미니프로젝트
(Ver 1.0 스핀오프)

디지털 영상 처리
GrayScale Image Processing

2024. 3 . 20 – 2024. 3 . 22
[Intel] 엣지 AI SW 아카데미-절차지향 프로그래밍

차창섭

목차

프로젝트

목표 와 환경

목표

1.디지털 영상처리 기술 학습

2.C언어로 구현한 디지털 영상처리 프로그램 Python으로 구현

환경

OS : Window 10 pro 64bit

Tool : Visual Studio 2022 preview

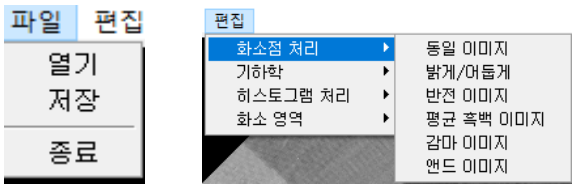
Cpu : Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz

RAM: ddr4 2400MHz 16GB

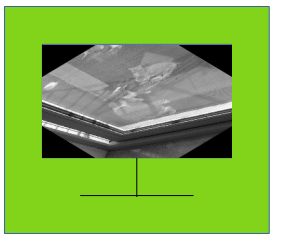
구성도

- 메뉴창 생성
- 원본 파일 데이터 로드
- 데이터 크기에 따라 메모리 확보
- 전역변수로 변환
- 영상처리 알고리즘 사용 하여 영상처리 구현
- 구현된 이미지 출력 및 저장

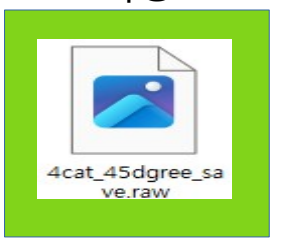
메뉴창



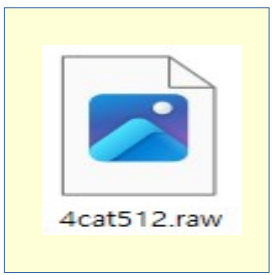
출력



저장



원본

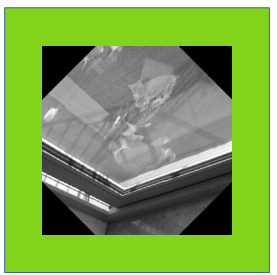


LoadImage
로드

inImage

알고리즘
적용

outImage



디지털
영상 처리

1.화소 점 처리

2.기하학 처리

3.히스토그램 처리

4.화소 영역 처리

1.화소 점 처리

1-1.산술 연산 밝기 조절

1-2.AND 연산

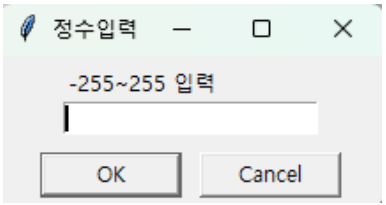
1-3.평균값 흑백 처리

1-4.평균값 흑백 처리

1-5.영상 반전 변화

1-1.산술 연산 밝기 조절
1-2.AND 연산

입력값(value)에 따라 밝기 조절




```
px = inImage[i][k] + value
if (px > 255) :
    px = 255
if (px < 0) :
    px = 0
outImage[i][k] = px
```

입력 값에 따라 AND 연산 수행

```
outImage[i][k] = inImage[i][k] & value
```

원본

밝게

어둡게

AND 연산 100

1-3.영상 반전 변화

8비트 Image의 화소 최댓값은 255

Output(q) = 255 - Input(p)

```
outImage[i][k] = 255 - inImage[i][k]
```

원본

반전

1-4.평균값 흑백 처리

화소값이 화소 평균값 128 보다 같거나 크면 최댓값 255 작으면 0

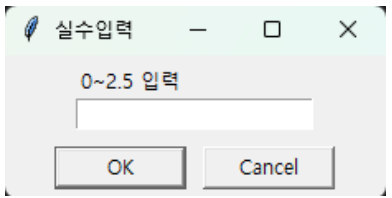
```
if (inImage[i][k] >= 128):
        outImage[i][k] = 255
```

```
if (inImage[i][k] < 128):
        outImage[i][k] = 0
```

흑백 처리

1-5.감마 보정

입력값(value)에 따라 감마 조절



$$y = M\left(\frac{x}{M}\right)^{g}$$

```
outImage[i][k] = int(((inImage[i][k]/255)**(1/value))*255)
```

원본

0.5

1.3

2.기하학 처리

영상을 구성하는 화소의 공간적 위치를 재배치 하는 과정

2-1.축소 스케일링 변화

2-2.영상 확대 스케일링

2-3.이동 기하학 변환

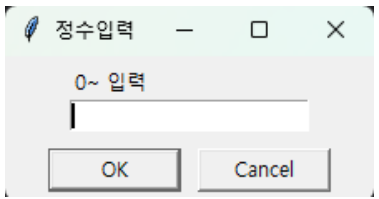
2-4.좌우 대칭 변환

2-5.상하 대칭 변환

2-6.회전 변환

2-1.축소 스케일링 변화
2-2.영상 확대 스케일링

입력값(value)에 따라 축소




```
outImage[int(i/value)][int(k/value)] = inImage[i][k]
```

입력값(value)에 따라 확대

```
outImage[i][k] = inImage[int(i/value)][int(k/value)]
```

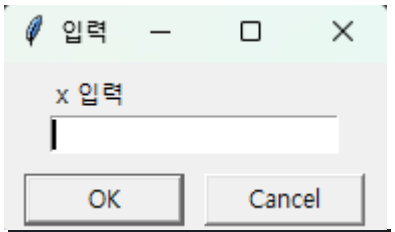
원본

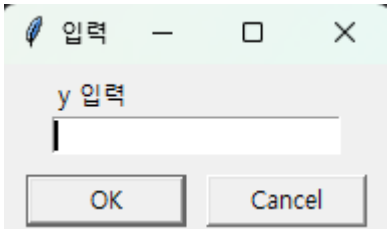
2배 확대

2배 축소

2-3.이동 기하학 변환

디지털 영상의 크기나 형태 등이 바뀌지 않고 평면의 한 위치에서 원하는 다른 위치로 옮기는 연산
수평(x),수직(y) 입력값(value)에 따라 이동





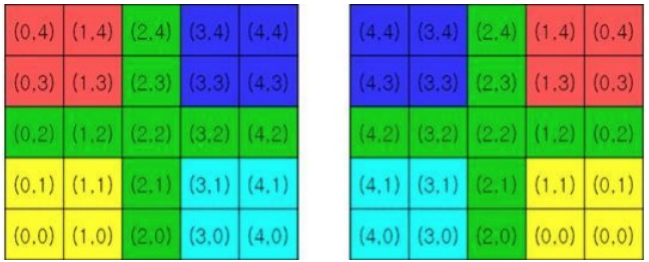

```
tx, ty = value1, value2
tty = inH-ty
ttx = inW-tx
```

```
outImage[int(i + ty)][int(k + tx)] = inImage[i][k]
```

수직 수평
10 이동

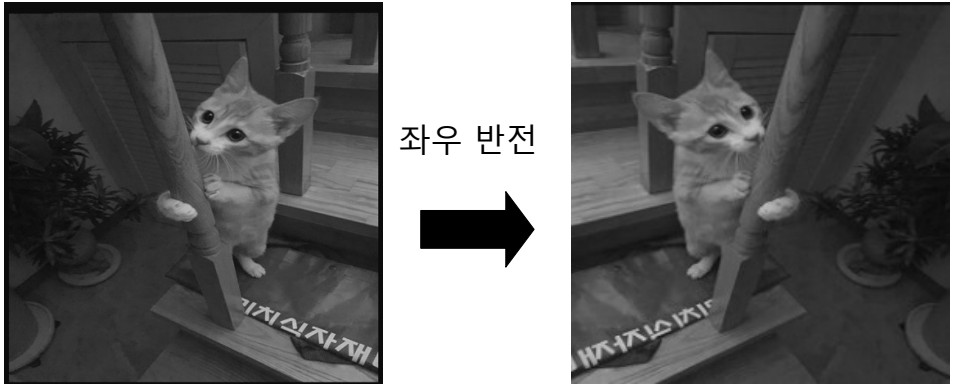
2-4.좌우 대칭 변환

영상 내의 한 수직선을 중심으로 왼쪽 화소와 오른쪽 화소 교환



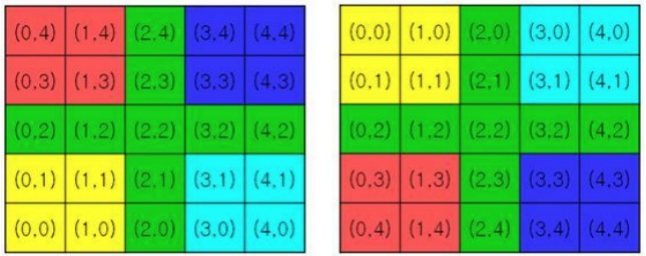
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -(x - x_0) \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ 0 \end{bmatrix}$$

```
mk = inW - 1 - k
outImage[i][k] = inImage[i][mk]
```



2-5.상하 대칭 변환

영상 내의 한 수직선을 중심으로 위쪽 화소와 아래쪽 화소 교환



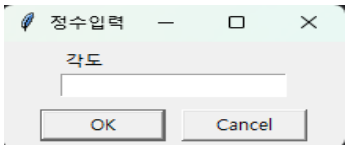
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ -(y - y_0) \end{bmatrix} + \begin{bmatrix} 0 \\ y_0 \end{bmatrix}$$

```
mi = inW - 1 - i
outImage[i][k] = inImage[mi][k]
```

2-6.회전 변환

영상을 올바르게 회전하도록 하려면 화면 좌표를 수학적 좌표로 변환하여 회전 한 뒤 다시 화면 좌표로 변환

입력값(degree)에 따라 각도 조절 `radian = float(degree * 3.141592 / 180.0)`



역방향 사상 공식 사용

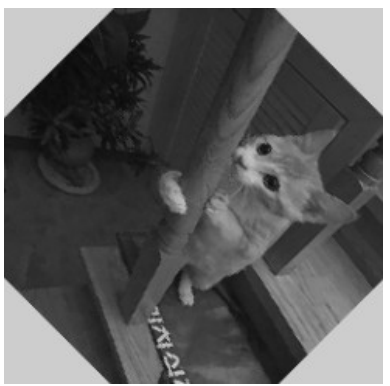
- **역방향 사상 공식**

$$x_{source} = (x_{dest} - C_x)\cos\theta + ((H_y - y_{dest}) - C_y)\sin\theta + C_x$$
$$y_{source} = -(x_{dest} - C_x)\sin\theta + (H_y - ((H_y - y_{dest}) - C_y)\cos\theta) + C_y$$

```
cx = int(inH / 2)
cy = int(inW / 2)
```

```
xd = i
yd = k
xs = int(math.cos(radian) * (xd - cx) + math.sin(radian) * (yd - cy))
ys = int(-math.sin(radian) * (xd - cx) + math.cos(radian) * (yd - cy))
xs += cx
ys += cy
if ((0 <= xs < outH) and (0 <= ys < outW)) :
    outImage[xd][yd] = inImage[xs][ys]
```

45도 회전



3.히스토그램 처리

- 3-1.히스토그램 스트레칭
- 3-2.앤드 인 탐색
- 3-3.히스토그램 평활화

3-1.히스토그램 스트레칭

이상적이지 못한 히스토그램 분포 중에서 명암 대비가 낮은 디지털 영상의 품질을 향상 시키는 기술

$$new\ pixel = \frac{old\ \ pixel - low}{high - low} \times 255$$

```
if (inImage[i][k] < low) :
    low = inImage[i][k]
if (inImage[i][k] > high) :
    high = inImage[i][k]
```

```
new = int(float(old - low) / float(high - low) * 255)
if (new > 255) :
    new = 255
if (new < 0) :
    new = 0
outImage[i][k] = new
```

3-2.앤드 인 탐색

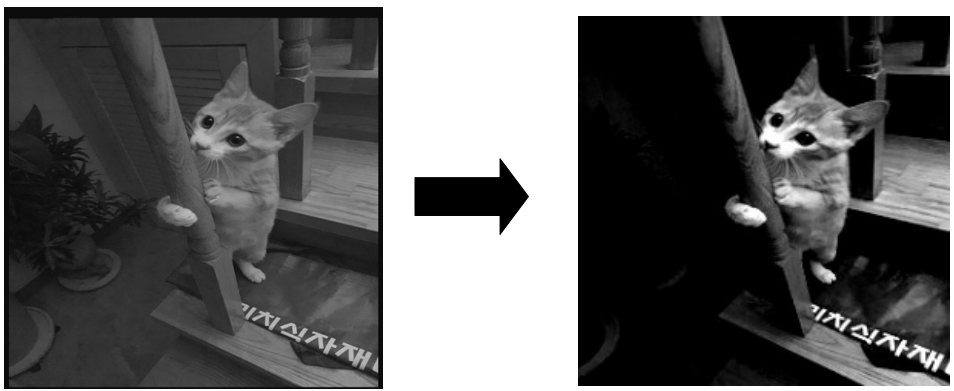
히스토그램의 분포를 균일하게 만들기 위해 일정한 양의 화소를 흰색 혹은 검정색으로 지정

$$new\ pixel = \begin{cases} 0 & old\ pixel \le low \\ \dfrac{old\ pixel - low}{high - low} \times 255 & low \le old\ pixel \le high \\ 255 & high \le old\ pixel \end{cases}$$

```
if (inImage[i][k] < low) :
    low = inImage[i][k]
if (inImage[i][k] > high) :
    high = inImage[i][k]
```

```
high -= 50
low += 50
```

```
old = inImage[i][k]
new = int(float(old - low) / float(high - low) * 255)
if (new > 255) :
    new = 255
if (new < 0) :
    new = 0
```



3-3.히스토그램 평활화

어둡게 촬영된 영상의 히스토그램을 조절해 명암 분포가 빈약한 영상을 균일하게 만들어 줌

정규화 누적합 공식 $n[i] = sum[i] \times \frac{1}{N} \times I_{max}$

1단계 빈도수 세기

```
histo[inImage[i][k]] += 1
```

2단계 히스토그램 누적합

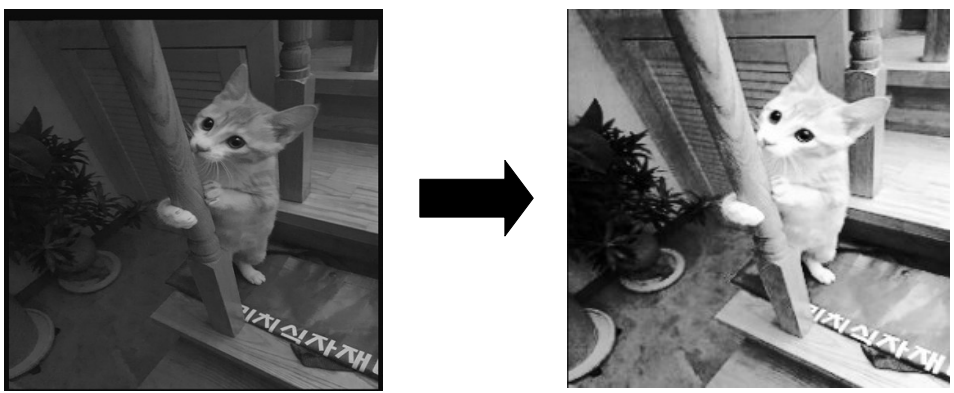
```
sumHisto[i] = sumHisto[i - 1] + histo[i]
```

3단계 정규화된 히스토그램 생성

```
normalHisto[i] = sumHisto[i]*(1.0/(inH*inW))*255.0
```

4단계 inImage를 정규화된 값으로 치환

```
outImage[i][k] = int(normalHisto[inImage[i][k]])
```



4.화소 영역 처리

회선 기법으로 수행하므로 화소의 영역 처리를 회선 처리라고 함

4-1.회선 수행 방법

4-2.엠보싱

4-3.블러링

4-4.샤프닝

4-5.에지

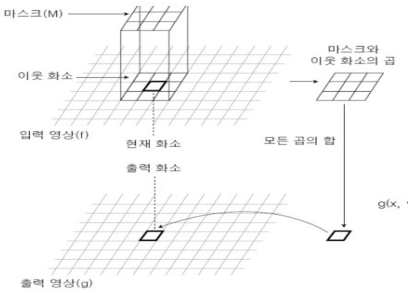
4-5-1.연산자 에지

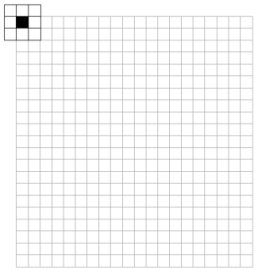
4-5-2.1차 미분을 이용한 에지

4-5-3.2차 미분을 이용한 에지

4-1.회선 수행 방법

가중치를 포함한 회선 마스크가 이동하면서 수행
회선 마스크가 왼쪽 위 화소에서 오른쪽 으로 한 화소 씩 이동하며 수행하여 새로운 화소 생선 한 줄이 끝나면 다음 줄 수행





1.빈도수 세기

```
mask = [[-1.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 1.0]]
```

2단계 마스크 크기에 맞게 Input 주위를 넓혀 준 임시 출력 영상 만들기

```
tmpInImage = malloc2D(inH + 2, inW + 2)
tmpOutImage = malloc2D(outH, outW)
```

3.임시 입력 메모리

```
tmpInImage[i + 1][k + 1] = inImage[i][k]
```

4.회선 연산

```
        S += tmpInImage[i + m][k + n] * mask[m][n]
tmpOutImage[i][k] = S
```

5.임시 출력 영상 출력 영상으로 전환

```
if (tmpOutImage[i][k] < 0.0) :
    outImage[i][k] = 0
else :
    if (tmpOutImage[i][k] > 255.0) :
        outImage[i][k] = 255
    else :
            outImage[i][k] = int(tmpOutImage[i][k])
```

4-2.엠보싱

가운데에 있는 계수가 다른 계수를 상쇄하게 구성해 양각 한 효과가 있는 경계선 검출

```
mask = [[-1.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 1.0]]
```

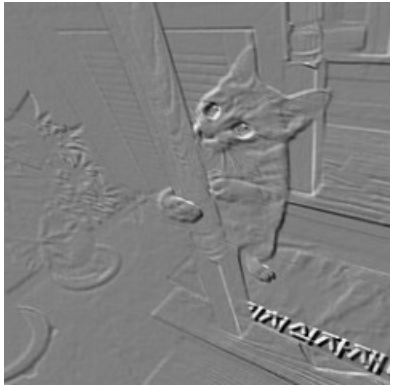
4-3.블러링

극단적인 값 제거 후 평균값으로 대처하여 영상을 흐리게 하는 기술

```
mask = [[1./9, 1./9, 1./9], [1./9, 1./9, 1./9], [1./9, 1./9, 1./9]]
```

원본

엠보싱



블러링

4-4.샤프닝

블러링 과 정반대 효과 증가
흐린 영상을 선명하게 하는 효과

일반 샤프닝 마스크

```
mask = [[0. ,-1. , 0.], [-1., 5., -1.], [0. ,-1. , 0.]]
```

고주파 샤프닝 마스크

```
mask = [[-1. / 9, -1. / 9, -1. / 9], [-1. / 9, 8. / 9, -1. / 9], [-1. / 9, -1. / 9, -1. / 9]]
```

저주파 샤프닝 마스크

```
mask = [[1. / 9, 1. / 9, 1. / 9], [1. / 9, 1. / 9, 1. / 9], [1. / 9, 1. / 9, 1. / 9]]
```

저주파 샤프닝 회선여산

$$h(x,y) = (1+\alpha)f(x,y) - \alpha \cdot G_{\sigma}(f(x,y))$$

```
        S += tmpInImage[i + m][k + n] * mask[m][n]
tmpOutImage[i][k] = S
hx = val * inImage[i][k] - S * (val - 1)
tmpOutImage[i][k] = hx
```

원본

일반

고주파

저주파

4-5.에지

화소가 낮은 값에서 높은 값으로 또는 높은 값에서 낮은 값으로 극격히 변하는 구간

4-5-1.연산자 에지

유사 연산자: 화소를 감산한 값에서 최대값을 결정하 여 에지를 검출

차 연산자: 유사 연산자의 계산 시간이 오래 걸리는 단점을 보안해 주는 방법

유사 연산자

$$\text{New Pixel} = \max(|\text{center} - m_1| \cdots |\text{center} - m_8|)$$

총 8번 계산

```
if (int(tmpInImage[i + 1][k + 1] - tmpInImage[i + m][k + n] >= max)) :
    max = int(tmpInImage[i + 1][k + 1] - tmpInImage[i + m][k + n])
```

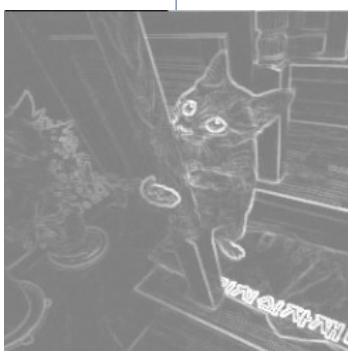
차 연산자

$$\text{New Pixel} = \max(|m_1 - m_9|, |m_3 - m_7|, |m_2 - m_8|, |m_6 - m_4|)$$

```
mask[0] = int(tmpInImage[i][k] - tmpInImage[i + 2][k + 2])
mask[1] = int(tmpInImage[i][k + 1] - tmpInImage[i + 2][k + 1])
mask[2] = int(tmpInImage[i][k + 2] - tmpInImage[i + 2][k])
mask[3] = int(tmpInImage[i + 1][k + 2] - tmpInImage[i + 1][k])
```

원본

유사



차

1차 미분을 이용한 에지

에지 추출에는 함수의 변화분을 찾는 미분 연산이 이용됨

1차 미분 에지 검출 : 로버츠 , 프리윗 , 소벨

로버츠 에지

장점: 빠른 속도로 동작

단점: 돌출된 값을 잘 평균할 수 없으며, 잡음에 민감함

행+열 마스크

```
[-1.0,0.0,-1.0],
[ 0.0,2.0, 0.0],
[ 0.0,0.0, 0.0]
```

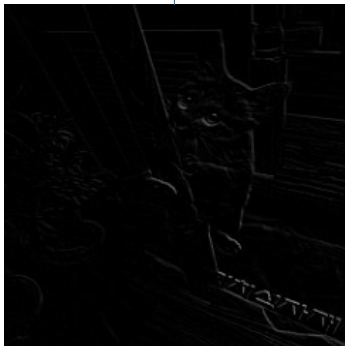
열 마스크

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

행 마스크

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

원본



로버츠 행

로버츠 열

로버츠 행 +열

1차 미분을 이용한 에지

프리윗 에지

장점: 돌출된 값을 비교적 잘 평균화함

단점: 대각선보다 수평,수직에 놓인 에지에 더 민감히 반응

행 마스크

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

열 마스크

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

행+열 마스크

```
[[0.0,-1.0,-2.0],
 [1.0, 0.0,-1.0],
 [2.0, 1.0, 0.0]]
```

원본

프리윗 행

프리윗 열

프리윗 행+열

1차 미분을 이용한 에지

소벨 에지

장점: 돌출된 값을 비교적 잘 평균화함

단점: 대각선 방향에 놓인 에지에 더 민감히 반응

행 마스크

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

열 마스크

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

행+열 마스크

```
[[0.0,-1.0,-2.0],
 [1.0, 0.0,-1.0],
 [2.0, 1.0, 0.0]]
```

원본

소벨 행

소벨 열

소벨 행+열

2차 미분을 이용한 에지

1차 미분의 단점을 완화시켜 둔감하게 반응하도록 만듦
2차 미분 엣지 검출 : 라플라시안 , Log , Dog

라플라시안 에지

장점: 검출된 엣지를 끊거나 하지 않고
연결된 폐곡선을 형성
단점: 잡음에 민감,윤곽의 방향 구하지 못함

마스크1

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

마스크2

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

마스크3

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

원본

마스크1

마스크2

마스크3

2차 미분을 이용한 에지

Log

잡음에 민감한 라플라시안의 문제점을 해결하기 위해 만듦
가우시안 스무딩을 수행하여 잡음 제거후 라플라시안 사용

Dog

계산 시간이 긴 Log의 문제점을 해결하기 위해 만듦
각 가우시안 연산에 분산 값을 서로 다르게 주어 이 차를 이용해 엣지맵을 구함

Log 공식

$$\mathrm{LoG}(x,\ y)=\frac{1}{\pi\sigma^4}\left[1-\frac{(x^2+y^2)}{2\sigma^2}\right]-e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Dog 공식

$$DoG(x,y)=\frac{e^{\frac{-(x^2+y^2)}{2\sigma_1^2}}}{2\pi\sigma_1^2}-\frac{e^{\frac{-(x^2+y^2)}{2\sigma_2^2}}}{2\pi\sigma_2^2}$$

5x5 Log 마스크

0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

7x7 Dog 마스크

0	0	−1	−1	−1	0	0
0	−2	−3	−3	−3	−2	0
−1	−3	5	5	5	−3	−1
−1	−3	5	16	5	−3	−1
−1	−3	5	5	5	−3	−1
0	−2	−3	−3	−3	−2	0
0	0	−1	−1	−1	0	0

log

dog

마치며

느낀점

- 프로젝트를 하며 포토샵 등의 영상 편집 프로그램이 어떤 방식으로 돌아가는지 조금 이나마 이해했습니다.

한계점

- 소스코드 단축 실패,구현 하지 못한 다수의 디지털 영상 처리 기능

향후 발전 방향

- 배운 것을 전부 응용 할 수 있게 만드는 것이 목표

감사합니다