[redacted]

W9 영상 필터링

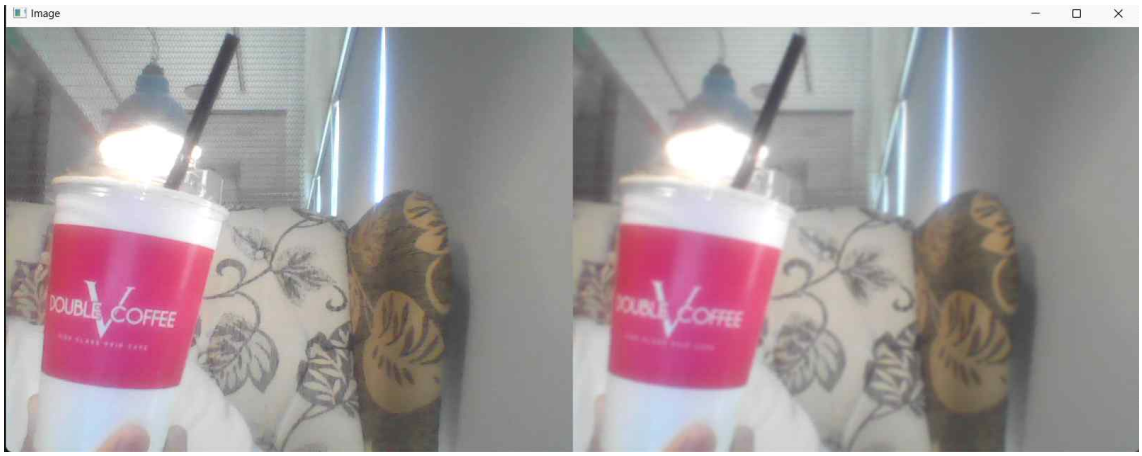
개요

1. 예제코드 실행 결과사진
2. 입력방식을 변경한 코드 첨부
3. 가우시안 잡음이 추가된 영상에 대한 가우시안 필터와 평균필터
    - 실험1 강의와 동일한 kernel size와 gain
    - 실험2 noise의 gain증가
    - 실험3 커널사이즈의 증가
        - <왜 평균필터에서 빨대가 커 보이는가?>

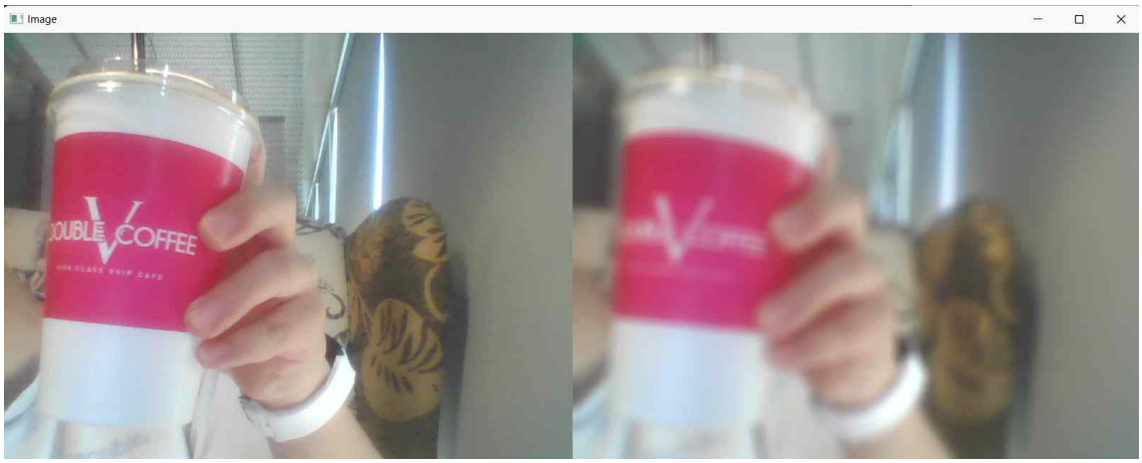
4. 공간 라플라시안 필터, sobel, canny필터
    - 4방향 공간필터와 8방향 공간필터비교
    - 필터의 원소의 부호를 변경해 실험한 결과 비교
    - sobel, canny필터와 라플라시안 공산필터 비교
        - <우측 하단의 빛이 추가적으로 엣지 검출되다.>
    - 왜 라플라시안 필터가 가장 좋았는가?
5. 예제에 대한 추가실험 salt & pepper noise
    - 가우시안 필터와 미디안 필터의 차이 비교
    - 인위 적인 점을 필터링으로 지울 수 있는가?
    - 컵위의 점은 왜 유일하게 선명함을 유지하는가?

과제 1 예제코드 실행결과

예제1 공간 필터링



<Kernel size = 5>



<kernel size = 15>

예제2 LPF의 noise제거

<Kernel size = 5 , gain = 22>

<kernel size = 11, gain = 22 >

예제3 LPF의 noise제거 <color>

<kernel size = 5, gain = 22 >

<kernel size = 11, gain = 22 >

예제4 unsharp_mask

예제5 median filter

<kernel size = 5, ratio = 0.1 ,salt>

예제6 2D highpass filter

예제7 sobel& canny filter

과제2 Cam 입력을 파일입력으로 변경<결과, Source code>

예제1 공간 필터링

Source code

```
import cv2
import numpy as np
ksize =int(input('Enter kernel size:'))
kernel = np.ones((ksize, ksize), np.float32) / (ksize * ksize)
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기
frame = cv2.imread(image_path)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
# 이미지 해상도 설정을 유지
width =640
height =480
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
# 이미지에 필터 적용
filtered_frame = cv2.filter2D(frame, -1, kernel)
# 원본 이미지와 필터링된 이미지를 가로로 나란히 붙여서 보여줌
cframe = np.hstack((frame, filtered_frame))
cv2.imshow('Image', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

예제2 LPF의 noise제거

Source code

```
import cv2
import numpy as np
ksize =int(input('Enter kernel size:'))
kernel = np.ones((ksize, ksize), np.float32)/(ksize*ksize)
ngain = float(input('Enter noise gain:'))
width =320
height =240
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
# 이미지를 흑백으로 변환
img = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# 노이즈 추가
noisy = np.clip(img + np.random.random((height, width)) * ngain, 0,
255).astype(np.uint8)
# 평균 필터 적용
filtered_img = cv2.filter2D(noisy, -1, kernel)
# 이미지들을 가로로 나란히 붙여서 보여줌
cframe = np.hstack((img, noisy, filtered_img))
cv2.imshow('Original, Noisy, Filtered', cframe)
# 'q' 키를 누르면 종료
whileTrue:
    key = cv2.waitKey(33)
    if key == ord('q'):
        break
cv2.destroyAllWindows()
```

예제3 LPF의 noise제거 <color>

Source code

<평균필터>

```
import cv2
import numpy as np
ksize =int(input('Enter kernel size:'))
kernel = np.ones((ksize, ksize), np.float32)/(ksize*ksize)
ngain = float(input('Enter noise gain:'))
width =560
height =720
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
noisy = np.clip(Y + np.random.random((height, width))*ngain, 0, 255).astype(np.uint8)
filtered = cv2.filter2D(noisy, -1, kernel)
cnoisy = cv2.cvtColor(cv2.merge((noisy, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cfiltered = cv2.cvtColor(cv2.merge((filtered, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, cnoisy, cfiltered))
cv2.imshow('Original, Noisy, Filtered', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

<가우시안 필터>

```
import cv2
import numpy as np
ksize =int(input('Enter kernel size:'))  # 가우시안 필터 크기 입력, 일반적으로 홀수 사용
ngain = float(input('Enter noise gain:'))  # 노이즈 게인 입력
width =560
height =720
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로를 여기에 입력하세요
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
# 이미지를 흑백으로 변환
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
# 노이즈 추가
noisy = np.clip(Y + np.random.random((height, width)) * ngain, 0,
255).astype(np.uint8)

filtered = cv2.GaussianBlur(noisy, (ksize, ksize), 0)
# 이미지들을 가로로 나란히 붙여서 보여줌
cnoisy = cv2.cvtColor(cv2.merge((noisy, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cfiltered = cv2.cvtColor(cv2.merge((filtered, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, cnoisy, cfiltered))
cv2.imshow('Original, Noisy, Gaussian-filtered', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key is ord('q'):
    cv2.destroyAllWindows()
```

예제4 unsharp_mask

Source code

```
import cv2
import numpy as np
width =560
height =720 # 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
blur = cv2.GaussianBlur(Y, (0, 0), 2)
filtered_Y = np.clip(2.0* Y - blur, 0, 255).astype(np.uint8)
cfiltered = cv2.cvtColor(cv2.merge((filtered_Y, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, cfiltered))
cv2.imshow('Original, Unsharp-mask', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

예제5 median filter

Source code

```
# 255 = 흰점 salt  , 0 은 검은점 pepper
import cv2
import random
import numpy as np
ksize =int(input('Enter kernel size:'))
rat_noise = float(input('Enter frequency of noise:'))
width =560
height =720
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
num_noise =int(width*height*rat_noise)
# Salt noise 추가
for i inrange(num_noise):
    y = random.randint(0, height-1)
    x = random.randint(0, width-1)
    Y[y][x] =255  # 흰 점 (Salt noise)
filtered = cv2.medianBlur(Y, ksize)
cnoisy = cv2.cvtColor(cv2.merge((Y, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cfiltered = cv2.cvtColor(cv2.merge((filtered, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, cnoisy, cfiltered))
cv2.imshow('Original, Noisy, Median-filtered', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

예제6 2D highpass filter

Source code

```
import cv2
import numpy as np
width =320
height =240
k_lap = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
k_lap_e = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
Yf_lap = cv2.filter2D(Y, -1, k_lap)
Yf_lap_e = cv2.filter2D(Y, -1, k_lap_e)
img_lap = cv2.cvtColor(cv2.merge((Yf_lap, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
img_lap_e = cv2.cvtColor(cv2.merge((Yf_lap_e, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, img_lap, img_lap_e))
cv2.imshow('High-pass filtered results', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

Source code

<흑백 전처리 과정 추가>

```
import cv2
import numpy as np
width =320
height =240
k_lap = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
k_lap_e = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]])
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
# 이미지를 흑백으로 변환
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# 라플라시안 필터 적용
gray_lap = cv2.filter2D(gray_frame, -1, k_lap)
gray_lap_e = cv2.filter2D(gray_frame, -1, k_lap_e)
# 결과 이미지 병합 및 표시
result_frame = np.hstack((gray_frame, gray_lap, gray_lap_e))
cv2.imshow('Original, Laplacian 4-neighborhood, Laplacian 8-neighborhood',
result_frame)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

예제7 sobel& canny filter

Source code

```
import numpy as np
width =320
height =240
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로를 여기에 입력하세요
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
sobel = cv2.Sobel(gray, cv2.CV_8U, 1, 0, 3)
canny = cv2.Canny(gray, 50, 150)
cframe = np.hstack((gray, sobel, canny))
cv2.imshow('High-pass filtered results', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)
if key == ord('q'):
    cv2.destroyAllWindows()
```

3. 가우시안 잡음이 추가된 영상<평균 필터, 가우시안 필터>

이론적 배경

평균 필터는 커널 내의 모든 값의 평균을 계산하여 적용한다. 이로인해 이미지는 부드럽게 만들지만, 엣지를 포함한 이미지의 세부정보가 흐려질 수 있다.

가우시안 필터의 경우 커널 중앙에 더 큰 가중치를 두고, 주변으로 갈수록 가중치가 감소하는 방식으로 가중 평균을 계산한다.
따라서 평균 필터에 비해 엣지를 더 잘 보존 하면서 이미지를 부드럽게 할 수 있다.

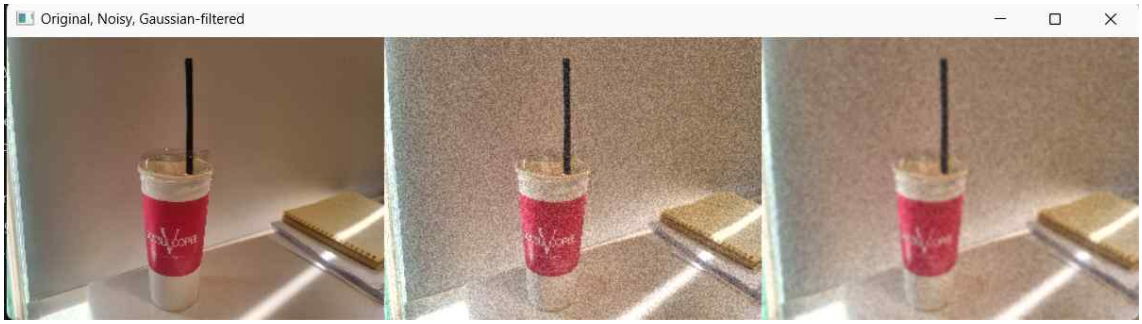
실험1

<gaussian filter , kernel size = 5, gain = 22>

<평균 filter , kernel size = 5, gain = 22>

두 필터를 비교할 때 가우시안 필터에 비해 평균필터의 결과가 좀더 흐릿해 보이는 것을 확인 할 수 있었다. 두 필터 모두 잡음은 어느정도 제거된 것을 확인 할 수 있다.

실험2 noise의 gain증가



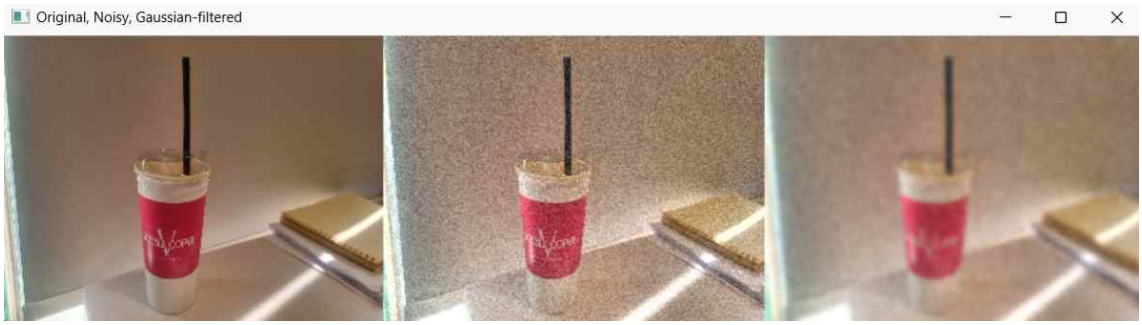
<gaussian filter , kernel size = 5, gain = 60>

<평균 filter , kernel size = 5, gain = 60>

noise의 gain을 증가시키자 두 필터 모두 필터가 적용된 결과에 노이즈가 남아있는 것을 확인할 수 있다.
이는 평균 필터의 경우 모든 픽셀에 동일한 가중치를 부여하는 특성으로인해 극단적으로 잡음을 올린 해당 실험에서 잡음을 제거하는 성능은 좋았다고 볼 수 있다.
다만 노이즈뿐만 아니라 실제 이미지의 정보에도 동일한 가중치를 부여하는 특성으로 인해 원본의 사진도 같이 흐리게 보였다.

즉 잡음이 극단적으로 커진 경우 잡음을 제거하는 성능은 평균필터가 더 좋았지만, 원본 이미지도 흐리게 만드는 단점이 존재하였다. 가우시안 필터는 반대로 잡음이 커지자 비교적 성능은 좋지 않았지만 원본의 데이터를 유지하였다.

실험3 커널 사이즈 증가



<gaussian filter , kernel size = 11, gain = 60>

<평균 filter , kernel size = 11, gain = 60>

평균필터의 결과를 보면 앞선 결과들과 같이 원본의 이미지가 가우시안 필터보다 흐렸으며 커널의 크기가 커져 더욱 흐려진 것을 확인할 수 있다.
하지만 빨대의 부분과 컵의 빨간 부분의 경계가 더욱 선명해져 보이는 느낌을 받을 수 있으며 **빨대는 원본의 이미지보다 크기가 커보이는 효과**를 얻게 되었다.

왜 평균 필터의 경우 빨대가 커 보이는가?

빨대의 경우 검은색이며 빨대 바로 옆의 배경은 비교적 하얀색이다. 평균필터의 특성으로 밝은 부분과 어두운 부분의 픽셀값이 평균화 되며 엣지의 밝은 부분과 어두운 부분이 중간 값인 회색을 띌 것이다. 이 가중치가 적용되며 엣지의 한쪽이 다른 쪽으로 확산되어 엣지가 넓게 보이는 효과를 얻은 것이다. 이로인해 빨대가 더 두꺼워 보이는 결과가 나왔다고 볼 수 있다.

빨간 부분이 선명하게 보이는 느낌을 받는 이유도 컵의 중심부 빨간 색 바로 주변은 색이 비교적 더 밝은 흰 색이라 이부분의 엣지가 강조되어 보이는 느낌을 받았다고 볼 수 있다.

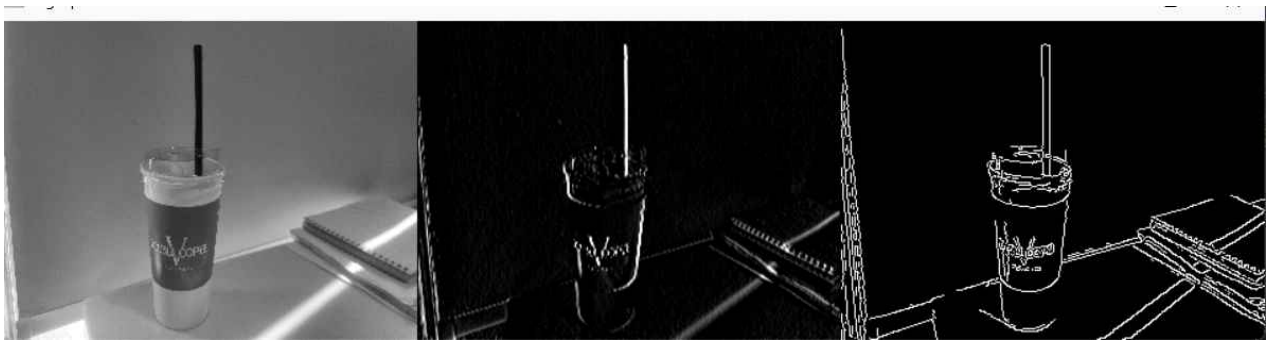
4. 공간 라플라시안 필터 , sobel, canny

실험1 공간 라플라시안 필터

<original, 4방향 Edge mask , 8방향 Edge mask>

<original 흑백버전, 4방향 Edge mask , 8방향 Edge mask>

<original 흑백버전, 부호변경한 4방향 Edge mask , 부호변경한 8방향 Edge mask>



<original, soble_filter , canny_filter>

정확한 비교를 위해 라플라시안 공간필터도 흑백이미지로 변환하는 전처리 과정을 거쳐 공간필터링을 진행하였다.

4방향 공간필터링 과 8방향공간 필터링

칼라 이미지와 흑백 이미지의 모두 4방향 엣지 마스크를 적용하였을 때 보다 8방향 엣지 마스크를 적용한 결과가 더욱 선명한 엣지 검출이 이루어 진 것을 확인 할 수 있다.
이는 수직, 수평 그래디언트 뿐만아니라 대각선 방향의 성분도 고려하며 중심의 가중치가 -8로 중심픽셀의 영향을 주변보다 훨씬 강하게 감소시키는 효과를 가져 더 선명한 것을 알 수 있다.

필터의 원소들의 부호를 변경

```
height = 240
k_lap = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
k_lap_e = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
```

해당 알고리즘은 중앙픽셀에 주변보다 작은 가중치를 주어 엣지를 검출하는 방식을 사용한다. 따라서 기존의 필터의 원소의 부호들을 서로 변경하여 새로운 실험을 진행하였다.
필터의 원소가 중심이 음수고 주변이 양수였던 기존 필터와 반대로 중심이 양수고 주변인 음수인 필터를 적용하였을 때 비슷한 결과였지만 기존의 라플라시안 공간필터에 비해 번짐이 더 적고 선명한 느낌을 받았다.

이는 기존의 필터는 중심을 낮추고 주변을 집중적으로 보며 차이를 확인한다면, 수정한 필터는 중심에 큰 가중치를 주어 중심을 집중적으로 주변과의 차이를 보는 차이에서 발생한다고 생각된다.

sobel filter와 canny filter<우측하단의 빛이 추가 검출>

soble필터와 canny filter의 결과에서 눈에 띄는 차이는 바로 컵 우측하단의 빛이였다.
라플라시안 필터에서 검출되지 않았던 빛이 두 필터에서는 엣지 검출이 되었다.

라플라시안 필터는 2차 미분을 이용하여 밝기의 변화율을 측정해 변화가 급격할 때 가장 큰 값을 반환한다.

반면sobel의 경우 1차 미분을 사용하여 밝기변화의 경사를 측정하여 연속적으로 변화하는 밝기 경사도를 감지할 수 있어 우측 하단의 빛과 같은 점진적인 밝기 변화도 감지가 가능하다. 또한 canny필터 역시 1차 미분을 활용하여 우측하단의 빛이 엣지검출이 되었다고 볼 수 있다.

sobel과 canny에서 sobel필터는 물체의 윤곽이 부분부분 끊겨서 검출되었지만 canny 필터의 경우 더 진하게 이어진 형태의 엣지 검출이 된 것을 확인 할 수 있다.
이는 이중 임계값 처리 및 히스테리시스 엣지 트래킹으로 canny 필터가 sobel필터와 다르게 약한 값이라고 강한 엣지와 연결된 값이면 선택을 하여 연속성을 보장해주기 때문이다.

왜 라플라시안 필터가 가장 좋았는가?

HPF인 라플라시안 공간필터는 잡음을 강조해 버리는 문제가 있다.
이를 보완하기 위해 sobel 필터가 개발 되었으며 canny필터는 sobel필터에 연속성을 보장해 주기위해 개발이 되었다. 하지만 실험결과 가장 결과가 좋은 것은 라플라시안 필터였다.
라플라시안 필터가 급격한 밝기 변화를 감지하는데 효과적이기 때문에 잡음이 없는 환경에서 가장 좋은 결과가 나온 것이다.

즉 부분부분 새어들어오는 빛으로 인해 이러한 점진적인 변화마저 검출해버리는 Canny filter의 특성으로 인해 원하지 않는 엣지가 검출되어 가장 선명하지만 가장 깔끔하지 않게 엣지가 검출되었다고 볼 수 있다.

결론

에지 검출이란 화소의 밝기가 변하는 부분을 검출하는 것이다.
영상안에서 밝기의 차이가 있는 곳은 일반적으로 물체의 윤곽선에 해당한다.
일반적으로 라플라시안 필터는 2차 미분을 활용하여 급격한 밝기 변화 검출에 효과적이다.
단점으로는 잡음을 강화하는 문제가 있다.

sobel 과 canny는 1차 미분을 활용하며 점진적인 밝기 변화 검출에 효율적이다.
이러한 이유로 빛의 반사와 같은 부분마저 검출이 되어 원하지 않는 엣지가 검출 될 수 있다.

5. 예제에 대한 추가 실험

1. salt & pepper을 가우시안과 미디안 필터적용

<original, salt noise0.1 , gaussian_filter(kernel size=3)>

<original, salt noise0.1 , median_filter(kernel size=3)>

pepper & salt noise 는 불특정한 위치에 검은색, 하얀색 점으로 잡음이 나타난다.
가우시안 필터는 선형필터이기 때문에 극단적인 값을 평균계산에 포함하게 되어 노이즈가 퍼진듯한 느낌을 주었다.

미디안 필터의 경우 주변 픽셀들의 중간값으로 각 픽셀의 값을 대체 하여 극단적 값이 추가된 pepper & salt 노이즈 제거에 효과 적임을 확인 할 수 있다.
다만 전체적인 사진의 색이 흰색이여서 완벽하게 노이즈가 제거 되지 않은 것을 확인 할 수 있다.

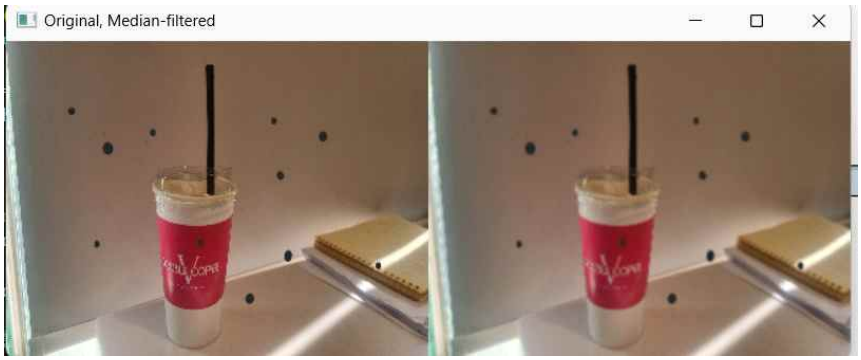
<original, pepper noise0.1 , median_filter(kernel size=3)>

같은 이유로 pepper noise의 경우 제거되지 않은 노이즈가 검은색인 빨대 주변에 많이 분포 하는 것을 확인 할 수 있다.

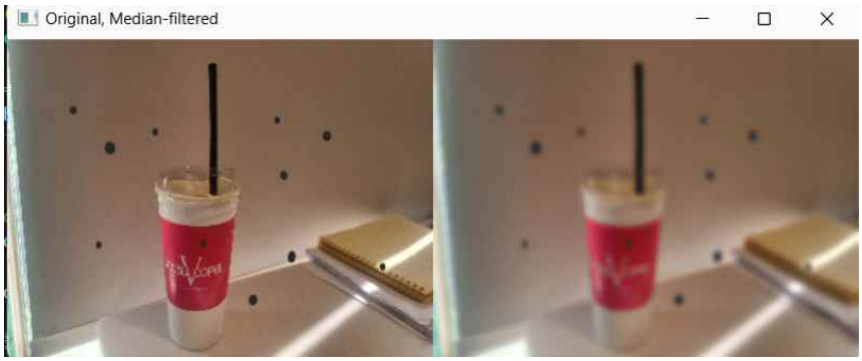
2. 인위적인 점을 필터적용

셀카 어플에 사용되는 필터와 유사한 기능의 실험에 흥미를 느끼고
인위적으로 사진에 점을 찍어 이를 필터링 할 수 잇는지를 테스트 해보았다.

가우시안 필터



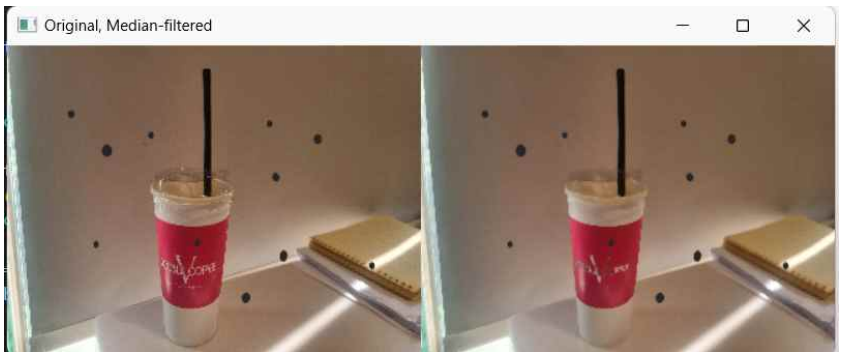
<kernel size = 3>



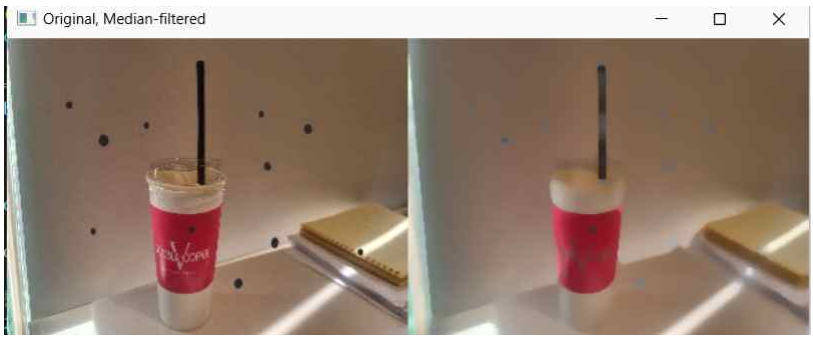
<kernel size = 11>

<kernel size = 21>

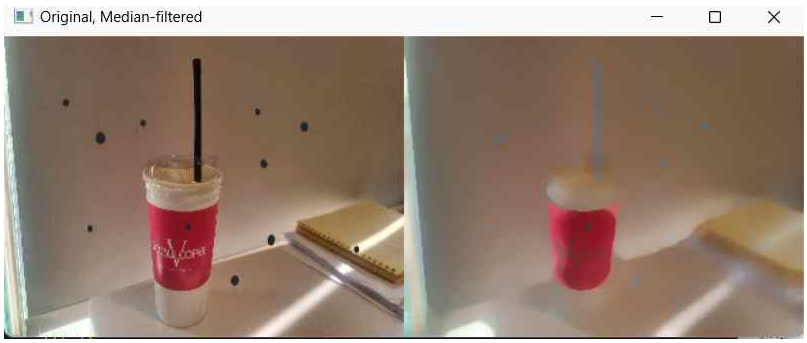
미디안 필터



<kernel size = 3>



<kernel size = 11>



<kernel size = 21>

가우시안 필터는 전체이미지가 흐려질뿐 커널의 크기와 상관없이 인위적인 점만을 제거하지는 못하였다.

미디안 필터의 경우는 점을 어느정도 제거하는데 효과가 있었다.
kernel size가 3일때는 그려둔 점의 크기가 커서 계산되는 중간값에 영향을 주었는지 점이 지워지지 않았다.
kernel size를 11, 21로 크게하자 점이 많이 지워졌으나 전체적인 이미지도 흐려졌다.
또한 컵의 로고가 흰색에서 어두운 색으로 변화하였으며 빨대의 색은 반대로 흰색에 가까워진 것을 확인할 수 있었다.

유일하게 선명한 컵위의 점

두 필터 모두 커널이 증가함에 따라 전체적인 이미지가 흐려지고 인위적인 점이 제거 되는 상황에도 컵에 그려진 점은 비교적 선명한 상태를 유지하였다.
해당점은 유알하게 주변이 빨간색인 점이였다.
필터가 주변 픽셀들의 값에 영향을 받아 두 필터에 사용된 중간값이나 평균값이 점의 색상에 가까워 해당 점만이 지워지지 않았다고 생각된다.

해당 추가 실험을 통해 미디안 필터의 특성과 필터에서 커널의 영향을 자세히 학습할 수 있었다.

Source code

```
import cv2
import numpy as np
ksize =int(input('Enter kernel size:'))
width =320
height =240
# 이미지 파일 경로 직접 설정
image_path ='D:/code/mmp/w9/data/test.jpg'  # 이미지 경로를 여기에 입력하세요
# 이미지 파일 읽기 및 크기 조정
frame = cv2.imread(image_path, cv2.IMREAD_COLOR)
if frame isNone:
    print("Error: Image could not be read. Check the path.")
    exit()
frame = cv2.resize(frame, (width, height))  # 이미지를 지정된 width와 height로 조정
t_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(t_frame)
#가우시안 필터 적용
filtered = cv2.GaussianBlur(Y, (ksize, ksize), 0)
```

```
# 미디안 필터 적용
#filtered = cv2.medianBlur(Y, ksize)
cfiltered = cv2.cvtColor(cv2.merge((filtered, Cr, Cb)), cv2.COLOR_YCrCb2BGR)
cframe = np.hstack((frame, cfiltered))
cv2.imshow('Original, Median-filtered', cframe)
# 'q' 키를 누르면 종료
key = cv2.waitKey(0)  # 대기 시간을 무한대로 설정하여 사용자가 키를 누를 때까지 기다립니다.
if key == ord('q'):
    cv2.destroyAllWindows()
```