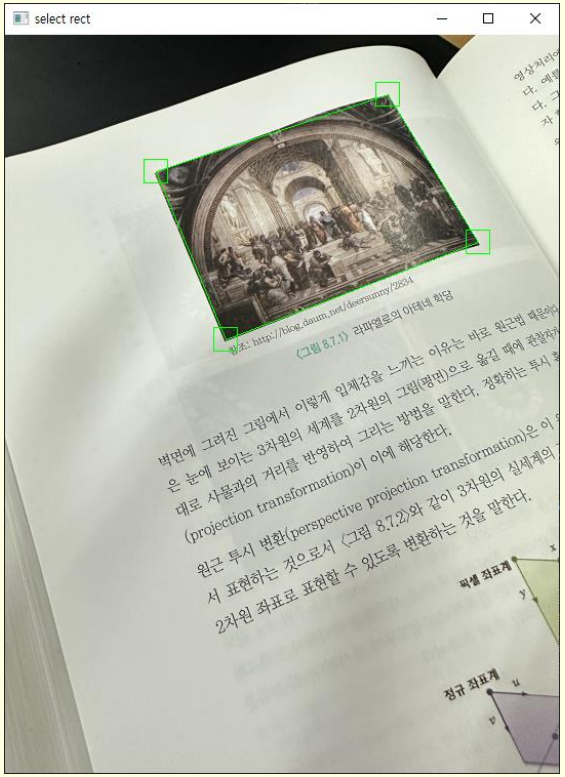
cv2.INTER_LANCZOS4: Lanczos 보간법
64개의 픽셀(8x8 이웃 픽셀)을 참조
처리속도 매우 느림, 퀄리티 매우 좋음

8. 기하학 처리

■ 8.7 원근 투시(투영) 변환

◆ 8.7.2 마우스 이벤트를 활용한 투영 변환

```
def warp(img):
    ...

    # 이미지 출력
    cv2.imshow("perspective transform", dst)
```



CHAPTER 9
변환영역 처리

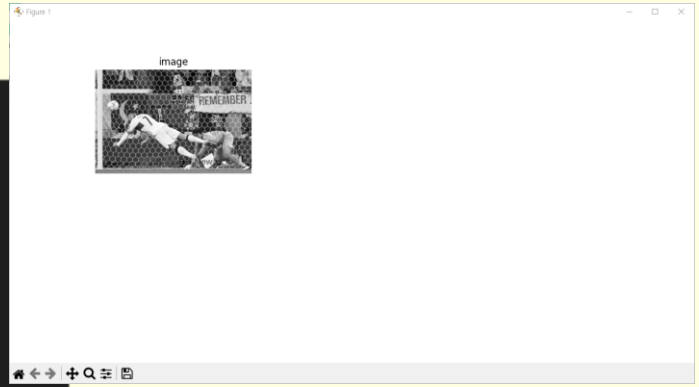
김 찬, 윤 영 선
ckim.esw@gmail.com, ysyun@hnu.kr
정보통신공학과

9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

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

# 이미지 읽기
image = cv2.imread("이미지 경로", cv2.IMREAD_GRAYSCALE)

# 반복문으로 이미지 한번에 출력
titles = ['image']
plt.figure(figsize=(12, 6))
for idx, title in enumerate(titles):
    plt.subplot(2, 3, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title), cmap='gray')
plt.show()
```



9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

```python
# 고속 푸리에 변환
def FFT(image):
    # 푸리에 변환
    dft = cv2.dft(np.float32(image), flags=cv2.DFT_COMPLEX_OUTPUT)

    # 주파수 시프트
    dft = fftshift(dft)

    # 주파수 스펙트럼 영상
    spectrum = calc_spectrum(dft)

    return dft, spectrum

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

# 행렬 중심점 구하기
cy, cx = np.divmod(image.shape, 2)[0]

# FFT 수행 및 셔플링
dft, spectrum = FFT(image)
```

cv2.dft
1. 실수로 변환된 입력 영상
2. 반환 행렬 타입

cv2.DFT_COMPLEX_OUTPUT:
2채널 복소수 행렬로 변환

9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

```python
# fft_shift를 진행할 때 중복으로 사용되는 코드 재활용
def shift_info(img):
    dst = np.zeros(img.shape, img.dtype)
    h, w = dst.shape[:2]
    cx, cy = w//2, h//2

    # 영상의 크기가 짝수일 때는 괜찮은데, 홀수인 경우 왜곡이 발생
    # fftshift와 ifftshift로 구분 구현
    xo = 0 if w % 2 == 0 else 1
    yo = 0 if h % 2 == 0 else 1
    return cx, cy, xo, yo, dst

def fftshift(img):
    cx, cy, xo, yo, dst = shift_info(img)
    dst[cy:, cx:] = np.copy(img[0:cy+yo, 0:cx+xo])  # 1사분면 -> 3사분면
    dst[0:cy, 0:cx] = np.copy(img[cy+yo:, cx+xo:])  # 3사분면 -> 1사분면
    dst[0:cy, cx:] = np.copy(img[cy+yo:, 0:cx+xo])  # 2사분면 -> 4사분면
    dst[cy:, 0:cx] = np.copy(img[0:cy+yo, cx+xo:])  # 4사분면 -> 2사분면
    return dst

def FFT(image):
```

같은 기능

```python
if w%2 == 0:
    xo = 0
else:
    xo = 1
```

9. 변환영역 처리

9.4 FFT를 이용한 주파수 영역 필터링

9.4.1 주파수 영역 필터링

```python
def fftshift(img):
    ...

def calc_spectrum(complex):
    # 만약 이미지가 2차원 행렬이라면
    if complex.ndim == 2:
        # 복소수 객체 행렬을 실수 행렬로 변환
        dst = abs(complex)
    # 만약 이미지가 3차원 행렬이라면
    else:
        dst = cv2.magnitude(complex[:,:,0], complex[:,:,1])

    dst = cv2.log(dst + 1)

    # 이미지 정규화
    cv2.normalize(dst, dst, 0, 255, cv2.NORM_MINMAX)

    # 모든 값을 절대값화 시키고 정수화
    return cv2.convertScaleAbs(dst)

def FFT(image):
```

cv2.magnitude
1. 2D 벡터의 x좌표를 나타내는 행렬
2. 2D 벡터의 y좌표를 나타내는 행렬

cv2.normalize
1. 정규화 이전의 데이터
2. 정규화 이후의 데이터
3. 정규화 구간 1 (alpha)
4. 정규화 구간 2 (beta)
5. 정규화 알고리즘

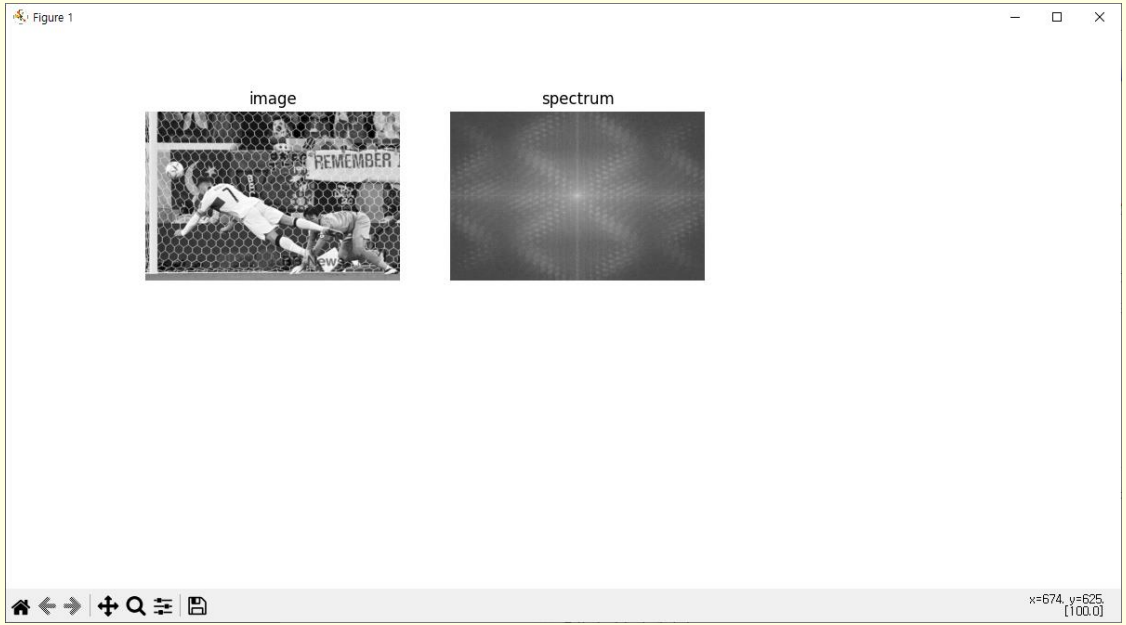
cv2.NORM_MINMAX:
alpha와 beta의 구간으로 정규화

9. 변환영역 처리

9.4 FFT를 이용한 주파수 영역 필터링

9.4.1 주파수 영역 필터링

```
def FFT(image):
    ...


...
dft, spectrum = FFT(image)

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



9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

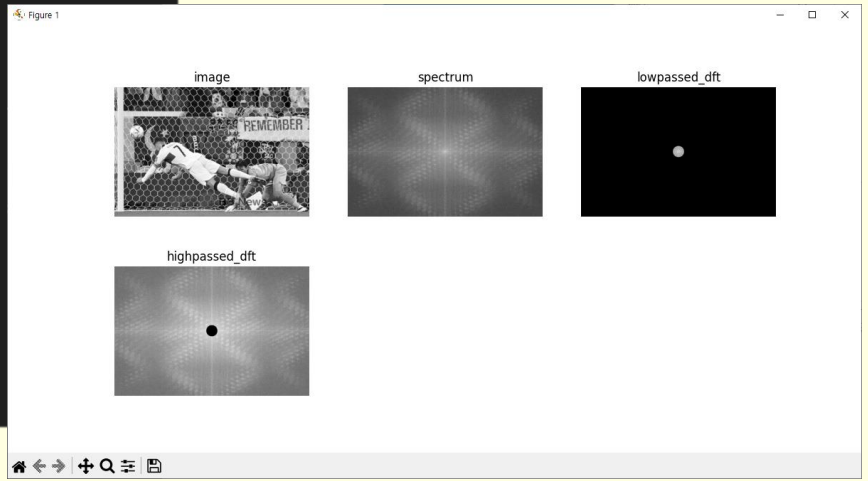
```
def FFT(image):
    ...

...
dft, spectrum = FFT(image)

# 저주파, 고주파 통과 필터
Lowpass = np.zeros(dft.shape, np.float32)
Highpass = np.ones(dft.shape, np.float32)

# 2개 채널로 값 지정
cv2.circle(lowpass, (cx, cy), 30, (1, 1), -1)
cv2.circle(highpass, (cx, cy), 30, (0, 0), -1)

# 주파수 필터링 = 주파수 계수 * 필터행렬
lowpassed_dft = dft * lowpass
highpassed_dft = dft * highpass

titles = ['image', 'spectrum', 'lowpassed_dft', 'highpassed_dft’]
...
```

9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

```
titles = ['image', 'spectrum', 'lowpassed_dft', 'highpassed_dft']
...

plt.figure(figsize=(12, 6))
for idx, title in enumerate(titles):
    plt.subplot(2, 3, idx+1)
    plt.axis("off")
    plt.title(title)
    if idx == 2 or idx == 3:
        plt.imshow(calc_spectrum(eval(title)), cmap='gray')
    else:
        plt.imshow(eval(title), cmap='gray')
plt.show()
```



9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

```
# 역 고속 푸리에 변환
def IFFT(dft, shape):
    # 주파수 영역에서 원래 영상으로 변환
    dft = ifftshift(dft)

    # 역 푸리에 변환
    img = cv2.idft(dft, flags=cv2.DFT_SCALE)[:,:,0]

    # 영삽입(zero-padding) 부분 제거
    img = img[:shape[0], :shape[1]]

    # 절대값 및 uint8 스케일링
    return cv2.convertScaleAbs(img)

...
highpassed_dft = dft * highpass

# 푸리에 역변환
lowpassed_img = IFFT(lowpassed_dft, image.shape)
highpassed_img = IFFT(highpassed_dft, image.shape)
```

cv2.idft
1. 실수로 변환된 입력 영상
2. 반환 행렬 타입

cv2.idft의 결과 행렬은 2개이다.
[:,:,0] = 실수부
[:,:,1] = 허수부

이 중에 실수부만 사용할 것이기 때문에 [:,:,0]을 사용하는 것

9. 변환영역 처리

■ 9.4 FFT를 이용한 주파수 영역 필터링

◆ 9.4.1 주파수 영역 필터링

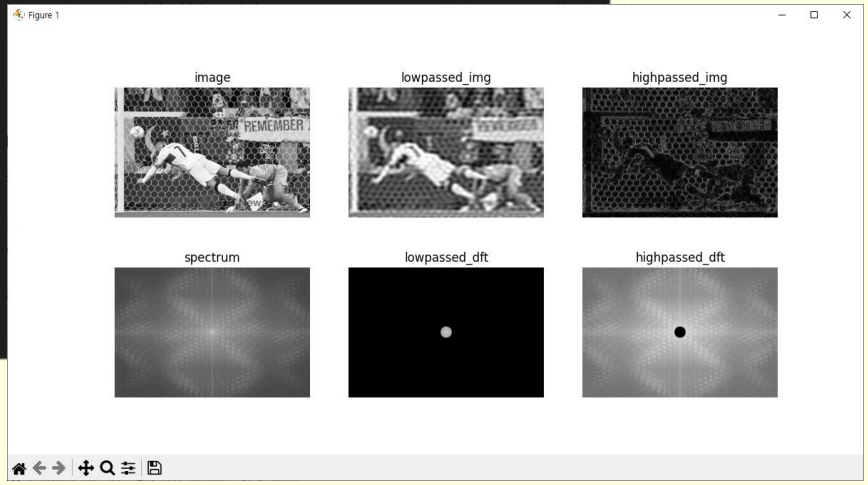
```
def fftshift(img):
    ...

def ifftshift(img):
    cx, cy, xo, yo, dst = shift_info(img)
    dst[cy+yo:, cx+xo:] = np.copy(img[0:cy, 0:cx])  # 1사분면 -> 3사분면
    dst[0:cy+yo, 0:cx+xo] = np.copy(img[cy:, cx:])  # 3사분면 -> 1사분면
    dst[0:cy+yo, cx+xo:] = np.copy(img[cy:, 0:cx])  # 2사분면 -> 4사분면
    dst[cy+yo:, 0:cx+xo] = np.copy(img[0:cy, cx:])  # 4사분면 -> 2사분면
    return dst

def calc_spectrum(complex):
    ...
```

9. 변환영역 처리

9.4 FFT를 이용한 주파수 영역 필터링

9.4.1 주파수 영역 필터링

```
...
highpassed_img = IFFT(highpassed_dft, image.shape)

# 반복문으로 이미지 한번에 출력
titles = ['image', 'lowpassed_img', 'highpassed_img', 'spectrum', 'lowpassed_dft', 'highpassed_dft']
plt.figure(figsize=(12, 6))
for idx, title in enumerate(titles):
    plt.subplot(2, 3, idx+1)
    plt.axis("off")
    plt.title(title)
    if idx < 4:
        plt.imshow(eval(title), cmap='gray')
    else:
        plt.imshow(calc_spectrum(eval(title)), cmap='gray')
plt.show()
```



CHAPTER 10
영상 분할 및 특징 처리

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

10. 영상 분할 및 특징 처리

■ 10.1 허프 변환

◆ 10.1.1 허프 변환을 이용한 직선 검출

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

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

# 반복문으로 이미지 한번에 출력
titles = ['image']
plt.figure(figsize=(6, 10))
for idx, title in enumerate(titles):
    plt.subplot(3, 1, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title), cmap='gray')
plt.show()
```

10. 영상 분할 및 특징 처리

10.1 허프 변환

10.1.1 허프 변환을 이용한 직선 검출

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

# 가우시안 블러링
blur = cv2.GaussianBlur(image, (5, 5), 2, 2)

# 캐니 엣지 검출
canny = cv2.Canny(blur, 100, 200, 5)

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

cv2.GaussianBlur
1. 입력 영상
2. 가우시안 커널 크기
3. x방향 sigma
4. y방향 sigma

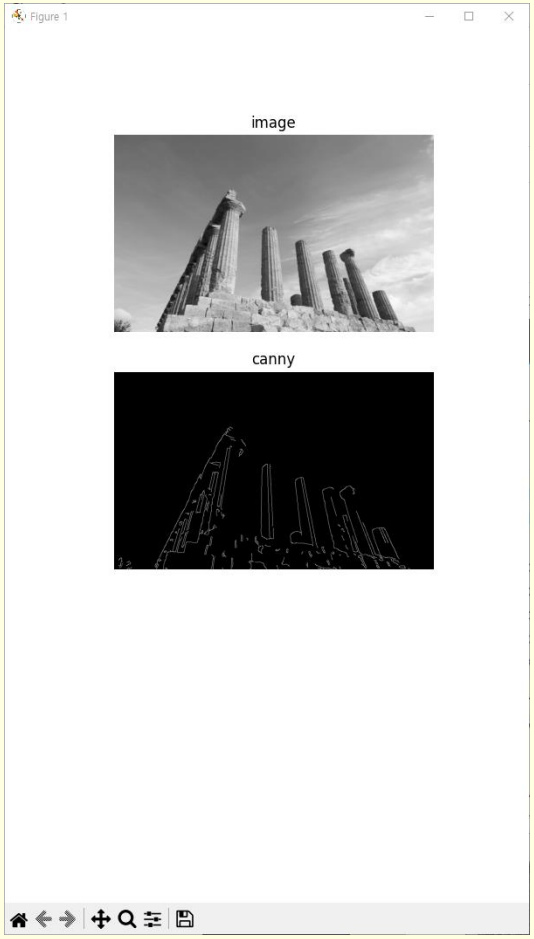
-> 값을 안주면 x방향 sigma의 값을 따라감

cv2.Canny
1. 입력 영상
2. threshold1

-> 엣지가 되기 쉬운 부분에 있어 엣지인지 아닌지 판단하는 임계값

3. threshold2

-> 엣지인지 아닌지 판단하는 임계값



10. 영상 분할 및 특징 처리

■ 10.1 허프 변환

◆ 10.1.1 허프 변환을 이용한 직선 검출

```
...
canny = cv2.Canny(blur, 100, 200, 5)

# 수직거리 간격, 각도 간격
rho, theta = 1, np.pi / 180

# 허프 변환 직선 검출
lines = cv2.HoughLines(canny, rho, theta, 20)

# 직선 그리기
dst = draw_hough_lines(image, lines, 7)

...
```

cv2.HoughLines
1. 입력 영상
-> 8bit, sigle-channel(binary), canny edge 선 적용
2. rho: 0 ~ 1 범위의 실수형
3. theta: 0 ~ 180 범위의 정수형
4. threshold: 만나는 점의 기준
-> 숫자가 작으면 많은 선이 검출 단, 정확도 떨어짐
-> 숫자가 크면 보다 적은 선이 검출 정확도 올라감

10. 영상 분할 및 특징 처리

■ 10.1 허프 변환

◆ 10.1.1 허프 변환을 이용한 직선 검출

```python
def draw_hough_lines(src, lines, nline):
    dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)
    min_length = min(len(lines), nline)

    for i in range(min_length):
        # 수직거리, 각도
        rho, radian = lines[i, 0, 0:2]
         # x, y축에 대한 삼각비
        a, b = math.cos(radian), math.sin(radian)
        # x, y축에 기준(절편) 좌표
        pt = (a * rho, b * rho)
        # 직선상의 이동 위치
        delta = (-1000 * b, 1000 * a)
        # 직선의 방정식으로 그리기 위한 시작점 계산
        pt1 = np.add(pt, delta).astype('int')
        # 직선의 방정식으로 그리기 위한 끝점 계산
        pt2 = np.subtract(pt, delta).astype('int')
        cv2.line(dst, tuple(pt1), tuple(pt2), (255, 0, 0), 2, cv2.LINE_AA)
    return dst
image = cv2.imread("이미지 경로", cv2.IMREAD_GRAYSCALE)
...
```

극 좌표: (r, theta)
직교좌표: (x, y)

직선의 방정식으로 선을 그리기 위해
극좌표를 직교좌표로 변환 해 줌

np.add
-> 직선 위 2개 좌표 + 직선 이동 위치
np.add(pt, delta)
= ((a * rho) + (-1000 * b)),
  ((b * rho) + (1000 * a))

np.subtract
-> 직선 위 2개 좌표 - 직선 이동 위치
np.subtract(pt, delta)
= ((a * rho) - (-1000 * b)),
  ((b * rho) - (1000 * a))

cv2.line
1. 입력 영상
2. 시작점 좌표 (x, y)
3. 종료점 좌표 (x, y)
4. 색상
5. 두께
6. 선 종류

10. 영상 분할 및 특징 처리

10.1 허프 변환

10.1.1 허프 변환을 이용한 직선 검출

```
def draw_hough_lines(src, lines, nline):
    ...

...
dst = draw_hough_lines(image, lines, 7)

# 반복문으로 이미지 한번에 출력
titles = ['image', 'canny', 'dst']
plt.figure(figsize=(6, 10))
for idx, title in enumerate(titles):
    plt.subplot(3, 1, idx+1)
    plt.axis("off")
    plt.title(title)
    plt.imshow(eval(title), cmap='gray')
plt.show()
```