REPORT

“ 디지털영상처리 2차 과제 ”

과 목 명	디지털영상처리
담당교수	김진헌 교수님
학 과	컴퓨터 공학과
학 번	2016305078
이 름	최영환
제 출 일	2021.11.10

목 차

< 과제 소개 및 설명 >

※ 동영상을 열어 다음 절차에 따라 화질 개선된 동영상을 저장하는 프로그램을 설계하시오.

1. 입력 원본 프레임 영상에 대해 히스토그램 평활화를 행한다.

2. 원본 영상과 히스토그램 평활화 영상을 가중 평균하여 평활화의 반영비율을 제어한다. 이때는 가중치 트랙바가 활용된다. 평활화 영상의 반영 비율을 0 ~ 100% 까지 선택할 수 있다.

3. 이렇게 개선된 영상에 대해 언샤프 마스킹을 행한다. 언샤프 마스킹의 시그마(sigma)를 트랙바로 설정할 수 있고, 그 고주파 성분의 강도(scale)를 다른 트랙바로 통제할 수 있다.

4. 이때 시그마나 강도의 값이 0일 경우, 언샤프 마스킹 처리를 하지 않는다.

< 프로그램 요약 >

※ 본 과제를 구현한 프로그램의 흐름을 간단히 요약하면 아래와 같다.

1. 동영상을 읽은 후, 트랙바 및 출력 윈도우를 생성한다.

2. 매 프레임마다 트랙바의 값을 획득하고, 영상 재생을 위한 변수들의 값을 업데이트 한다.

3. 히스토그램 평활화를 시행한다. 이때, 트랙바에서 설정한 가중치 값(HE weight 값)을 기반으로 평활화 비율을 조절한다.

4. 평활화된 영상에 언샤프 마스킹을 수행한 다음 출력한다. 이때, 시그마(sigma)나 강도(scale) 의 값이 0일 경우, 언샤프 마스킹을 수행하지 않고 출력한다.

5. 사용자의 키 입력을 기다린다.

※ 본 페이지부터는 프로그램에 관한 설명입니다.

< 프로그램 구현 >

1. 동영상을 읽은 후, 트랙바 및 출력 윈도우 생성 (+ 라이브러리 import)

```python
# 사용된 라이브러리
import cv2 as cv
import numpy as np
import time

# 변수 선언
Path = './'
Name = 'matrix.mp4'
```

본 과제에서 사용된 라이브러리는 numpy 와 cv2, 라이브러리, time 라이브러리이다. 또한, 과제의 조건이 경로와 파일변수 명을 Path 와 Name으로 사용하는 것이므로, 위와 같이 변수를 선언하였다.

```python
FullName = Path + Name      # 재생 할 파일의 전체 경로

# 영상 객체 선언 및 로드
cap = cv.VideoCapture(FullName)

success, frame = cap.read()        # 영상 로드

fps = cap.get(cv.CAP_PROP_FPS)     # 가져온 영상의 fps
number_of_total_frames = cap.get(cv.CAP_PROP_FRAME_COUNT)  # 영상의 총 프레임 개수
dly_ms = 1000/(fps)       # dly_ms: ms로 표시한 프레임간의 간격[ms]

print('fps of input file:' + Name + '=', fps)
print('Number of total frames, CAP_PROP_FRAME_COUNT=', int(number_of_total_frames))
print(f'delay time between frames={int(dly_ms)}[ms]')

cv.namedWindow(Name)    # 출력 윈도우 생성

# 트랙바 생성
cv.createTrackbar('HE wgt', Name, 100, 100, onChange)                          # 평활화 가중치
cv.createTrackbar('sigma', Name, 0, 7, onChange)                               # 시그마 값
cv.createTrackbar('scale', Name, 0, 5, onChange)                               # 강도 값
cv.createTrackbar('position', Name, 0, int(number_of_total_frames), onChange)  # 현재 프레임의 위치 값
```

원본 파일의 전체경로표현을 위한 변수로 FullName을 사용하였다. 또한, 동영상을 사용하므로, cv 모듈의 VideoCapture() 객체 cap을 생성하고, 재생 할 동영상을 로드한다. 이때, 영상의 FPS와 총 프레임 개수를 각각의 변수에 할당한다. dly_ms 변수는 예제 프로그램의 소스를 참고하여 추가한 변수로, 영상의 재생속도 조절에 사용되었다.

영상의 자료형에 대한 조건이 없으므로, 고려하지 않았으며, 출력 윈도우를 먼저 생성하고, 과제의 조건으로 주어진 네 개의 트랙바를 생성한다.

각각 히스토그램 평활화 비율 조절, 언샤프 마스킹의 시그마 및 강도 조절, 현재 프레임의 위치 조절에 사용되는 값들이다.

< 프로그램 구현 >

2. 매 프레임마다 트랙바의 값을 획득하고, 영상 재생을 위한 변수들의 값을 업데이트 한다.

※ 영상 재생부는 예제 프로그램을 참조하였습니다.

```
74  # 영상 재생부
75  margin = 1                  # 순수한 영상출력(재생) 외의 다른 작업에 소비되는 예상 추정시간[ms]. 경험치
76  count = 0                   # 재생된 총 프레임 개수
77  s_time = time.time()        # ms 단위의 현재 tick count을 반환
78
79  while success:          # 영상의 끝까지 재생
88      s = time.time()     # 시작 시간
81      count += 1          # 재생된 총 프레임 개수
82
83      # 트랙바 값 획득
84      sigma = cv.getTrackbarPos('sigma', Name)
85      scale = cv.getTrackbarPos('scale', Name)
86      pos = cv.getTrackbarPos('position', Name)
87      weight = cv.getTrackbarPos('HE wgt', Name)
88
89      pos += 1                                # 현재 트랙바의 위치 값 증가
90      cv.setTrackbarPos('position', Name, pos)    # 현재 트랙바의 위치 설정
91      cap.set(cv.CAP_PROP_POS_FRAMES, pos)        # 현재 프레임 = 현재 트랙바의 위치
92      success, frame = cap.read()                 # 다음 프레임을 읽어온다
```

영상을 재생하기 위한 준비를 하는 부분으로, margin은 영상의 재생 속도 조절을 위해 사용된 변수이며, 역시 예제 프로그램을 참조하였다. count 변수는 프로그램 실행 중 총 재생된 프레임 개수를 저장하며, s_time은 재생시간 측정을 위해 사용되었다.
동영상을 재생해야하므로 while 루프 내에서 모든 동작이 처리가 되도록 하였으며, 위에서 생성한 트랙바의 값을 얻어온다.
현재 트랙바의 위치 값을 얻어서 이를 다음 프레임을 읽을 때 사용한다.

3. 히스토그램 평활화를 시행한다. 이때, 트랙바에서 설정한 가중치 값(HE weight 값)을 기반으로 평활화 비율을 조절한다.

```
94      # 히스토그램 평활화(HE) 시행
95      if frame is not None:
96          frameC = frame.copy()                                   # 원본 프레임 복사
97          frameCeq = Historgram_Equalization(frameC, weight, frameC.dtype)  # HE 프레임
98
99          # 원본 영상에 텍스트 추가
100         org = (0, 30);  org2 = (0, frameC.shape[0]-10)
101         cv.putText(frameC, f"org_index = {pos}", org, 0, 0.8, (0, 0, 255), 2)
```

읽어온 원본 프레임을 원본 손상 방지를 위해 copy 함수를 통해 복사하고, 히스토그램 평활화(이하 HE) 함수를 호출하여 HE를 시행한다. 복사본과 가중치(weight), 자료형을 매개변수로 전달한다. 이후, 원본 영상에 텍스트를 미리 추가해둔다.

< 프로그램 구현 >

< 히스토그램 평활화(HE) 함수 >

```python
# 히스토그램 평활화 함수
def Historgram_Equalization(img, weight, dtype):
    imgG = cv.cvtColor(img, cv.COLOR_BGR2GRAY)      # 그레이 스케일로 변환

    # 히스토그램(분산함수, DF) 를 구함
    hist, bins = np.histogram(a=imgG, bins=256, range=[0, 255])
    cdf = hist.cumsum()  # 누적분포함수

    # 매핑 후 LUT(Look Up Table) 생성
    mapping = cdf * 255 / cdf[255]
    LUT = mapping.astype(dtype)

    # LUT 기반 HE 시행, 컬러
    imgCeq = LUT[img]

    # 평활화 비율 조절
    w = weight * 0.01
    imgCeq = np.clip((255 * (w * imgCeq / 255)) + (255 * ((1 - w) * img / 255)), 0, 255).astype(dtype)

    return imgCeq
```

HE는 영상 내 모든 화소의 분포를 균등하게 만듦으로써 영상의 가시성을 높이는 작업이다.
히스토그램 평활화 함수에서의 영상 처리 과정은 교재와 예제 프로그램을 참조하였으며, 과정은 다음과 같다. 매개변수로 전달받은 영상을 그레이 스케일의 영상으로 변환하고, numpy 모듈의 histogram() 함수를 사용하여, 분산함수(DF)를 구한 뒤, cumsum() 함수를 사용하여 누적 분포함수(CDF)를 구한다.(cumsum 함수는 배열에서 주어진 축에 따라 누적되는 원소들의 누적 합을 계산하는 함수이다.) 각각의 함수들은 hist 변수와 cdf 변수로 표현되었다.
이렇게 구해진 누적 분포 함수를 최댓값인 cdf[255]로 나누어 정규화 누적 분포 함수를 구하고, 이 값에 255를 곱하여 매핑함수를 구한다. 얻어진 매핑 함수를 이용해 LUT를 생성하고, LUT를 기반으로 HE를 시행한다.

다음으로는, 가중치 값을 사용하여 평활화의 비율을 조절하는 부분이다.
과제의 공고와 평가표에서 가중 평균하여 처리하여야 한다는 조건이 있어서 가중 평균 값을 구한 뒤, 가중 평균을 하려 하였으나, 여러 문제점들에 부딪히던 중, 가중치 값을 사용하여, 원본과 평활화 영상의 비율만을 조정하는 시도를 하였고, 그 결과 과제의 시연영상과 매우 흡사한 결과를 얻었다. 처리 과정은 다음과 같다. HE의 적용 비율을 0~100%까지로 한다는 조건에 근거하여, 얻어온 트랙바의 값에 0.01 을 곱하여 가중치의 값을 0~1 사이의 수로 설정하고, 이 값을 평활화 된 영항에 곱한다. 동시에 원본영상에는 (1 – 가중치의 값)을 곱하고, 이 두 영상을 합친다.
그 다음의 처리는, clip 함수와 astype 함수를 사용하여 uint8 타입의 영상으로 다시 변환하는 처리이다. 이렇게 함으로써, 시연영상에 보았던 것과 같이 가중치가 0일 경우는 원본 영상이, 100 일 경우는 평활화 영상이 출력이 되며, 평활화의 반영 비율이 조절이 된다.
그리고 평활화가 수행되고, 반영 비율도 적용이 된 영상을 반환하면서 함수의 동작이 종료된다.

※ 여러 문제점들에 대한 설명은 본 보고서의 결론 및 후기 부분에 기술하였습니다.

< 프로그램 구현 >

4. 평활화된 영상에 언샤프 마스킹을 수행한 다음 출력한다. 이때, 시그마(sigma)나 강도(scale) 의 값이 0일 경우, 언샤프 마스킹을 수행하지 않고 출력한다.

```
103         # 언샤프 마스킹(UM) 미적용. 히스토그램 평활화(HE)만 적용
104         if (scale == 0) or (sigma == 0):
105             # 평활화 영상에 텍스트 추가
106             cv.putText(frameCeq, f"this_index = {count}",
107                        org, 0, 0.8, (0, 0, 255), 2)
108             cv.putText(frameCeq, f"sigma={sigma}, scale={scale}, weight={weight}",
109                        org2, 0, 0.8, (0, 0, 255), 2)
110             res = cv.hconcat([frameC, frameCeq])  # 출력될 결과 영상
111         # 언샤프 마스킹(UM) 적용.
112         else:
113             um = Unsharp_Masking(frameCeq, sigma, scale, frameCeq.dtype)
114             # 마스킹 영상에 텍스트 추가
115             cv.putText(um, f"this_index = {count}",
116                        org, 0, 0.8, (0, 0, 255), 2)
117             cv.putText(um, f"sigma={sigma}, scale={scale}, weight={weight}",
118                        org2, 0, 0.8, (0, 0, 255), 2)
119             res = cv.hconcat([frameC, um])      # 출력될 결과 영상
120
121         cv.imshow(Name, res)                    # 영상 출력
```

언샤프 마스킹(이하 UM)의 처리 과정으로, 두 개의 분기 중 첫 분기는, 과제의 조건으로 주어진 시그마나 강도의 값이 0일 경우, UM 을 수행하지 않는 부분으로, 평활화가 시행 된 변환영상에 텍스트를 추가하고, 원본 영상과 변환 영상을 hconcat 함수를 사용하여 나란히 배열하여 출력될 결과 영상을 생성한다.

두 번째 분기는, 두 값이 모두 0이 아닐 경우 즉, UM을 적용하는 경우로, UM함수에 매개변수로 평활화 영상과 시그마 값, 강도 값, 영상의 자료형을 전달하여 함수를 호출하여 UM을 처리한 뒤, 처리된 영상에 텍스트를 추가하고, 원본 영상과 변환 영상을 hconcat 함수를 사용하여 나란히 배열하여 출력될 결과 영상을 생성한다. 그 다음 결과 영상을 출력 윈도우에 출력한다.

※ UM 함수 구현 부분은 다음 페이지에 작성 되어있습니다.

< 프로그램 구현 >

< 언샤프 마스킹(UM) 함수 >

```
40  # 언샤프 마스킹 함수
41  def Unsharp_Masking(img, sigma, scale, dtype):
42      img = img / 255  # 프레임 화소 값 정규화
43      k = sigma * 6 + 1  # 커널의 크기
44      blur = cv.GaussianBlur(src=img, ksize=(k, k), sigmaX=sigma)  # 블러링 영상
45      um = img + scale * (img - blur)
46      um = np.clip(um * 255, 0, 255).astype(dtype)
47
48      return um
```

UM처리 함수이다. 먼저 UM에 대한 정의를 내리자면, 블러링된 영상을 이용하여 영상을 선명하게 만드는 알고리즘이 UM이며, 시그마를 이용하여 블러링 특성 제어를 통해 강조할 고주파 성분의 특성을 제어함으로써 선명한 영상을 얻어낸다.

본 함수의 구현은 예제 프로그램을 참조하였으며, 처리과정은 다음과 같다.
매개변수로 받은 영상을 255로 나누어 정규화하고, 시그마의 값 * 6 + 1을 커널의 크기로 설정하였다. 블러링 영상을 만드는 과정은 가우시안 블러링을 사용하였으며, 이렇게 얻어진 블러링 영상을 원본 영상에서 빼서 영상의 고주파 성분을 얻고, 원본 영상과 강도(scale), 고주파 성분을 곱하여 UM 이 시행된 영상을 얻는다. 현재의 영상은 부동소수 영상이므로, 정수형 영상으로 되돌린 뒤, 이 영상을 반환하면서 함수 동작이 종료된다.

< 프로그램 구현 >

5. 사용자의 키 입력을 기다린다.

```
123         # 키 입력 대기
124         key = cv.waitKey(1)
125         if key == 27:           # esc 입력 시 종료
126             print('\nTerminate Program...')
127             exit(0)
128         elif key == ord('s'):   # 저장
129             saveName = 'tmp.png'
130             cv.imwrite('./' + saveName, res)
131             if res is not None:
132                 print(f"\nFile name is {saveName}")
133                 print('\nImage write success!')
134             else:
135                 print('\nError Occured!')
136         elif key == 32:     # 스페이스바 입력 시 정지
137             print(f'\nPause!')
138             cv.waitKey(0)
```

사용자의 키 입력을 기다리는 부분으로, esc 입력 시 프로그램이 종료되고, s 입력 시, 현재 출력되고 있는 영상을 'tmp.png' 라는 이름의 파일로 현재의 위치에 저장한다. 스페이스 바를 입력할 경우, 일시정지를 하고, 아무키나 누르면 다시 동작하도록 waitKey 함수의 매개변수를 0 으로 전달하여 호출한다.

※ 이후의 소스코드들은 각종 출력문과 사용한 객체 반환 부분으로, 생략하였습니다.

※ 본 페이지부터는 결과 관측입니다.

※ 키 입력 부분에 관련해서는 명확한 제시가 어려워 영상에서 제시함을 사전에 밝힙니다.

< 결과 화면 >

1. HE weight 트랙바의 값이 0인 경우

모든 값이 0인 경우이다. 원본 영상과 같은 영상이 출력되는 것을 확인할 수 있다.
이는 처리 연산에서 가중치의 값이 0이면, 평활화된 결과값이 모두 0이 되어버리고, 원본영상의 값은 그대로 적용되는 연산을 사용하였기 때문이다.

2) 's' 키 입력 전 현재 폴더의 상태

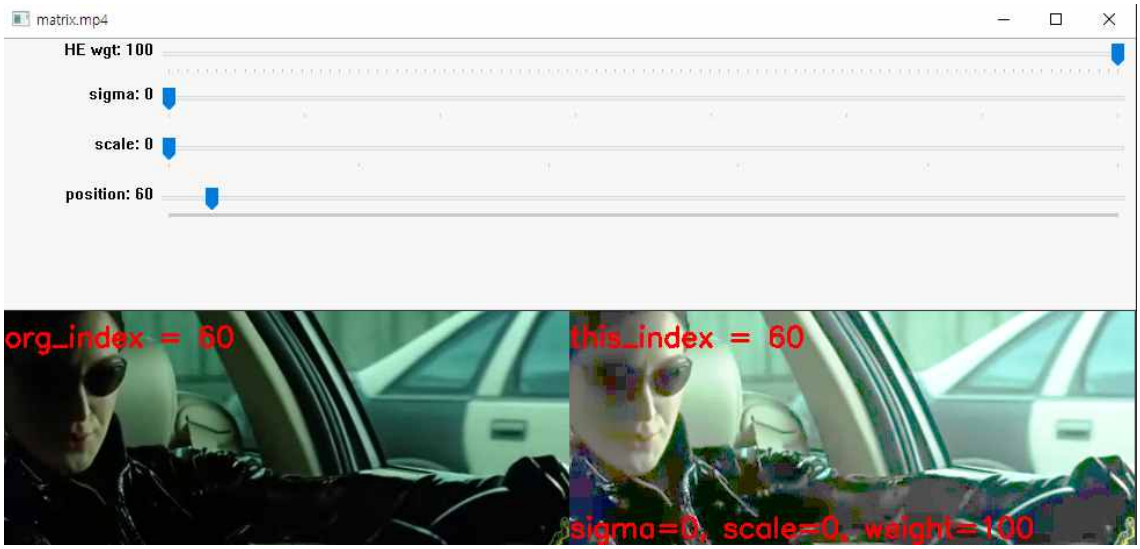
3) 's' 키 입력 후 현재 폴더의 상태

지정된 형식과 같은 이름으로, 현재의 폴더에 저장되었음을 확인하였다.

※ 내용중복 및 명확한 제시가 어렵기 때문에 이후 결과 화면에서는 키 입력에 대한 부분은 제외하고 기술하였습니다.

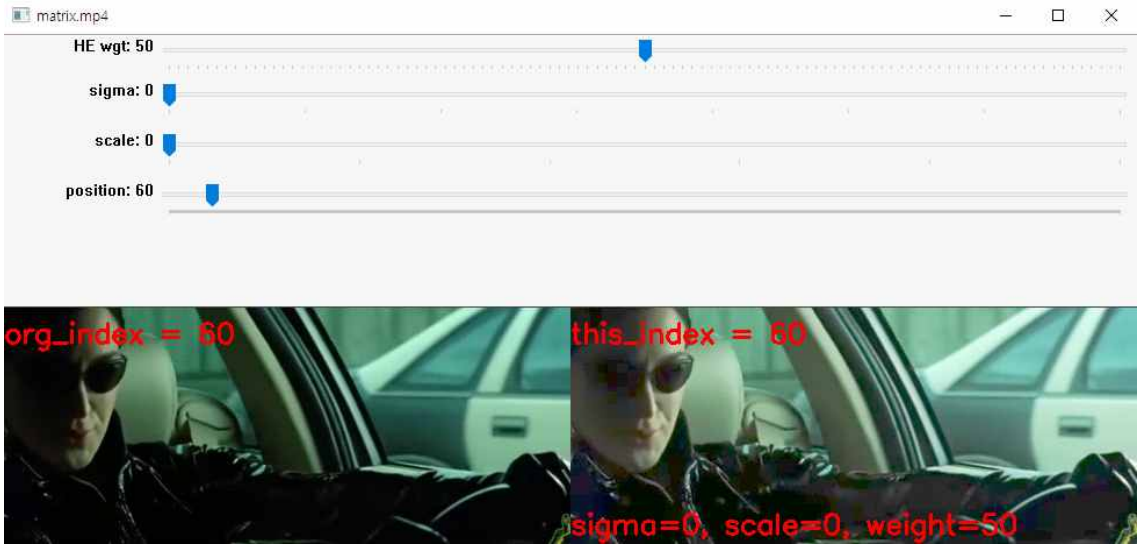
< 결과 화면 >

2. HE weight 트랙바의 값이 100인 경우



HE weight의 값이 100인 경우이다. 원본 영상과 다른, 평활화가 적용된 영상이 출력되는 것을 확인할 수 있다. 이는 앞서 기술한 처리 연산의 결과이다. 이때, 영상의 가시성이 향상되기는 커녕, 기존 영상보다 더 안 좋은 품질의 영상이 출력됨 또한 확인할 수 있다.

3. HE weight 트랙바의 값이 50인 경우



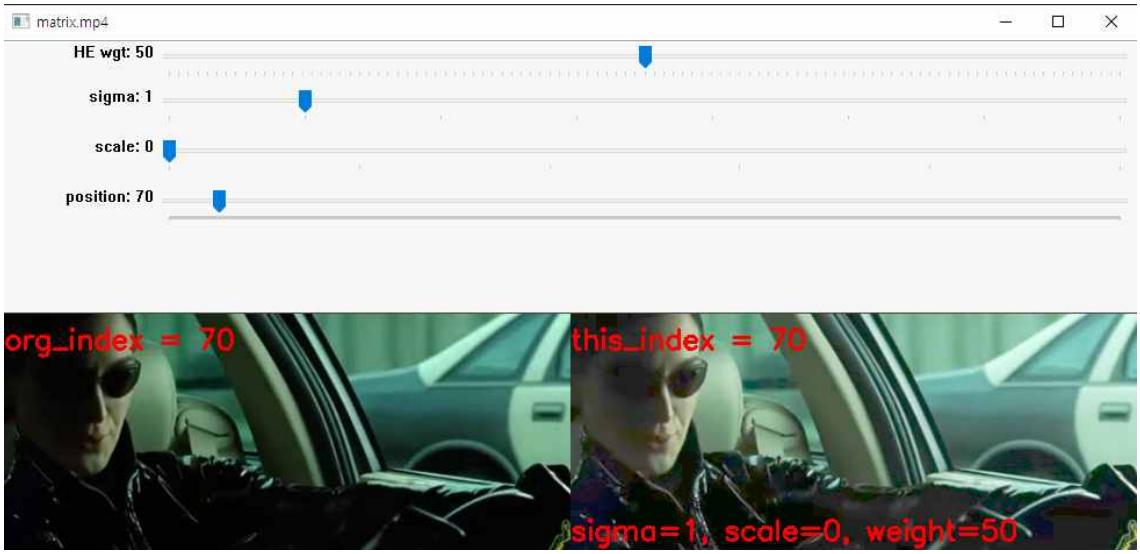
HE weight의 값이 50인 경우이다. 앞서 확인하였던 100인 경우보다 품질 저하가 덜하고, 원본 영상에 비해 비교적 선명한 영상이 얻어짐을 확인하였다.

또한, 위 세 경우 모두 텍스트가 정상적으로 추가되어있음을 확인할 수 있다.

※ 이후 UM 적용 결과는 모두 가중치가 50인 경우를 기준으로 테스트 하였습니다.

< 결과 화면 >

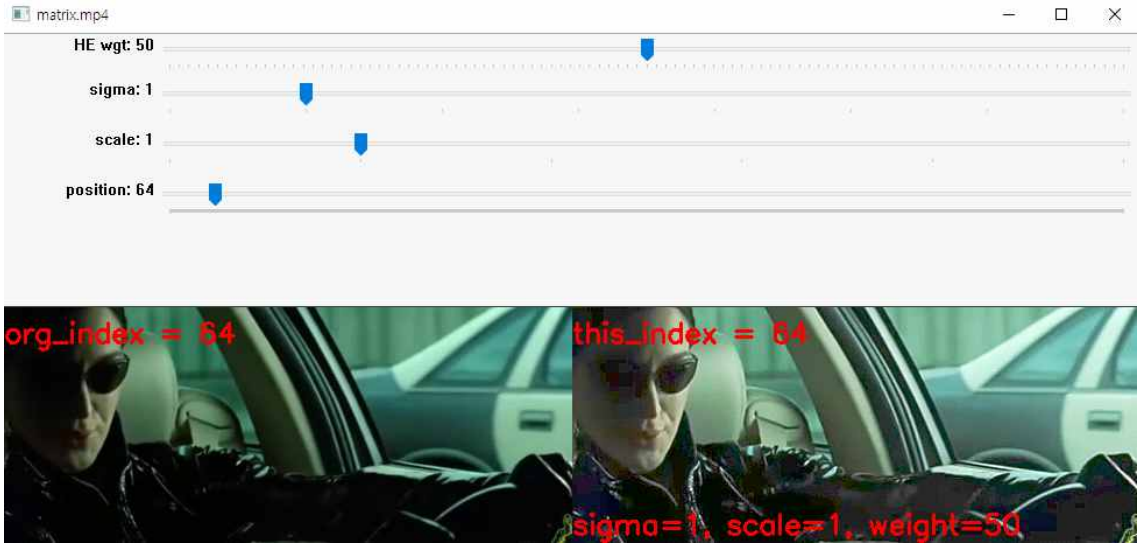
4. sigma 나 scale 트랙바의 값이 0인 경우



이전 영상과 비교해 보았을 때, 아무런 변화가 일어나지 않았음을 확인하였다. 따라서, 위 두 값 중 어느 하나라도 0인 경우에, UM이 시행되지 않음을 확인할 수 있다.

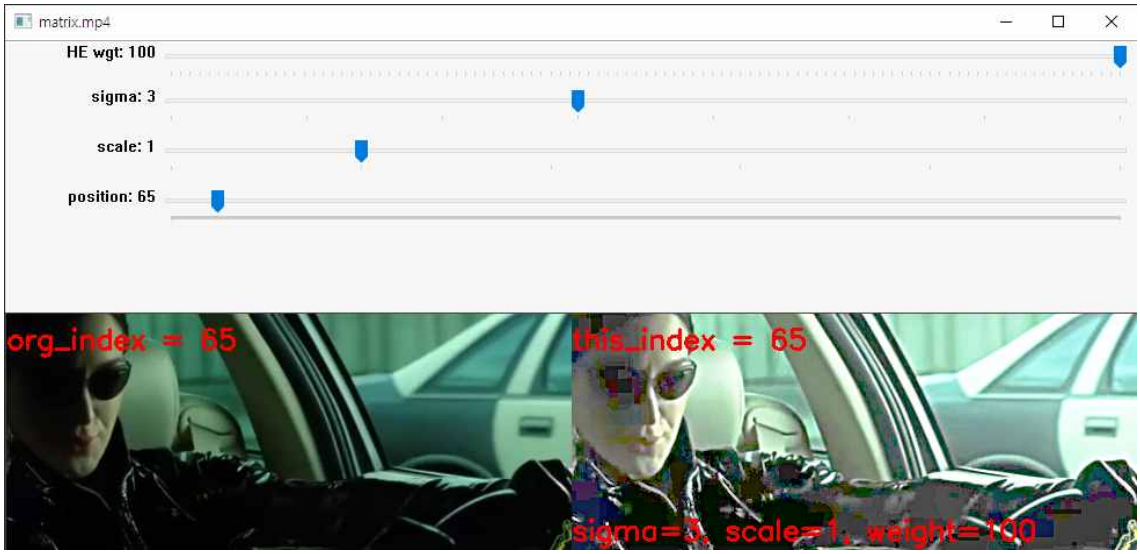
< 결과 화면 >

5. sigma = 1, scale = 1 인 경우



평활화만 시행 하였을 때의 영상보다, UM이 적용된 영상의 선글라스 부분의 품질이 좋지 않음을 확인하였고, 전체적으로 영상이 선명해졌다는 느낌을 받지 못하였다.

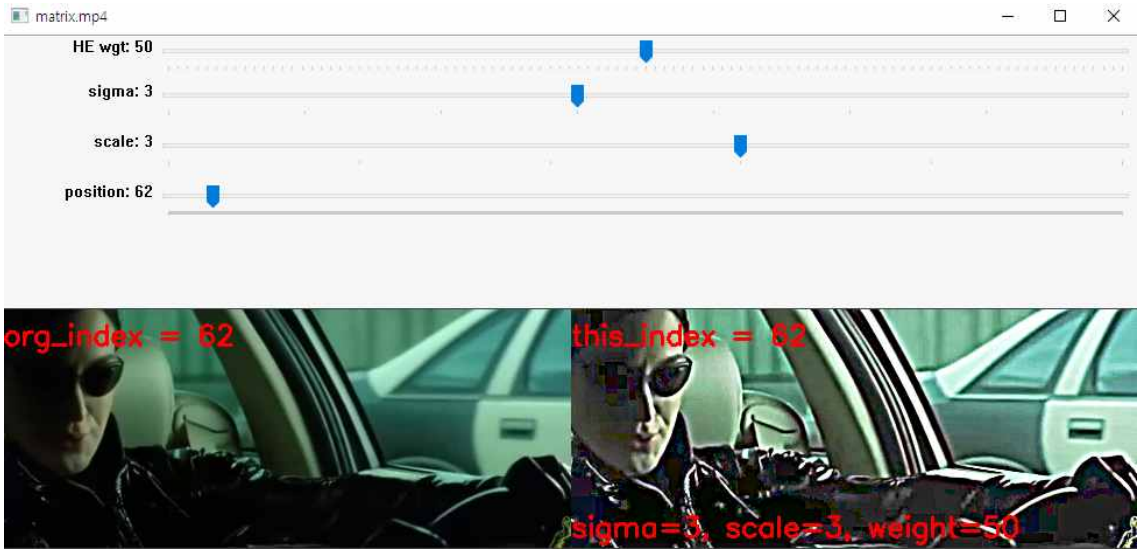
6. sigma = 3, scale = 1 인 경우



시그마의 값을 올려보았으나, 해당 프레임에서는 역효과를 낳았음이 확인되었다.

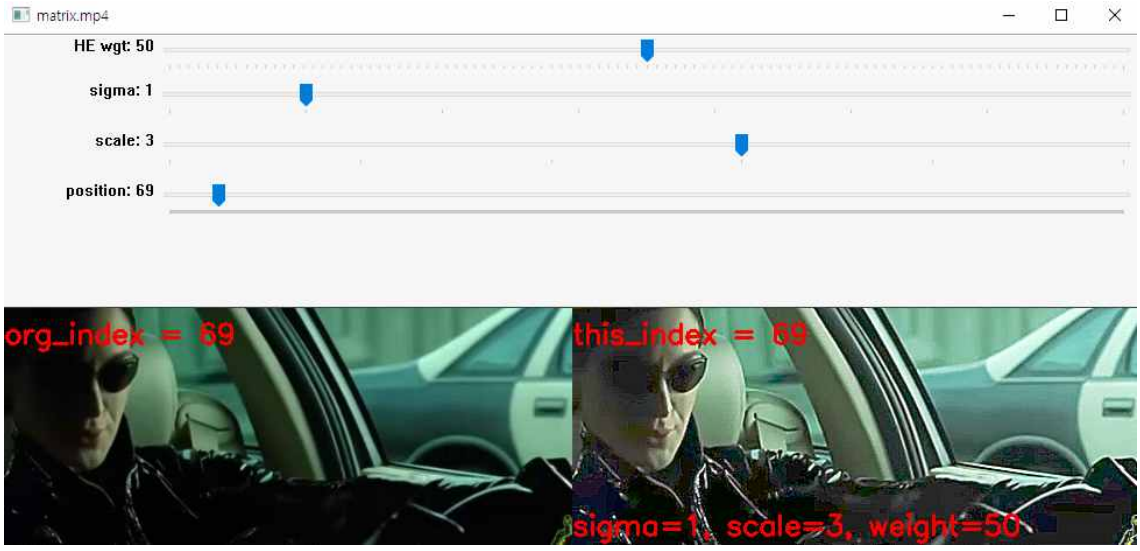
< 결과 화면 >

7. sigma = 3, scale = 3 인 경우



두 값을 모두 3으로 올린 경우, 이전 영상들에 비해 차량, 의상의 선명도는 올라갔으나, 사람에 대한 처리는 좋지 않음을 확인하였다.

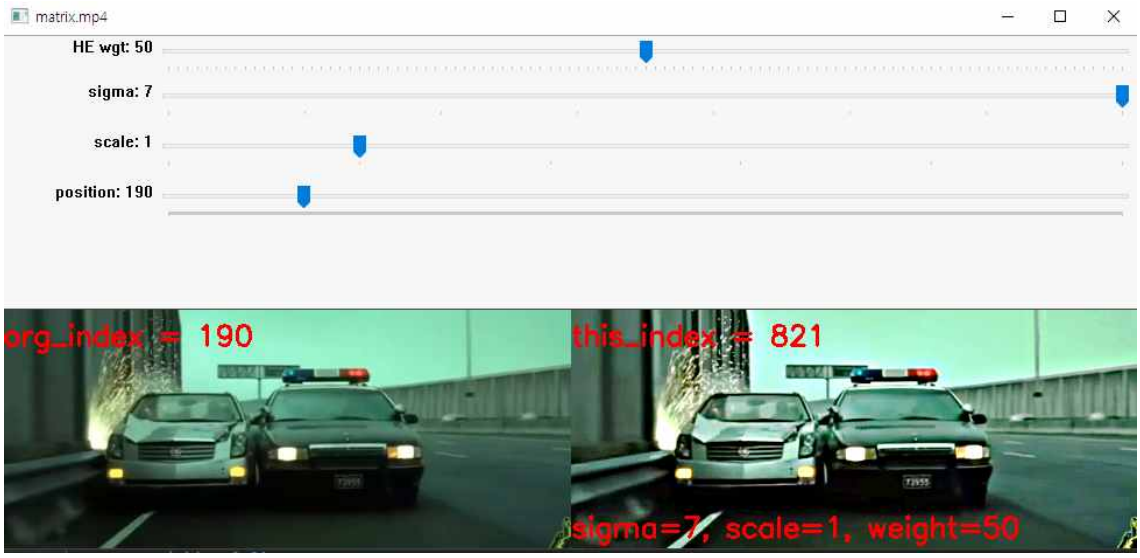
8. sigma = 1, scale = 3 인 경우



시그마의 값을 다시 1로 내린 뒤의 테스트 결과로, 지금까지 관측한 모든 결과들에 비해 사람의 얼굴이 가장 선명하게 변환이 되었으며, 차량이나 의상의 선명도는 두 값 모두 3인 경우에 비해 약간 떨어지는 것으로 관측되었다.

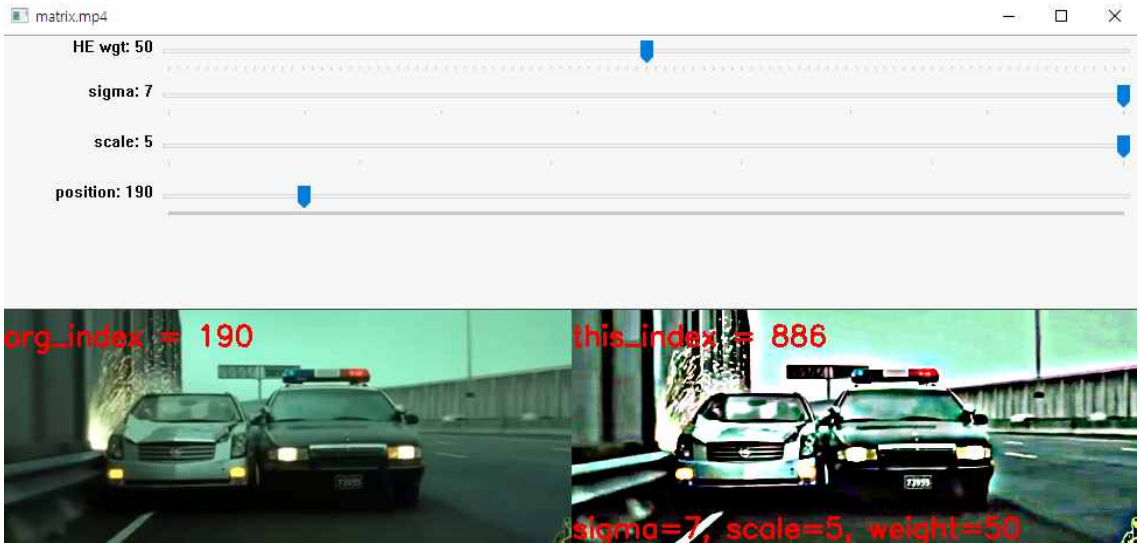
< 결과 화면 >

9. sigma = 7, scale = 1 인 경우



여러 테스트 결과, 이전에 테스트했던 영상에 대한 선명도는 확실하게 확인이 불가능하였으며, 이에 따라 차량이 등장하는 프레임에 대한 테스트를 진행하였다.
시그마의 값을 최대로, 강도의 값을 최소로 줄여보았으며, 차량에 대한 선명도는 눈에 띄게 증가하였음을 확인할 수 있다.

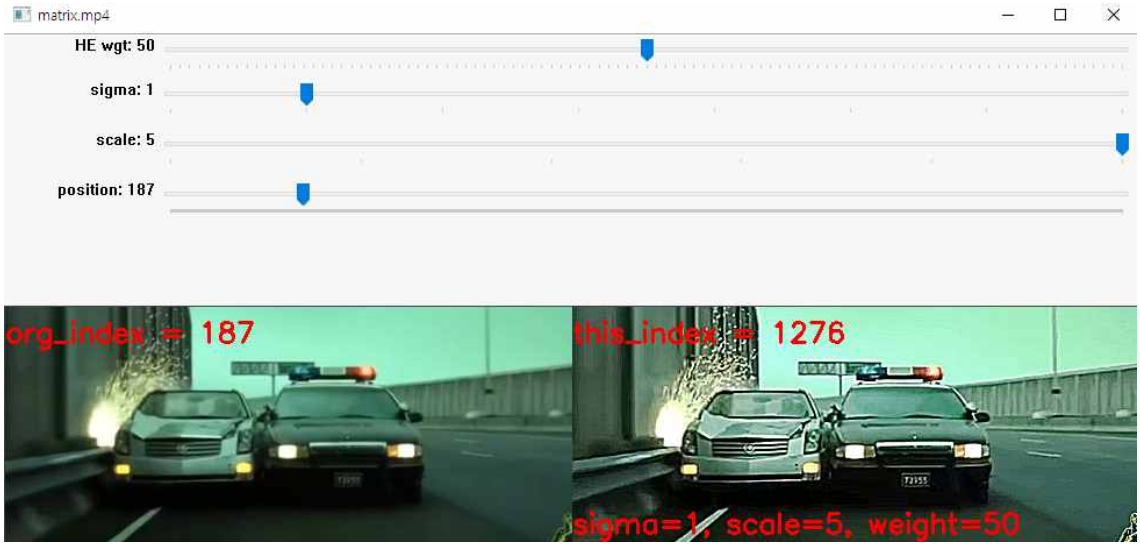
10. sigma = 7, scale = 5 인 경우



두 값 모두 최대로 올린 경우이다. 이전 영상에 비해 훨씬 지나치게 강조가 되었다.

< 결과 화면 >

11. sigma = 1, scale = 5 인 경우



시그마의 값을 1로, 강도를 최대로 올린 경우이다. 이 경우도 9번의 경우와 같이 상당히 선명해졌음을 알 수 있으며, 개인적인 시각에서, 시그마가 최대인 경우보다 보기 좋게 선명해졌다는 느낌을 받았다.

< 결론 및 후기 >

1. 결론

(1) HE 관련 결론

앞서 기술한 바와 같이, HE는 영상 내 모든 화소의 분포를 균등하게 만듦으로써 영상의 가시성을 높이는 작업이다. 따라서 테스트의 결과는 가시성이 향상된 영상이 출력될 것이라고 추측이 가능하다. 그러나, 앞서 살펴본 결과와 같이, HE의 문제점이 드러나게 되었는데, 원본영상에 비해 밝기만 증가하고 품질은 오히려 떨어지는 경우가 발생했음을 확인할 수 있었다. 따라서 과제의 조건대로 평활화 영상의 반영 비율을 가중치를 통하여 적절히 조절함으로써 위와 같은 문제가 해결 가능함을 확인하였다.

(2) UM 관련 결론

앞서 기술한 바와 같이, UM은 블러링된 영상을 이용하여 영상을 선명하게 만드는 알고리즘이며, 시그마를 이용하여 블러링 특성 제어를 통해 강조할 고주파 성분의 특성을 제어함으로써 선명한 영상을 얻어낸다. 테스트 결과, sigma값의 변화에 따라 영상의 경계 부분이 더 뚜렷하게 구분이 되는 것을 확인할 수 있으며, scale값의 변화에 따라 영상의 경계선이 더 날카롭게 변하는 것을 확인할 수 있다. 위 두 변화로 인하여 영상의 선명도는 증가하게 된다. 그러나 이 역시 HE와 동일하게 값이 클수록 좋은 것이 아니라, 적절한 값을 조절하여야 한다는 것을 확인하였다.

2. 후기

(1) HE 관련 시행착오

가중평균 하여 평활화 영상의 비율을 반영하라는 과제의 조건에 따라, 가중평균의 개념과 식을 먼저 확인 후 진행하였는데, 가중평균을 구하는 식은 다음과 같다.

$$\text{Weighted Average} = \frac{w_1 \times x_1 + w_2 \times x_2 + \cdots + w_n \times x_n}{w_1 + w_2 + \cdots + w_n}$$

* W_n : X_n 의 가중치 (weight)

위 식을 통하여 가중평균의 값을 구할 때, 여러가지 문제점들이 발생하였고, 결국 앞서 프로그램 구현 부분에 기술된 바와 같이 가중치의 값만 사용하여 평활화 영상에는 가중치 값을 곱하고, 원본영상은 (1-가중치) 값을 곱한 뒤, 둘을 더해서 출력하는 방식을 채택하였다.
문제점을 크게 두 가지로 분류하면 다음과 같다.(가중치의 값은 0.01이 곱해진 값으로, 0~1의 값)

1. 위 식에서의 X를 본 과제에선 어떤 것으로 해야하는가?
2. 어디에 적용해야하는가?

우선 위 식에서의 X는 원본과 변환 영상 둘 중 하나일 것이다. 그러나 위 두 경우 중 그 어떤것도 정상적인 결과를 내지 못하였다. 우선 가장 큰 문제 중 하나로, 위와 같은 가중평균식이 영상에서도 적용이 가능한 문제인지에 대한 의문을 가지게 되었으나, 해보기전엔 알 수 없으므로,
두 경우 모두의 가중평균 값을 구한 뒤, 평활화 영상을 출력하는 부분에서 다양하게 적용을 해보았으나, 가중치가 0인 경우에는 가중평균 식의 분모가 0이 되는 문제점, 가중치가 1인 경우에 가중평균의 값이 1이 되어야 하지만, 모든 픽셀 화소의 값이 같지 않으면, 그럴 수 없는 문제점 등이 발견이 되었다.

어디에 적용해야 하는가에 대한 문제는 출력이 될 평활화 영상에 적용을 하면 되는 것이므로, 쉽게 해결이 가능하였다. 가장 큰 문제는 가중 평균 값이 사용 가능한지에 대한 문제였다.
앞서 기술된 문제점들 이외에도 수많은 계산과 시도가 있었으나, 결국 이 시도들에 대한 결론은 실패였으며, 그 과정에서 얻은 해결책이 앞서 기술된 방식인 가중치 값만을 사용한 반영 비율 처리였으며, 해당 방식 적용 시 시연영상과 유사한 결과가 출력이 됨을 확인할 수 있었다.

가중평균 필터 사용도 조사해보았으나, 해당 방식은 영상의 노이즈 제거로 사용될 뿐, 영상 비율 반영에는 사용되지 않는 것으로 판단되었다.

따라서 원본 영상과 평활화 영상을 가중평균하여 비율을 조절하는 것은 가중평균의 값을 사용할 수 없다는 결론에 도달하였으며, 앞서 기술한 바와 같이 프로그램을 구현하였다.

(2) 종합

히스토그램 평활화와 언샤프 마스킹 모두 이론과 예제로만 경험한 채로 지나감에 따라, 정확하게 어떤 동작을 하는지, 각 값이 변화할 때 어떤 변화가 일어나는지에 대한 명확한 이해가 부족하였으나, 과제와 보고서 작성을 통해서 보다 깊게 공부하고 이해할 수 있게 된 계기였다.

< 평가표 >

※ 각 미션에 대한 평가만 작성하였습니다.

1. 동영상을 읽어 들여 이 영상에 대한 처리를 트랙 바로 제어하는 모습을 보인다. 답변 : 프로그램 구현에서 해당 과정을 상세히 설명하였습니다.
2. 원본 영상과 화질개선한 영상을 나란히 배열하여 출력한다. 좌측: 원본 동영상, 우측: 화질 개선 동영상. 답변 : 프로그램 구현 부분에서 해당 과정을 상세히 설명하였으며, 출력 결과에서 해당 부분을 확인하였습니다.
3. 트랙 바의 설치는 공고문 및 시연 동영상 자료에 따른다. 답변 : 프로그램 구현 부분에서 해당 과정을 상세히 설명하였으며, 출력 결과에서 해당 부분을 확인하였습니다.
4. 히스토그램 평활화만 혹은 Unsharp Masking만 수행되도록 트랙 바를 설정할 수 있다. 이때의 처리 결과가 정상적이다. 답변 : 출력 결과에서 해당 부분을 확인하였으며, 영상에서 더 명확하게 확인이 가능합니다.
5. 히스토그램 평활화가 수행된 후 원본 영상과의 가중 평균한 값이 Unsharp Masking의 입력으로 사용된다. 답변 : 프로그램 구현 부분에서 해당 과정을 상세히 설명하였으며, 출력 결과에서 해당 부분을 확인하였습니다.
6. key='s'를 입력하면 전체 영상(좌+우)을 글씨까지 포함하여 현재 폴더에 파일이름='tmp.png'로 저장된다. 답변 : 프로그램 구현 부분에서 해당 과정을 상세히 설명하였으나, 출력 결과에서 해당 부분에 대한 명확한 확인은 불가하므로, 영상에서 해당 내용을 명확히 확인할 수 있습니다.
7. 우측화면에 트랙 바의 설정 값(sigma, scale, weight) 올바르게 출력된다. 답변 : 프로그램 구현 부분에서 해당 과정을 상세히 설명하였으나, 출력 결과에서 해당 부분을 확인하였습니다.
8. 트랙 바의 position을 지정하면 동영상의 프레임 번호가 이에 따라 바뀐다. 답변 : 정상적으로 동작함을 확인하였습니다. 영상에서 해당 내용을 명확히 확인할 수 있습니다.
9. space바로 정지하고, 아무 키나 누르면 계속 진행, esc 키를 입력하면 종료한다. 답변 : 정상적으로 동작함을 확인하였습니다. 영상에서 해당 내용을 명확히 확인할 수 있습니다.