디지털영상처리

과제2 – 동영상 화질 개선 편집 프로그램

2021. 11. 04. 목

컴퓨터공학과

2019305059

이현수

목차

1. 문제 정의

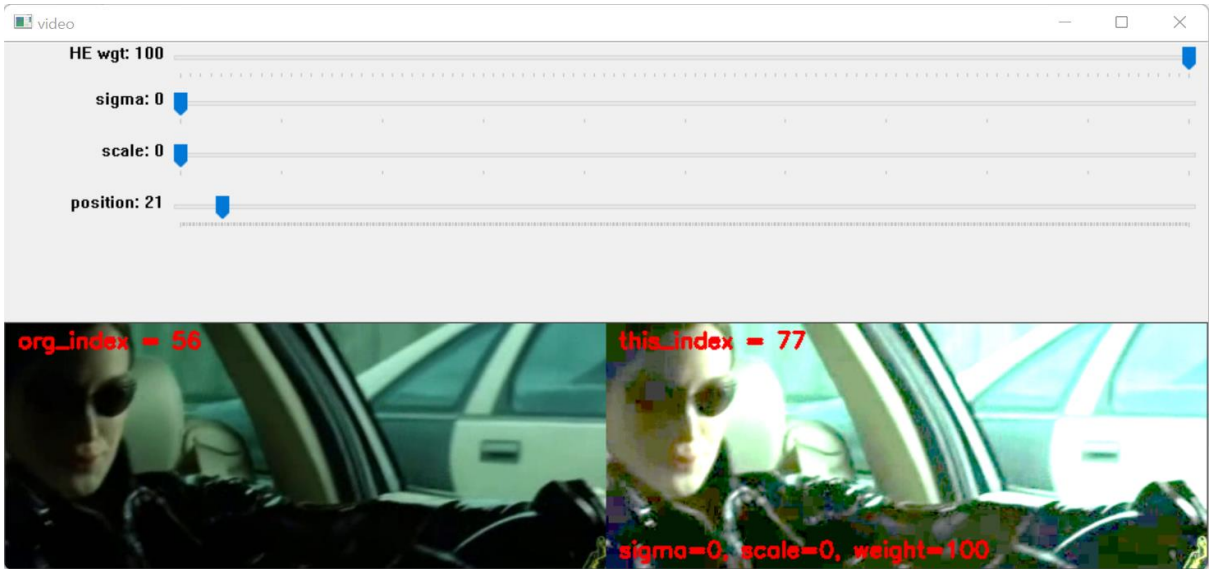
2. 문제풀이 절차

3. 소스코드

4. 실행결과

5. 시행착오, 배운점, 기술적의의

1. 문제 정의



입력 원본 프레임 영상에 대해 히스토그램 평활화와 unsharp masking을 행한다.

- 히스토그램 평활화를 행할 때 트랙바의 HE wgt 값(0~100)을 조절해 평활화 영상의 반영비율을 0~100%까지 선택할 수 있다. Wgt 값이 0이라면 히스토그램 평활화를 하지 않은 원본영상이다.

- 히스토그램 평활화로 개선된 영상에 대해 unsharp masking을 행한다. 이때 트랙바를 통해 UM의 시그마(sigma)와 고주파성분 강도(scale)을 통제할 수 있다. 만약 sigma=0 혹은 scale=0이면 UM 처리를 하지 않는다.

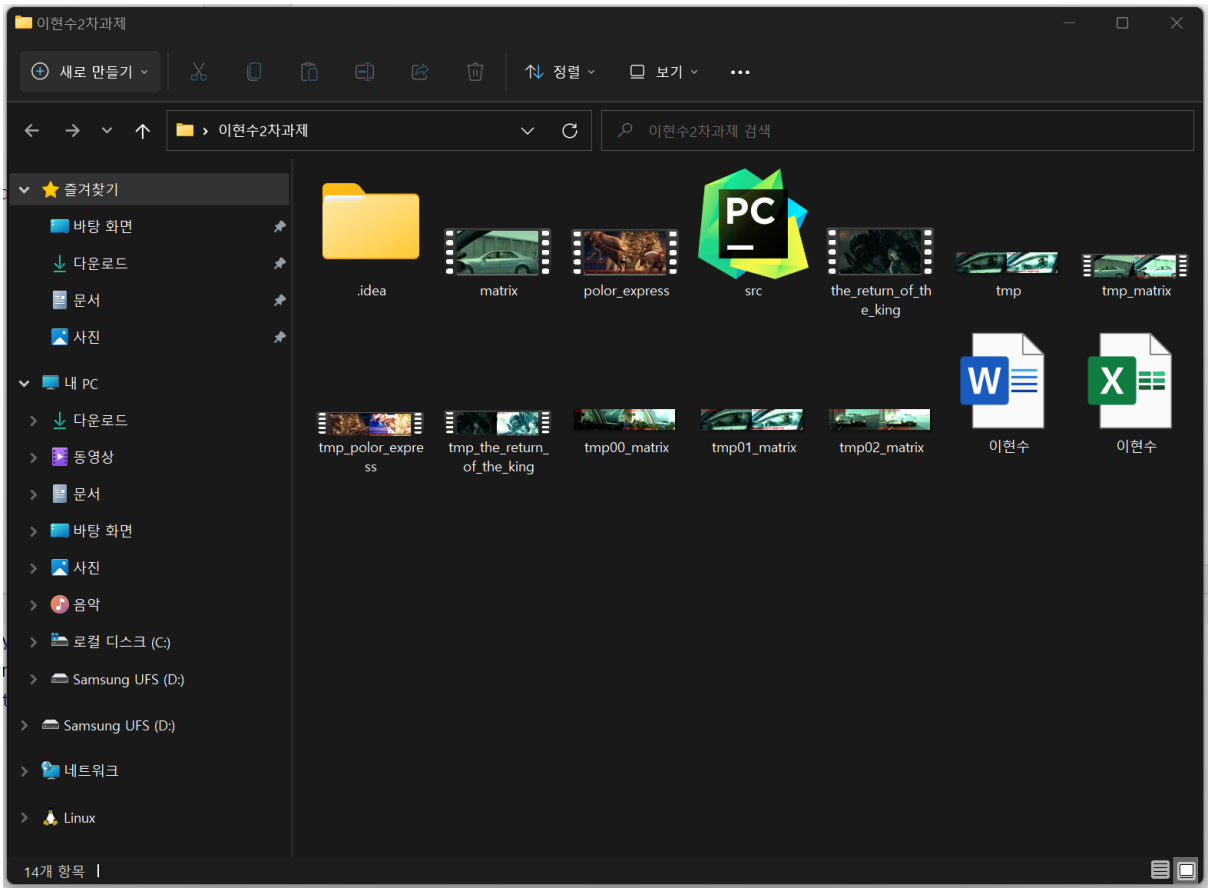
- 좌측에는 원본영상과 org_index 값을 텍스트로 출력하고, 우측에는 this_index 값과 sigma, scale, weight 값을 텍스트로 출력한다.

- 트랙바의 position 값은 500으로 원본 영상의 인덱스와는 관련이 없다. Position의 값과 원본영상의 총 프레임 값의 비율로 영상이 보이게 된다.

- 트랙바의 position 값을 임의로 조정하면 해당 프레임으로 이동한다.

- 프로그램 실행 중 space 키를 누르면 영상이 멈추고, 그 때 아무키나 누르면 영상은 다시 재생된다.

- Key 'esc'값을 누르면 영상 재생 도중에 혹은 영상이 멈춰있는 경우 모두 프로그램이 종료된다.



- 만약 key 's'를 누르면 원본영상, 개선영상 두개가 텍스트가 표시된 채로 'tmpXX_파일이름.png'이름으로 저장된다.

- 프로그램 실행 중 space 키를 눌러 영상이 멈춤을 제외하고 영상이 재생되는 동안에 프레임 값들을 저장해 avi 동영상 파일로 만든다. 프로그램이 종료되면 폴더에서 저장된 영상 확인이 가능하다.

2. 문제풀이 절차

1) 동영상 파일 읽기 및 여러 정보 알아내기

cv.VideoCapture(filename)을 통해 비디오 객체를 얻었다. 그 후 영상의 사이즈, fps를 알아냈다.

2) 영상 저장에 필요한 설정

동영상을 저장하기 위해서 cv.VideoWriter을 이용했다. 파일 이름은 tmp_파일이름으로 avi파일로 저장된다. CODEC은 cv.VideoWriter_fourcc('F','M',P','4')로 설정했다. Fps는 위에서 얻은 fps로 설정하고, 영상의 사이즈는 원본+개선 영상 두개를 합친 크기이므로 너비의 크기는 원본 영상의 너비 크기의 2배로 설정했다.

3) 트랙바 만들기

트랙바에는 총 4가지 변수를 제어할 수 있어야한다. 'HE wgt', 'sigma', 'scale', 'position' 값이다.

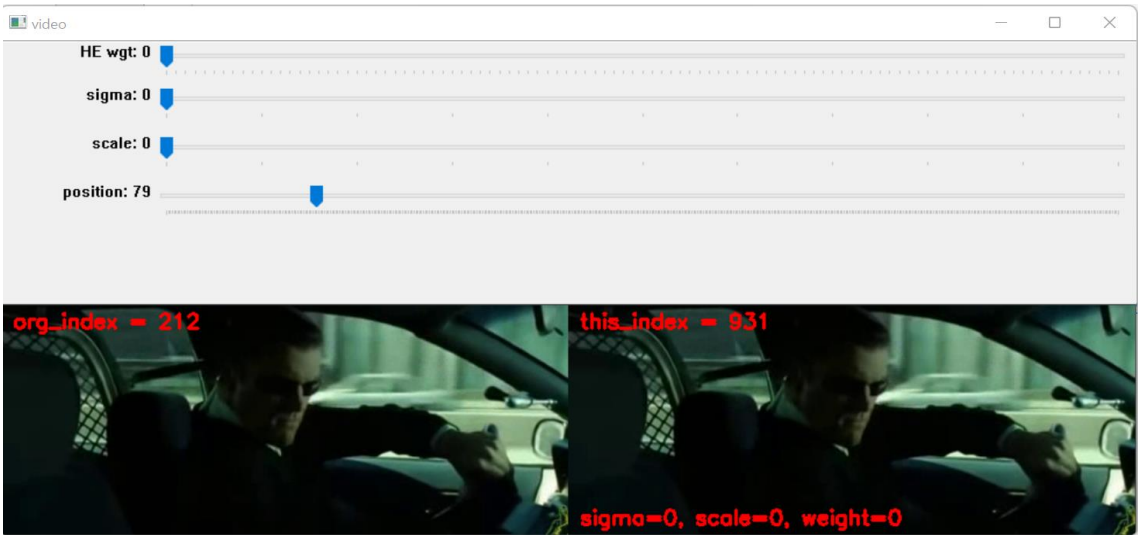
'HE wgt'는 히스토그램 평활화의 원본영상 반영비율을 조절한다. 0~100값을 가지며 초기값은 100으로 설정했다. 이벤트 함수 이름은 callback_wgt이다.

'sigma'는 언샤프마스킹 할 때 이용하는 변수이다. 0~10값을 가지며 초기값은 0이다. 이벤트 함수 이름은 callback_sigma이다.

'scale'는 언샤프마스킹 할 때 이용하는 변수이다. 0~10값을 가지며 초기값은 0이다. 이벤트 함수 이름은 callback_scale이다.

'position'은 상대적인 영상 프레임 위치를 나타내는 것으로 최댓값은 500이다. 만약 원본영상의 프레임수가 1000이면 position값 1을 증가시킬 때마다 프레임 2개씩 재생된다. 이벤트 함수 이름은 callback_position이다.

4) 트랙바 이벤트 함수 만들기



트랙바 이벤트 함수는 총 4개이다.

callback_wgt는 전역변수 wgt에 트랙바에 설정한 값을 저장했다.

callback_sigma는 전역변수 sigma에 트랙바에서 설정한 값을 저장했다.

callback_scale는 전역변수 scale에 트랙바에서 설정한 값을 저장했다.

callback_position은 전역변수 count, Touched, cnt, position을 이용했다. 우선 position값은 트랙바에서 설정한 값을 저장했다. 그 후 count는 int(total_count*(x/position_total))을 통해 position의 비율에 맞게 원본영상의 해당 프레임을 저장했다. 그 후 Touched를 True로 만들어주면 while 반복문 안에 있는 if Touched==True: 조건문이 만족된다. 이 조건문은 videoCapture.set메소드를 통해 count로 프레임을 설정하고 videoCapture.read를해 이미지 히스토그램 평활화, 언샤프 마스킹 후 이미지를 합쳐 보여주게 된다.

이 이벤트 함수를 통해 영상이 멈춰있을 때 혹은 영상이 재생되고 있을 때 position 값을 사용자가 조정해 원하는 구간으로 이동이 가능하다.

5) 히스토그램 평활화 구현(함수)

```python
# 히스토그램 평활화
def HistogramEqualization(img, wgt):
    img = (np.clip(255 * img, 0, 255)).astype('uint8')
    imgC = img.copy()  # img의 복사본을 하나 저장해 둔다.
    imgG = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
    hist, bins = np.histogram(imgG, 256, [0, 255])

    cdf = hist.cumsum()  # 누적분포함수, Cumulative Distribution Function
    cdf_normalized = cdf / cdf.max()  # 0~1 범위의  정규화된 누적분포함수, NCDF
    mapping = 255*cdf_normalized

    LUT = mapping.astype('uint8')  # Look Up Table - 256바이트 크기. 화소 값 변환 테이블.
    imgCeq = LUT[imgC]  # 3채널 칼라 영상에 대한 LUT 기반 히스토그램 평활화

    imgCeq = (np.clip(imgC + (wgt*0.01*imgCeq), 0, 255)).astype('uint8')  # 원본영상 반영비율 조절
    return imgCeq
```

히스토그램 평활화를 위한 함수를 만들었다. 매개변수로는 img, wgt가 있다. Img는 평활화 대상 프레임 이미지, wgt는 원본영상의 평활화 반영 비율이다. 우선 매개변수로 받아온 img는 0~1로 정규화된 프레임이다. 그래서 이것을 0~255 uint8 이미지로 변환한다. 그리고 cvtColor을 통해 3채널 이미지를 1채널 이미지로 만든다. 그 후 numpy모듈의 histogram 함수를 통해 hist, bin을 얻는다. 그 후 hist.cumsum함수를 통해 누적분포함수를 구한다. 그 후 0~1 범위의 정규화된 누적 분포 함수를 만들고 255를 곱해 mapping을 만든 후 이것을 uint8로 만들어 LUT를 만든다. 그 후 컬러 이미지를 LUT에 넣으면 평활화된 이미지를 얻을 수 있다. 그 후 원본영상 반영비율을 조절하기 위해 (원본3채널영상 + (wgt*0.01*평활화영상)을 한다. Wgt가 0이면 원본영상이되고, 100으로 가까울수록 히스토그램 평활화가 강하게 된다.

6) unsharp masking 구현(함수)

```python
# 언샤프마스킹
def UnsharpMasking(img, sigma, scale):
    um = img + scale * (img - cv.GaussianBlur(src=img, ksize=(6*sigma+1, 6*sigma+1), sigmaX=sigma))
    return np.clip(um, 0, 1)
```

히스토그램 평활화된 이미지를 언샤프마스킹 하기위해서 함수를 만들었다. 매개변수로는 img, sigma, scale이 있다. 우선 가우시안 블러링 영상을 만든다. 이때 커널 사이즈는 6*sigma+1로 했다. 여기서 sigma가 사용된다. 그리고 원본영상에서 가우시안 블러링한 영상을 빼서 고주파영상을 얻는다. 그 후 scale값을 곱해 고주파 성분을 강조 후 원본 이미지를 더해 만들었다.

7) Key 입력값 ‘s’가 들어올 때 구현

Key ‘s’를 누르면 이미지가 저장된다. 이때 이미지 저장은 원본과 개선 프레임을 같이 저장하고, ori_index, this_index, sigma, scale, weight 텍스트를 표시한 이미지 프레임을 저장한다.

Key ‘s’를 누르는 경우는 두가지이다. 영상이 재생되고 있을때와 영상이 정지되었을 때이다.

두 경우 모두 while 반복문안에 key값을 처리하는 조건문을 넣어주면 구현이 된다. 그래서 키값 s 즉 아스키코드 115가 들어오면 원본, 개선 이미지를 합친 이미지 프레임을 복사해 255를 곱하고 파일이름을 정해 imwrite 함수를 통해 이미지를 저장한다.

8) 영상 멈춤, 다시 재생 구현

재생되고 있는 영상을 멈추가 위해서는 space 키를 누른다. Space 키는 아스키코드 32이다. 키값이 32라면 IsPlay 변수를 False로 만들어 줌으로써 영상을 계속해 재생시키던 while 반복문 조건이 불만족해 영상이 멈춘다.

그 후 아무키나 누르면 다시 영상이 재생된다. 이때 다른 키값처리를 조건문으로 처리하고 마지막에 키값이 -1이 아닌경우를 구현해 IsPlay를 True로 만들어 영상을 재생하던 while 반복문 조건이 만족되어 다시 영상은 재생된다.

9) Key 입력값 ‘esc’가 들어올 때 구현

Key 값 esc는 아스키코드로 27이다. 조건문으로 esc키 값이 들어오면 즉 27이면 영상이 멈춰있다면 break문을 통해 프로그램을 종료하고 영상이 재생중이라면 exit(0)을 통해 프로그램을 종료한다.

3. 소스코드

```python
# 아래 Path와 Name은 그대로 현재 설정을 사용해 주세요....
# 현재 d:/dip/the_return_of_the_king.avi 파일을 입력 동영상으로 사용하고 있습니다.
Path = ''#'d:/dip/'
Name = 'matrix.avi'

import cv2 as cv
import time
import numpy as np

FullName = Path + Name

videoCapture = cv.VideoCapture(FullName)

total_count = int(videoCapture.get(cv.CAP_PROP_FRAME_COUNT))
fps = videoCapture.get(cv.CAP_PROP_FPS)
dly_ms = 1000/(fps)       # dly_ms: ms로 표시한 프레임간의 간격[ms]
size = (int(videoCapture.get(cv.CAP_PROP_FRAME_WIDTH)),
    int(videoCapture.get(cv.CAP_PROP_FRAME_HEIGHT)))

CODEC = cv.VideoWriter_fourcc('F', 'M', 'P', '4')          # 파일 용량 적음
fourcc = cv.VideoWriter_fourcc(*'DIVX')
# 출력 저장용 동영상 -----------------------------
CODEC = cv.VideoWriter_fourcc('F', 'M', 'P', '4')          # 파일 용량 적음
fourcc = cv.VideoWriter_fourcc(*'DIVX')
fname = Name[0:-4]
SaveFileName = f'tmp_{fname}.avi'

videoWriter = cv.VideoWriter(SaveFileName, CODEC, fps, (size[0]*2, size[1]))

position_total = 500 # 트랙바 'position' 최대값
position = 0 # position 값
count = 0 # 프레임 카운트 값
margin=1
IsPlay = True # 동영상 일시정지 여부
Touched = False # 트랙바 'position' 값을 제어했는지 여부
wgt = 100 # 트랙바 'HE wgt' 값
sigma = 0 # 트랙바 'sigma' 값
scale=0 # 트랙바 'scale' 값
cnt = 0 # this_index 변수

# 트랙바 'HE wgt' 이벤트 함수
def callback_wgt(x):
    global wgt
    wgt = x

# 트랙바 'sigma' 이벤트 함수
def callback_sigma(x):
    global sigma
    sigma = x

# 트랙바 'scale' 이벤트 함수
def callback_scale(x):
    global scale
    scale = x

```

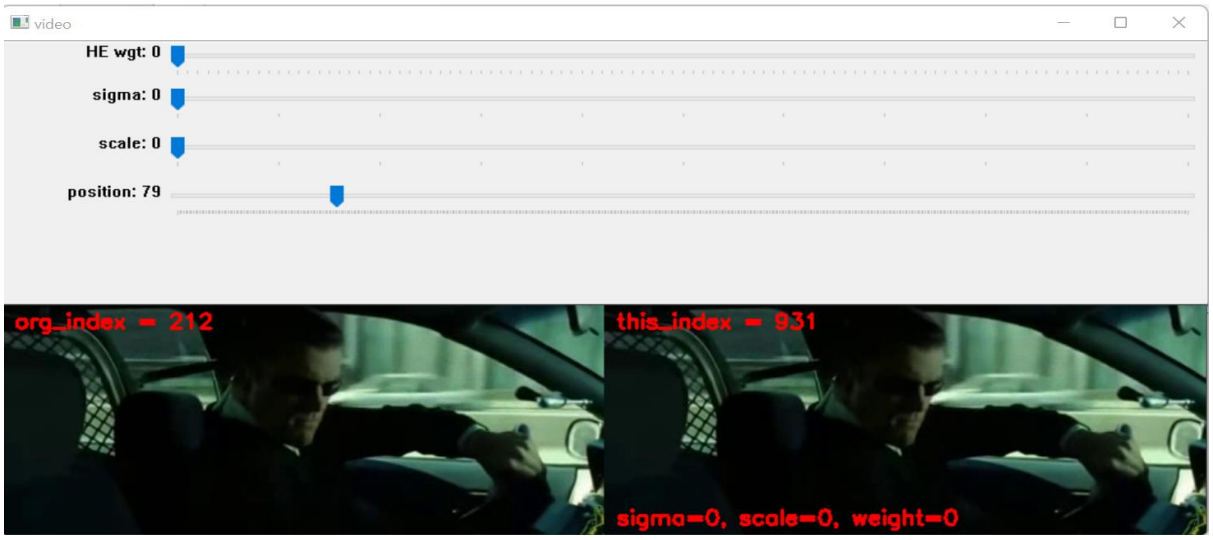
```python
# 트랙바 'position' 이벤트 함수
def callback_position(x):
    global count, Touched, cnt, position
    position = x
    precount = count
    count=int(total_count*(x/position_total))
    cnt += (abs(count-precount) + 1)
    Touched=True

# 트랙바 설정 및 만들기
cv.namedWindow('video')
cv.createTrackbar('HE wgt', 'video', 100, 100,  callback_wgt)
cv.createTrackbar('sigma', 'video', 0, 10,  callback_sigma)
cv.createTrackbar('scale', 'video', 0, 10,  callback_scale)
cv.createTrackbar('position', 'video',  0, position_total,  callback_position)

# 히스토그램 평활화
def HistogramEqualization(img, wgt):
    img = (np.clip(255 * img, 0, 255)).astype('uint8')
    imgC = img.copy()  # img의 복사본을 하나 저장해 둔다.
    imgG = cv.cvtColor(img, cv.COLOR_RGB2GRAY)
    hist, bins = np.histogram(imgG, 256, [0, 255])

    cdf = hist.cumsum()  # 누적분포함수, Cumulative Distribution Function
    cdf_normalized = cdf / cdf.max()  # 0~1 범위의  정규화된 누적분포함수, NCDF
    mapping = 255*cdf_normalized

    LUT = mapping.astype('uint8')  # Look Up Table - 256바이트 크기. 화소 값 변환 테이블.
    imgCeq = LUT[imgC]  # 3채널 칼라 영상에 대한 LUT 기반 히스토그램 평활화

    imgCeq = (np.clip(imgC + (wgt*0.01*imgCeq), 0, 255)).astype('uint8')  # 원본영상 반영비율 조절
    return imgCeq

# 언샤프마스킹
def UnsharpMasking(img, sigma, scale):
    um = img + scale * (img - cv.GaussianBlur(src=img, ksize=(6*sigma+1, 6*sigma+1), sigmaX=sigma))
    return np.clip(um, 0, 1)

success, frame = videoCapture.read()
frame_dip = frame.copy()
frame_h = cv.hconcat([frame, frame_dip])
file_save_cnt = 0 # 이미지 파일 저장 시 파일 이름에 붙는 숫자.
```

```python
while(1):
    k = cv.waitKey(1)
    if k==27: # esc
        break # 프로그램 종료
    elif k == 115: # s
        frame_h_save = frame_h.copy()
        frame_h_save = frame_h_save * 255 # 0~255 정규화
        filename = f'tmp{file_save_cnt:0>2}_{Name[:Name.rfind(".")]}.png' # 파일 이름
        cv.imwrite(filename, frame_h_save) # 파일 저장
        file_save_cnt+=1 # 파일 저장 횟수 1 증가
    elif k!=-1: # 그 외 키를 입력하면 다시 동영상 재생
        IsPlay=True

    if Touched==True:
        videoCapture.set(cv.CAP_PROP_POS_FRAMES, count) # 프레임 설정
        success, frame = videoCapture.read()  # 프레임을 읽어온다.
        if success!=True:
            break
        frame = frame / 255 # 0~1 정규화
        frame_copy = frame.copy()
        frame_hist = HistogramEqualization(frame_copy, wgt) # 히스토그램 평활화
        frame_hist_um = UnsharpMasking(frame_hist / 255, sigma, scale) # 언샤프 마스킹

        # 원본, 개선 이미지 프레임에 텍스트 출력
        frame_str = f'org_index = {count}'
        frame_hist_str = f'this_index = {cnt}'
        frame_hist_str2 = f'sigma={sigma}, scale={scale}, weight={wgt}'
        cv.putText(frame, frame_str, (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
        cv.putText(frame_hist_um, frame_hist_str, (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
        cv.putText(frame_hist_um, frame_hist_str2, (10, size[1]-10), cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        frame_h = cv.hconcat([frame, frame_hist_um]) # 원본영상과 개선영상 합치기
        cv.imshow('video', frame_h) # 보여주기
        Touched=False

    while success and IsPlay:  # Loop until there are no more frames.
        k = cv.waitKey(1)
        if k == 32:  # space
            IsPlay=False # 영상 멈추기
        elif k==27: # esc
            exit(0) # 프로그램 종료
        elif k==115: # s
            frame_h_save = frame_h.copy()
            frame_h_save = frame_h_save * 255
            filename = f'tmp{file_save_cnt:0>2}_{Name[:Name.rfind(".")]}.png'
            cv.imwrite(filename, frame_h_save)
            file_save_cnt += 1

        s = time.time()  # start. time in sec.
        cv.imshow('video', frame_h) # 프레임 보이기
        position+=1 # position 1 증가
        cv.setTrackbarPos("position", "video", position) # 트랙바 'position' 1증가된 값으로 설정
        videoCapture.set(cv.CAP_PROP_POS_FRAMES, count) # 프레임 설정
        success, frame = videoCapture.read()  # 프레임을 읽어온다.
        if success!=True:
            break
        frame = frame/255
        frame_copy = frame.copy()
        frame_hist = HistogramEqualization(frame_copy, wgt) # 히스토그램 평활화
        frame_hist_um = UnsharpMasking(frame_hist/255, sigma, scale) # 언샤프 마스킹

```

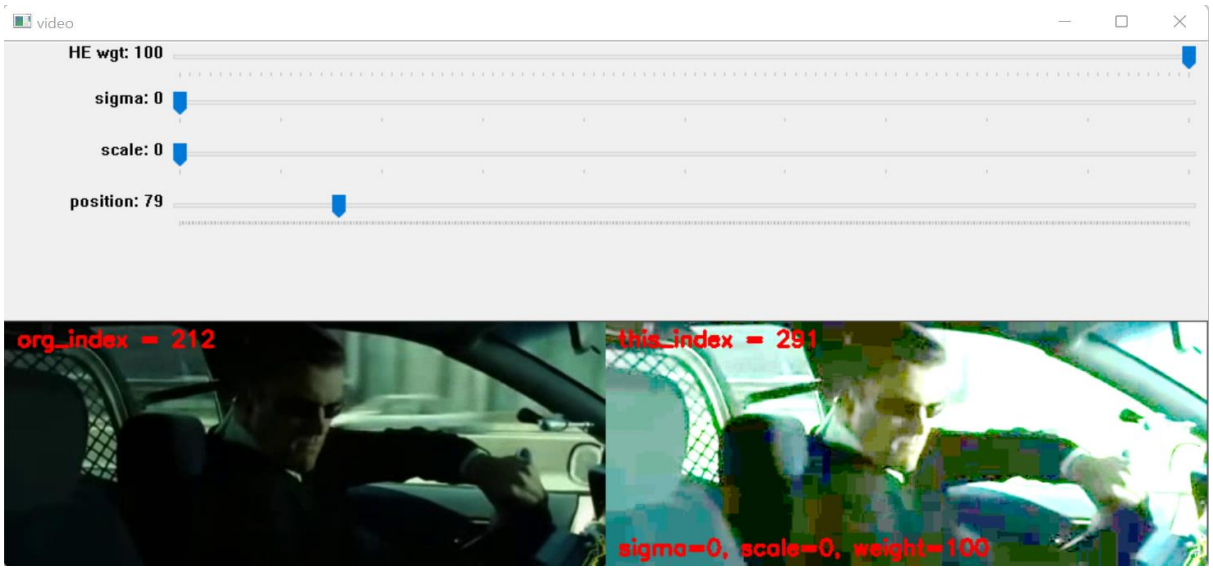
```
        # 원본, 개선 이미지 프레임에 텍스트 출력
        frame_str = f'org_index = {count}'
        frame_hist_str=f'this_index = {cnt}'
        frame_hist_str2 = f'sigma={sigma}, scale={scale}, weight={wgt}'
        cv.putText(frame, frame_str, (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
        cv.putText(frame_hist_um, frame_hist_str, (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.68, (0, 0, 255), 2)
        cv.putText(frame_hist_um, frame_hist_str2, (10, size[1] - 10), cv.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        frame_h = cv.hconcat([frame, frame_hist_um]) # 이미지 합치기
        frame_h_video = (np.clip(255 * (frame_h), 0, 255)).astype('uint8') # 비디오로 저장할 이미지 가공
        videoWriter.write(frame_h_video)  # 비디어로 저장
        print("\rCurrent frame number = ", count, end=' ')
        while ((time.time() - s) * 1000) < (dly_ms - margin):  # dly_ms: ms로 표시한 프레임간의 간격[ms]
            pass

videoCapture.release()
cv.destroyAllWindows()
```

4. 실행결과

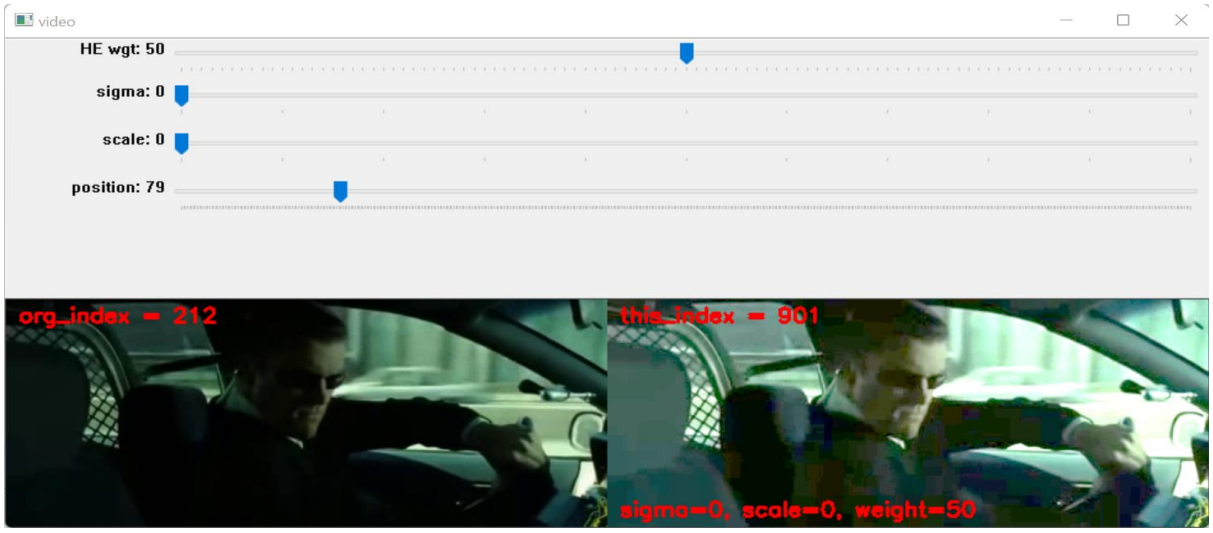
1) 테스트1



▲원본 영상 (HE wgt = 0, sigma = 0, scale = 0)

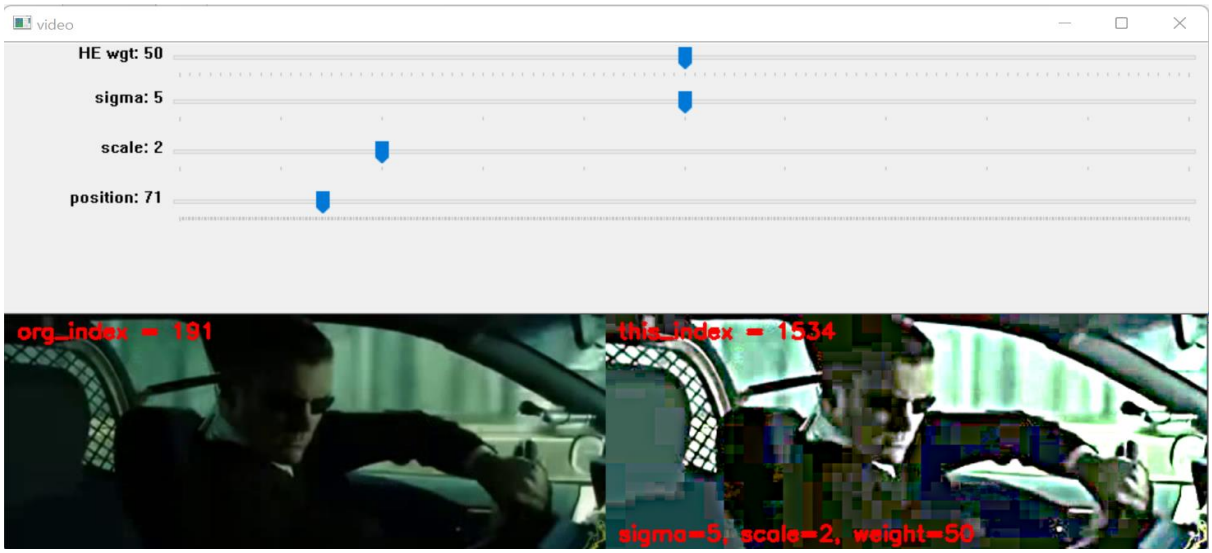


▲히스토그램 평활화 100 (HE wgt = 100, sigma = 0, scale = 0)

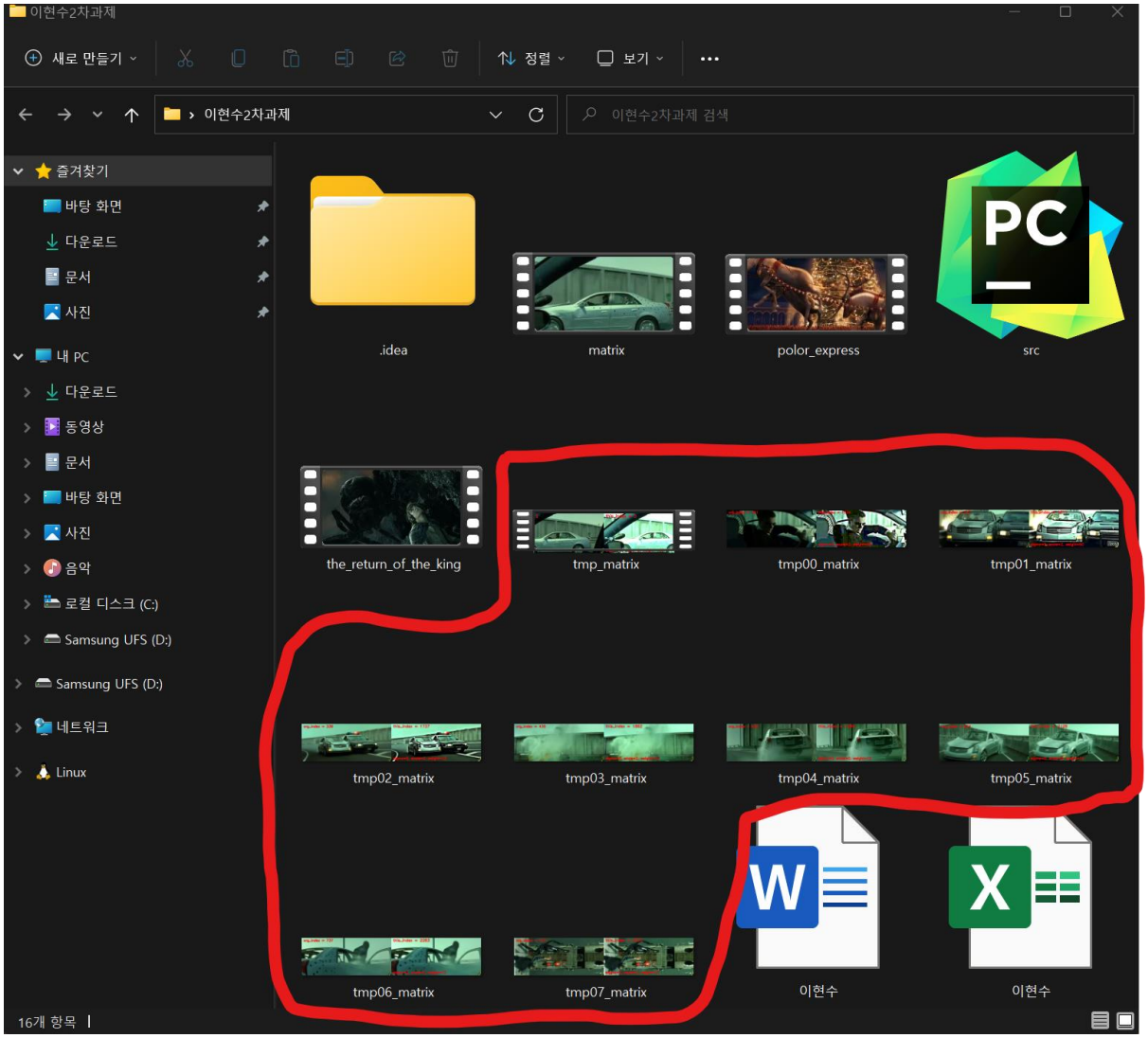


▲히스토그램 평활화 50 (HE wgt = 50, sigma = 0, scale = 0)

▲unsharp masking (HE wgt = 0, sigma = 5, scale = 2)

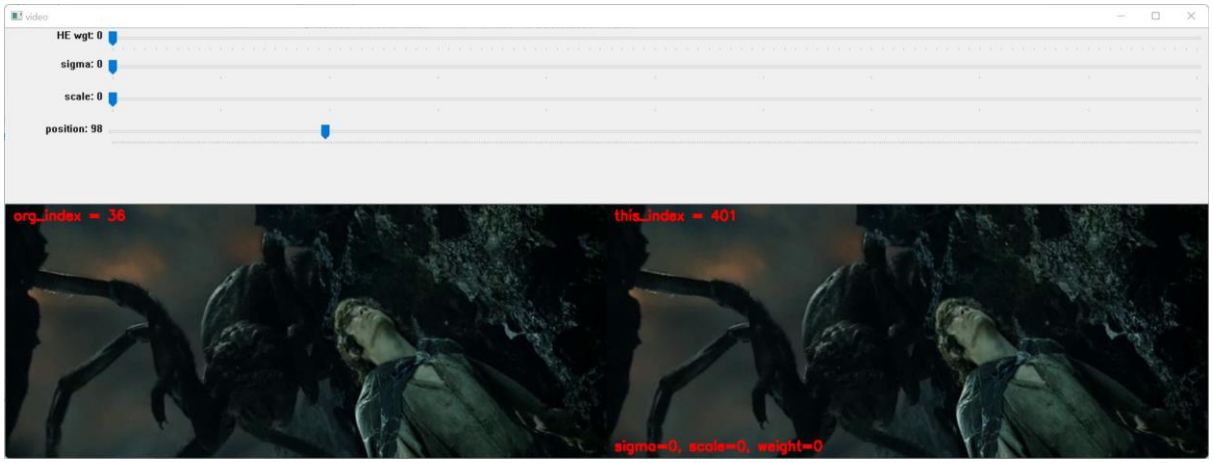


▲히스토그램 평활화 + unsharp masking (HE wgt = 50, sigma = 5, scale = 2)

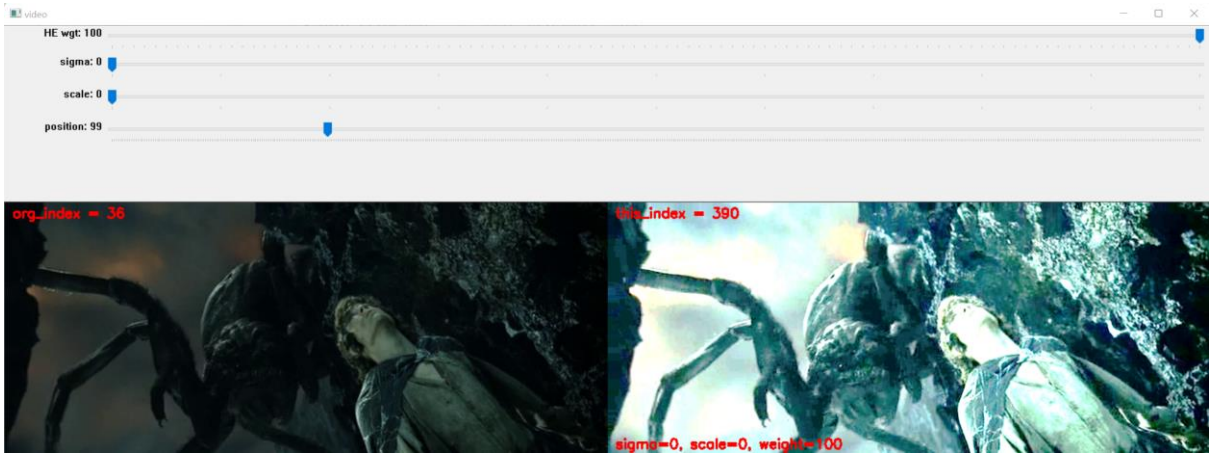


▲tmp.png 파일저장, 영상 저장

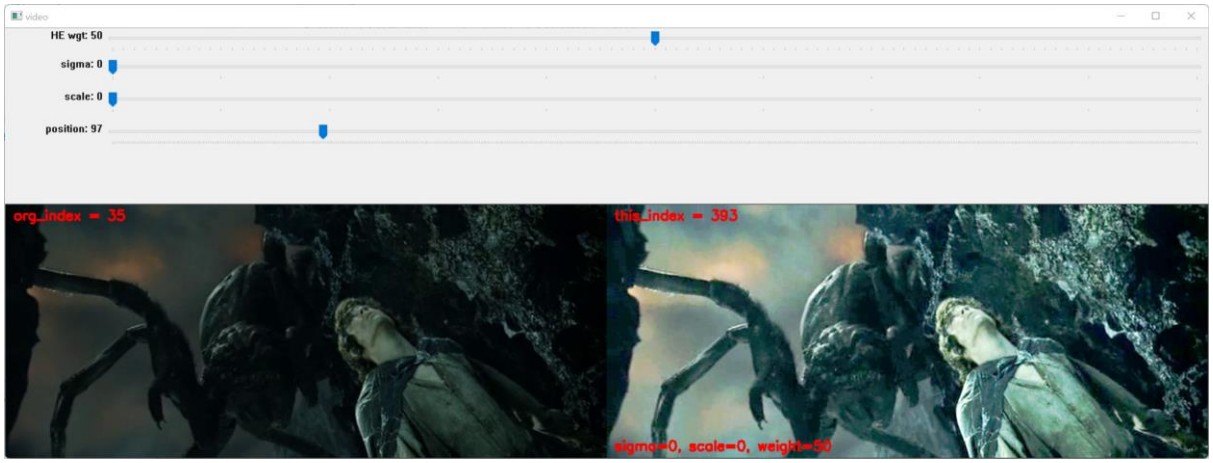
2) 테스트2



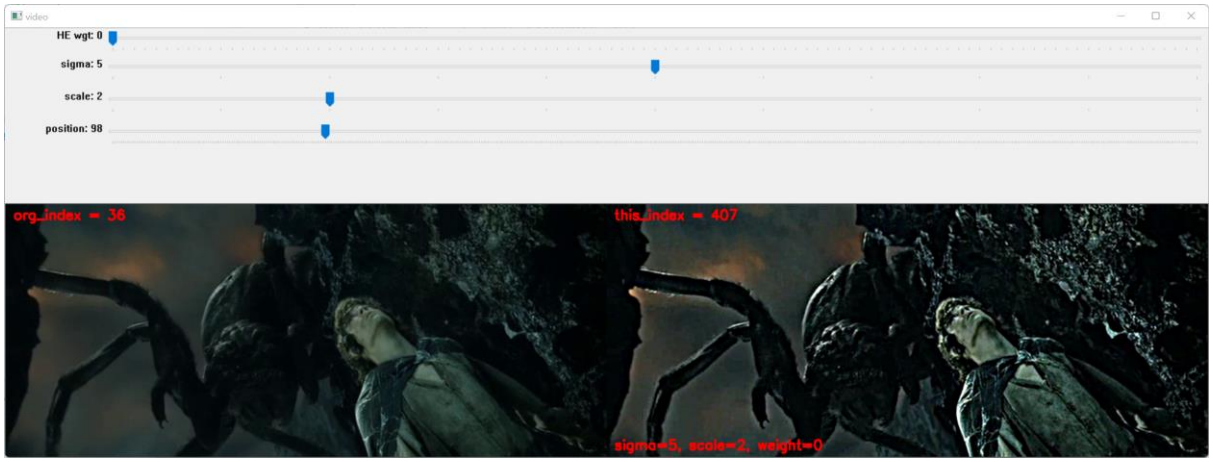
▲원본 영상 (HE wgt = 0, sigma = 0, scale = 0)



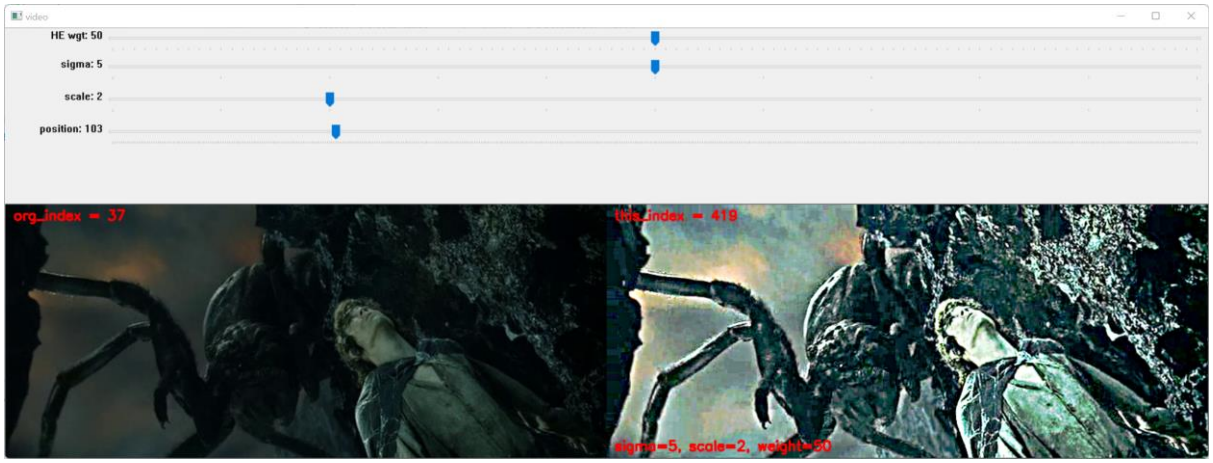
▲히스토그램 평활화 100 (HE wgt = 100, sigma = 0, scale = 0)



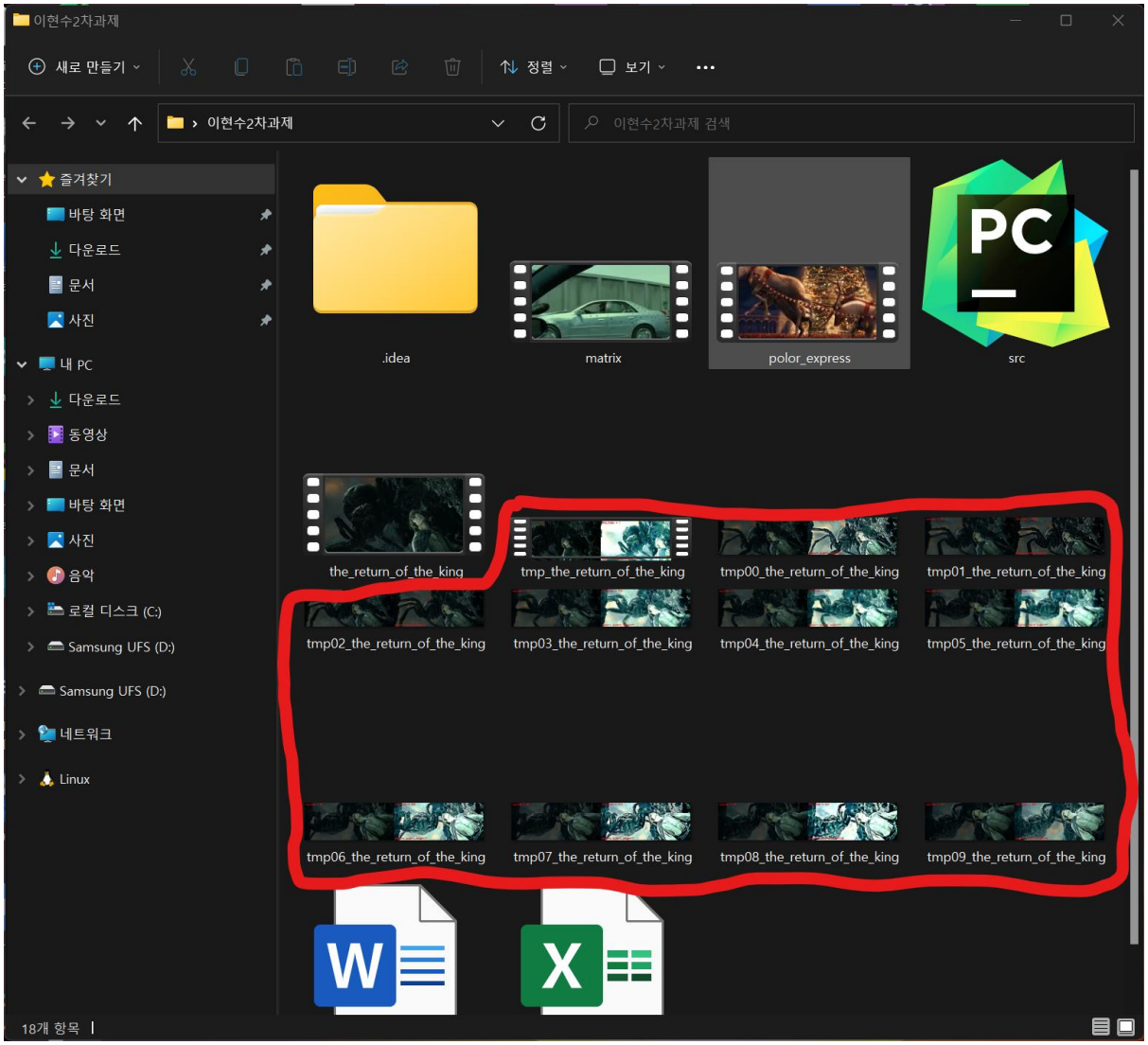
▲히스토그램 평활화 50 (HE wgt = 50, sigma = 0, scale = 0)



▲unsharp masking (HE wgt = 0, sigma = 5, scale = 2)

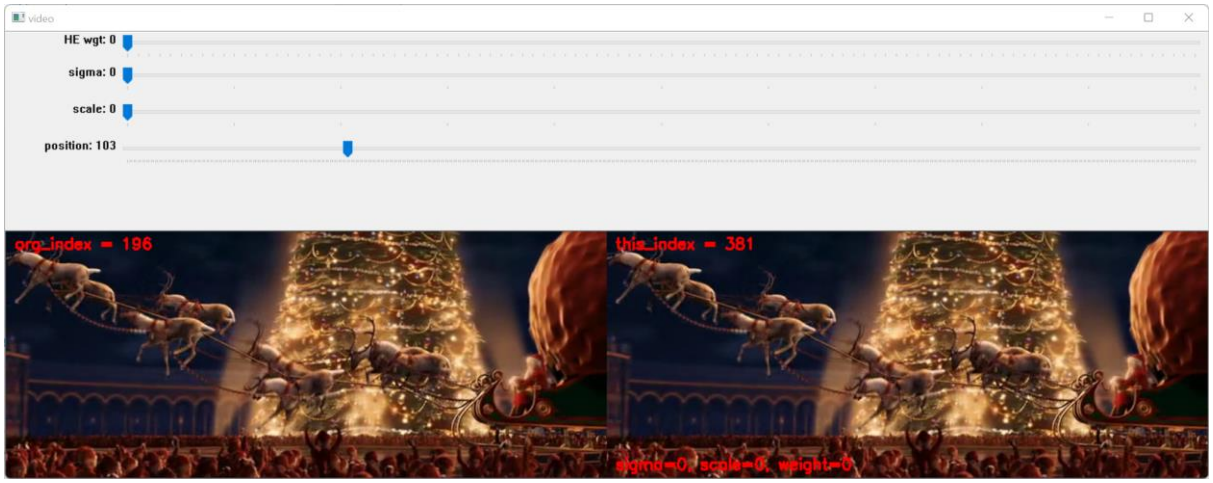


▲히스토그램 평활화 + unsharp masking (HE wgt = 50, sigma = 5, scale = 2)

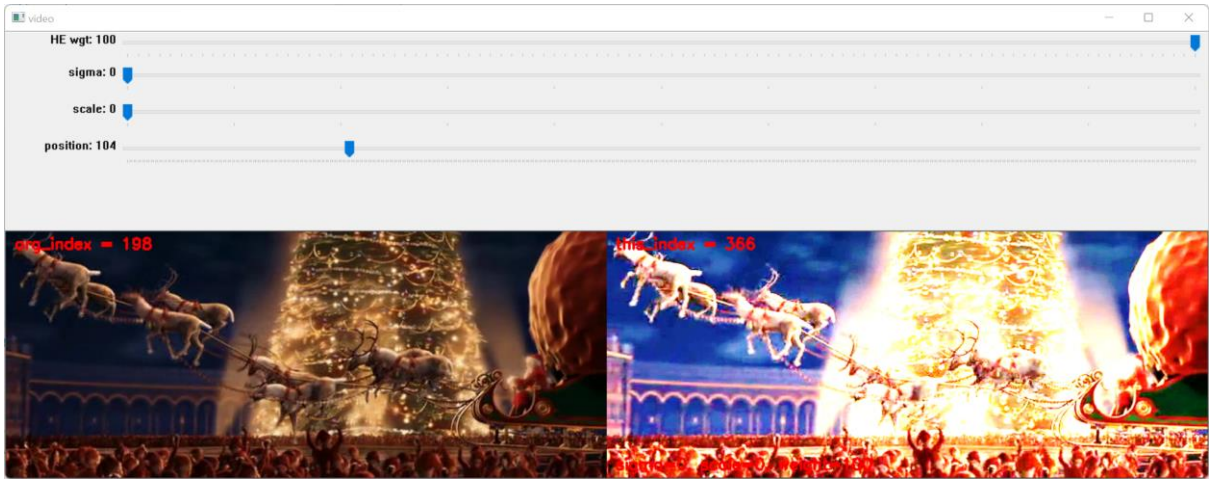


▲tmp.png 파일저장, 영상 저장

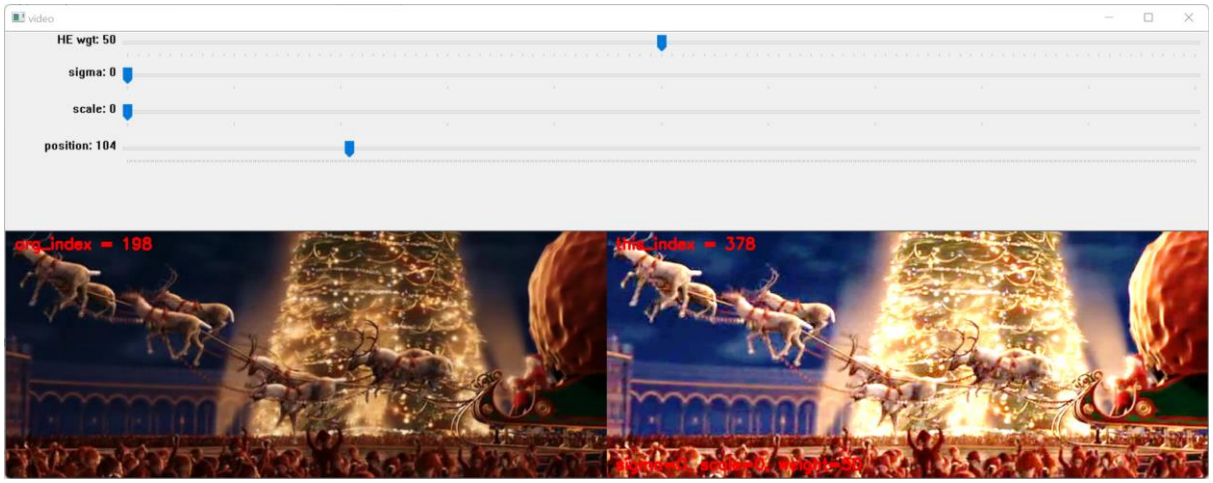
3) 테스트3



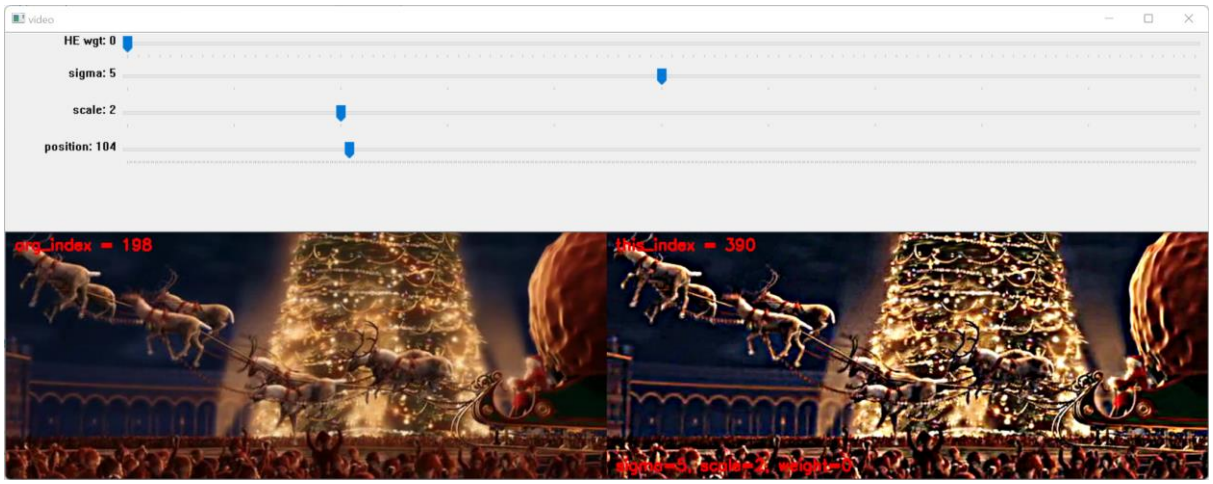
▲원본 영상 (HE wgt = 0, sigma = 0, scale = 0)



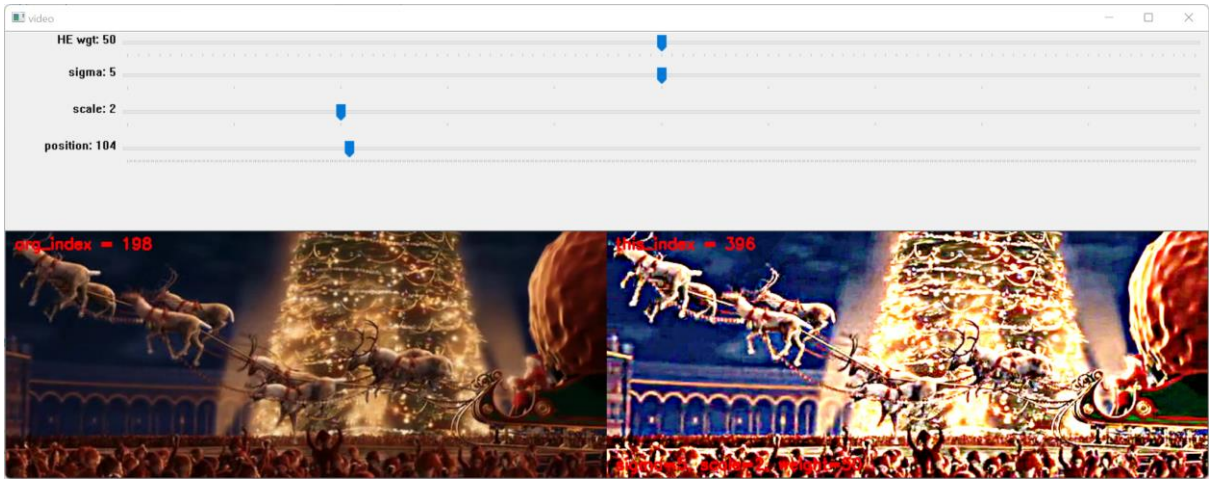
▲히스토그램 평활화 100 (HE wgt = 100, sigma = 0, scale = 0)



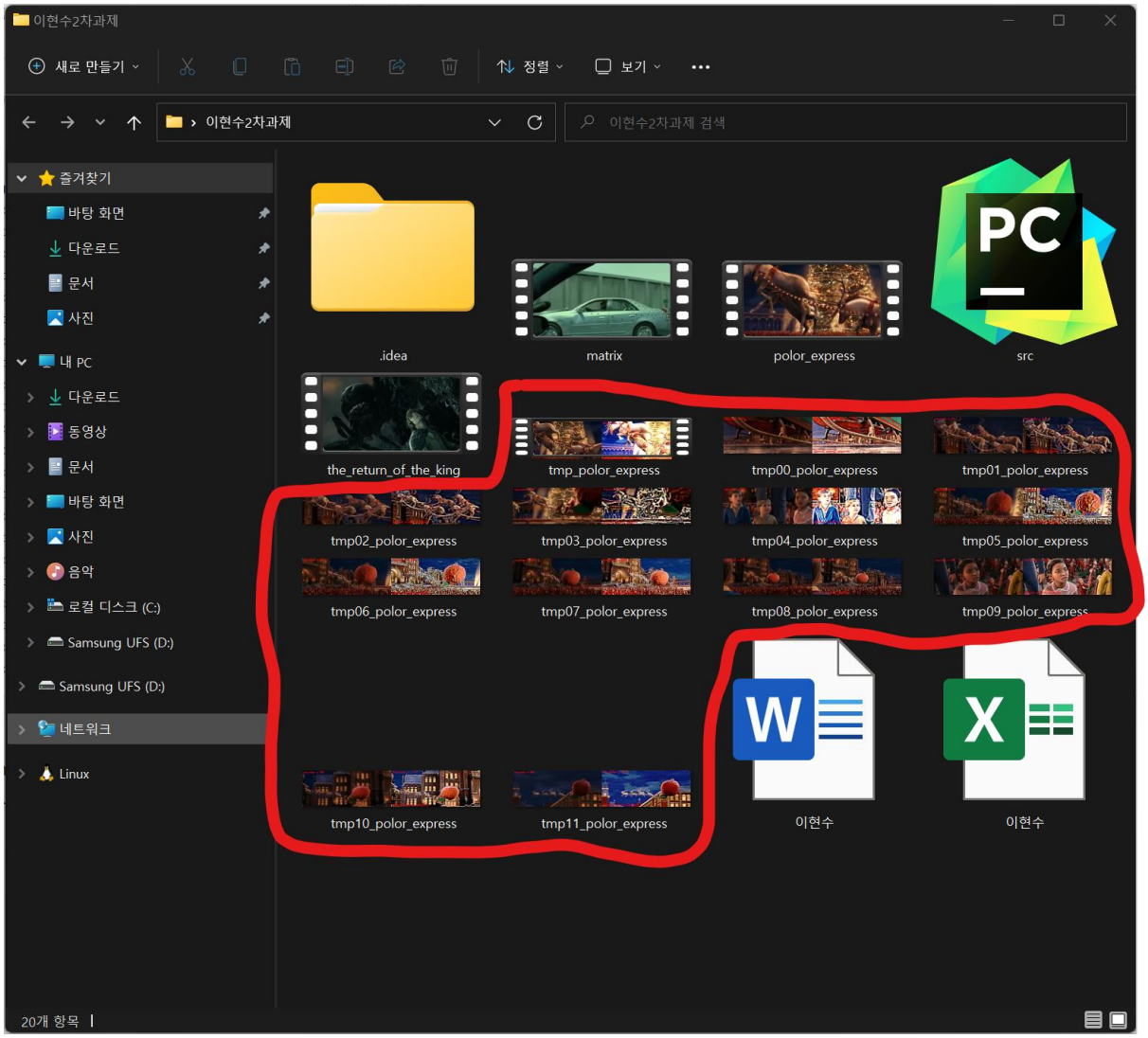
▲히스토그램 평활화 50 (HE wgt = 50, sigma = 0, scale = 0)



▲unsharp masking (HE wgt = 0, sigma = 5, scale = 2)



▲히스토그램 평활화 + unsharp masking (HE wgt = 50, sigma = 5, scale = 2)



▲tmp.png 파일저장, 영상 저장

5. 시행착오, 배운점, 기술적의의

1) 시행착오1 – 부동소수 영상은 보이는건 0~1, 저장하는건 0~255

부동소수 영상을 imshow를 이용해 보이는건 0~1영역에 대해서, 저장할 때는 0~255 범위의 이미지가 정상적으로 보이고 저장되었다.

2) 배운점 및 느낀점

다양한 기능이 포함된 동영상 화질 개선 프로그램을 만들면서 수업시간에 간단히 실습해본 히스토그램 평활화, 언샤프 마스킹, 동영상 저장하기 등의 기능을 실제 상황에서 구현해봄으로 동영상과 동영상 프레임에 대한 이해가 올라갔다.

처음에 과제가 어려워보였지만 수업시간에 배운 내용과 코드를 기반으로 문제를 풀고 여러 시행착오를 겪으면서 완성할 수 있었다.

3) 기술적 의의

3채널 영상 이미지의 히스토그램 평활화와 언샤프 마스킹을 동시에 할 수 있다. 또 트랙바를 통해 원하는 값으로 조절해 보다 정교하게 화질 개선이 가능하다.