<A DCT-domain system for robust image watermarking>

1. **def water_marking( )** - original image에 watermark 삽입

```
def water_marking(org_image, M, alpha):
    # 0~1 사이 float 형 image 로 변경하여 dct 변환
    image = np.float32(org_image) / 255.0
    dct_image = cv2.dct(image)

    #0~255 범위 zig-zag 스캔
    zig_zag = []
    for i in range(255):
        for j in range(i + 1):
            if i % 2 == 0:
                zig_zag.append(dct_image[i - j, j])
            else:
                zig_zag.append(dct_image[j, i - j])
    zig_zag = np.asarray(zig_zag)
    # zig_zag_picked : watermarking 할 M ~ 2M 범위 slicing
    # ( [L+1] ~ [L + M], L = M ) -> [M : 2 * M]
    zig_zag_picked = zig_zag[M : 2 * M]

    # normal distribution 랜덤 변수 x 생성 (watermark)
    x = np.random.normal(0, 1, size=M)

    # zig_zag_picked 에 watermark x 적용
    for i in range(len(zig_zag_picked)):
        # t'_{L+1} = t_{L+1} + α|t_{L+1}|x_i
        zig_zag_picked[i] += (alpha * np.abs(zig_zag_picked[i]) * x[i])
    zig_zag[M : 2 * M] = zig_zag_picked

    # watermarking 된 값 다시 zig_zag input
    index = 0
    for i in range(255):
        for j in range(i + 1):
            if i % 2 == 0:
                dct_image[i - j, j] = zig_zag[index]
            else:
                dct_image[j, i - j] = zig_zag[index]
            index += 1
    # inverse dct 하여 다시 image 로 변환
    inv_dct = cv2.idct(dct_image)

    return inv_dct, x
```

2. **def detect_watermark( )** – watermarking된 이미지에 y 이용하여 watermark detect

```
def detect_watermark(image, y, M, alpha):
    # 0~1 사이 float 형 image 로 변경하여 dct 변환
    image = np.float32(image) / 255.0
    dct_image = cv2.dct(image)

    #0~255 범위 zig-zag 스캔
    zig_zag = []
    for i in range(255):
        for j in range(i + 1):
            if i % 2 == 0:
                zig_zag.append(dct_image[i - j, j])
            else:
                zig_zag.append(dct_image[j, i - j])
    #zig-zag 스캔한 0~255 범위 중 M~2M 범위 slicing
    zig_zag = np.asarray(zig_zag)[M : 2 * M]

    # zig_zag 의 평균 값과 alpha 값 이용하여 watermark 검출 기준 threshold 설정
    threshold = (alpha / 2) * np.mean(np.abs(zig_zag))

    # detect list 에 np.mean(y[i] * zig_zag)를 append
    # - watermarked image 를 지그재그 스캔한 zig_zag list 와,
    # 길이가 M 인 정규분포 난수 1000 개를 각각 곱해 평균낸 값
    # 총 1000 개를 detect list 에 담는다
    detect = []
    for i in range(len(y)):
        # z = \frac{1}{M}\sum_{i=1}^{M} y_i(t_i + \alpha|t_i|x_i)
        detect.append(np.mean(y[i]*zig_zag))

    return detect, threshold
```

Fig2. Original image and watermarked image with parameters, Alpha = 0.2, M = L = 16000

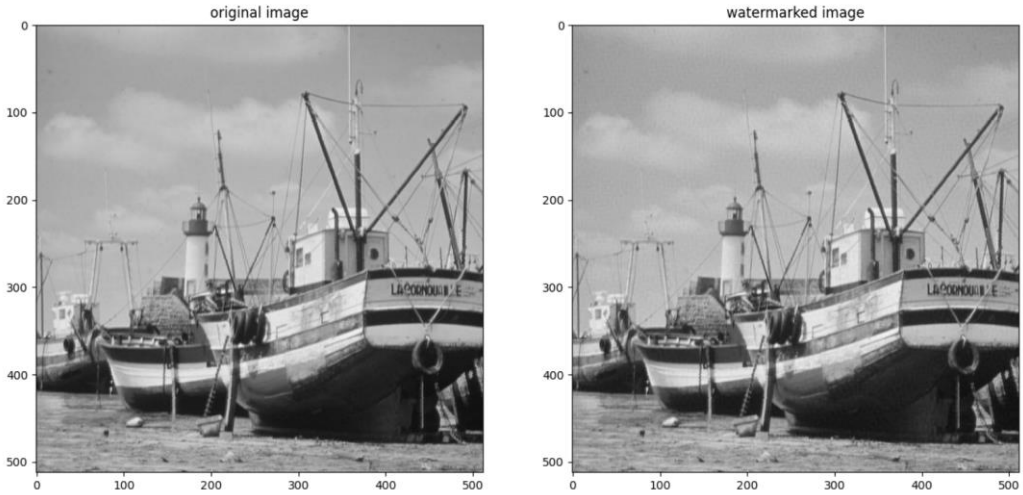

Fig3. The Detector Response of the watermarked image. (index : [100])

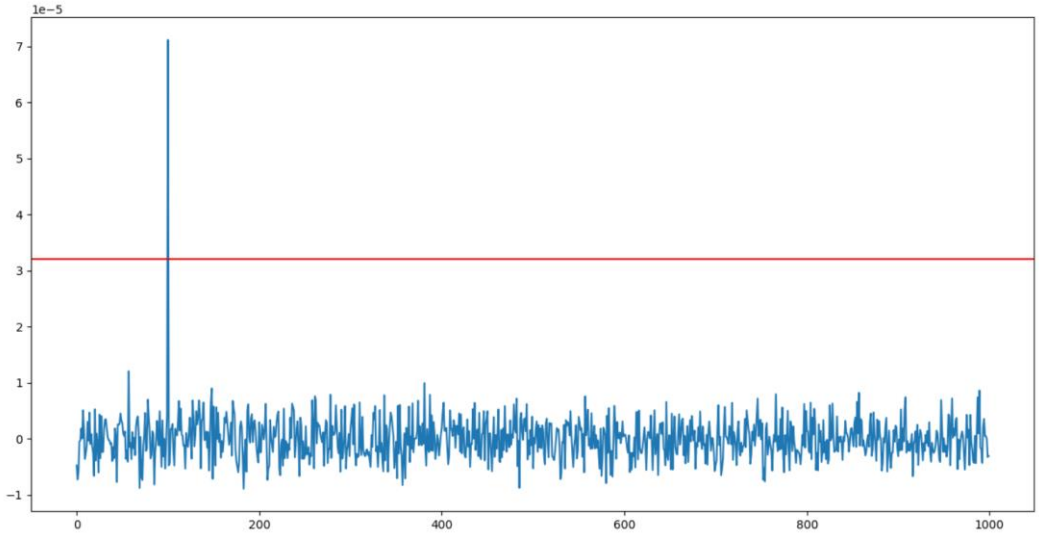

Fig5. Watermarked image 'Boat' low pass filtered 5x5 (Left), and the corresponding of the detector response

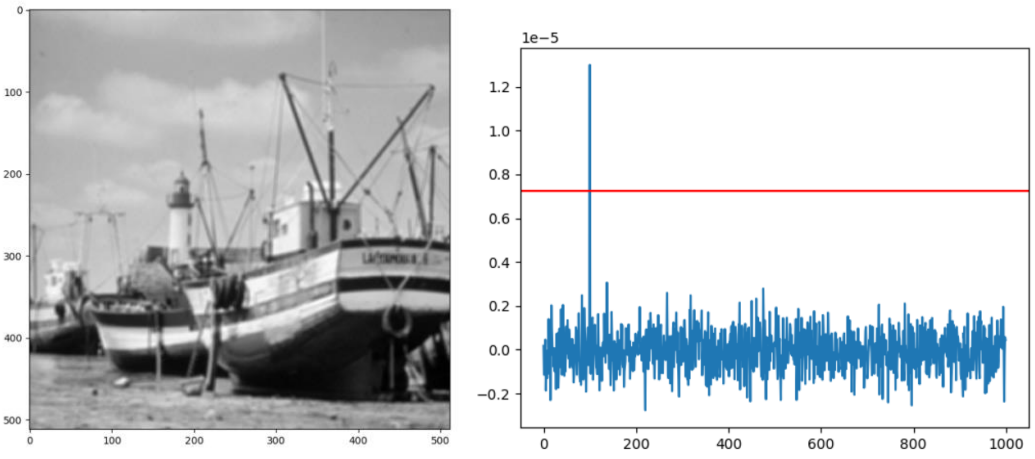

Fig14. Image 'Boat' with five different watermarks, and the corresponding of the detector response

➔ index : [100], [300], [500], [700], [900]