Image/Video BM 이해
-이미지/비디오 처리-

김 승 환

swkim4610@inha.ac.kr

2. 이미지/비디오 자료처리

2.1 이미지 자료

2.2 비디오 자료

2.3 Binary & Gray Image

2.4 Image Histogram

2.5 Image Blending

2.6 Image Differencing

2.7 Chroma key

2.8 이미지 원근변환

2.9 Image Blurring

2.10 Noise Filter

2.11 Edge Detection

2.12 Sketch Camera

2.13 Image Contour

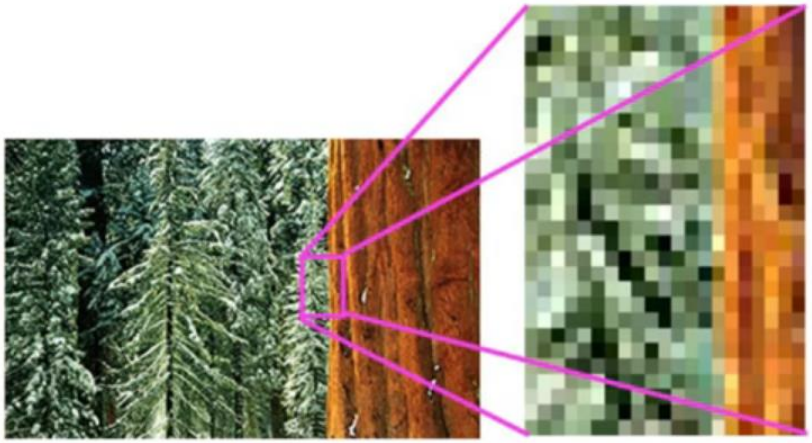
2.14 차선 검출

2.15 유사 이미지 검색

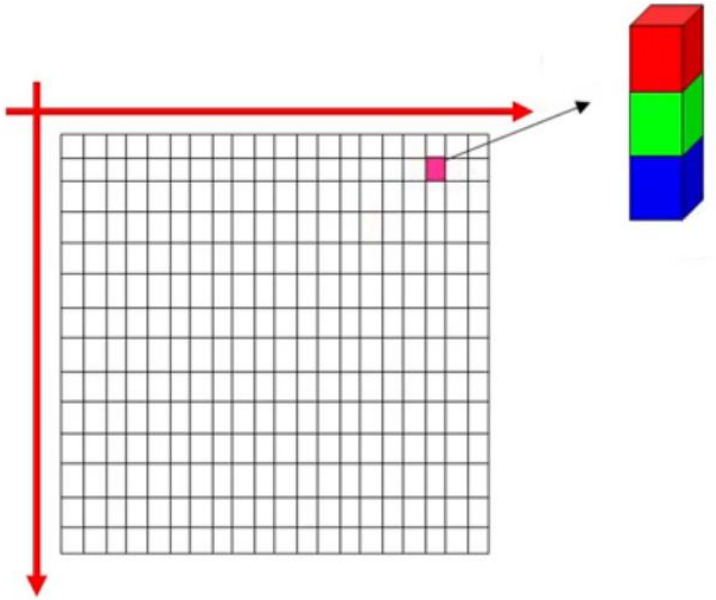
2.16 BM Ideation

2.1 이미지 자료

- 디지털 이미지는 픽셀(Pixel: Picture Element)로 구성되어 있음
- 일반적으로 픽셀은 2**k(bit) 개의 수준을 가짐(2**8 = 256 = 0~255 사이) 정수
- resolution은 이미지의 픽셀 수를 의미함
  (VGA: 640*480, FHD: 1920*1080(2k), UHD: 3840*2160(4k))



https://www.ultimate-photo-tips.com/what-is-a-pixel.html



2.1 이미지 자료

OpenCV 설치: install opencv-python

```
(640, 1920, 3)
```

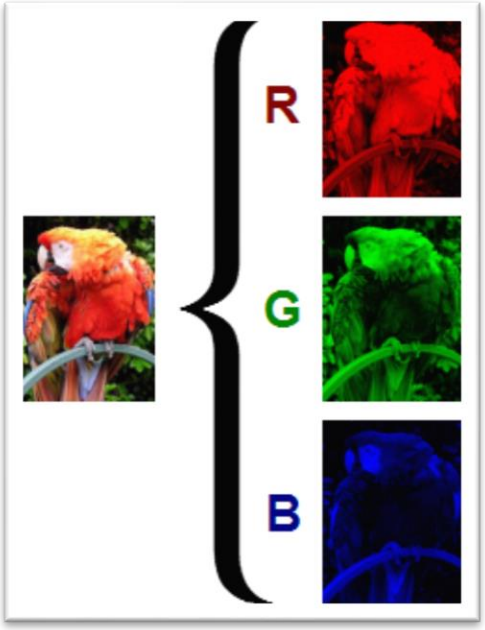
img_show.py

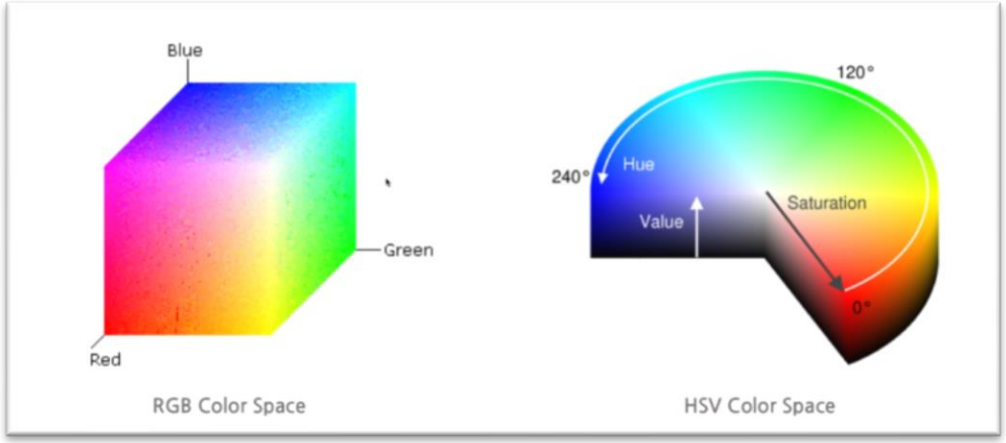
```
(640, 1920)
```

img_show_gray.py

2.1 이미지 자료

- 컬러 이미지는 RGB에서 0~255 사이 숫자로 색강도를 표현하여 픽셀당 256*256*256 = 1670만개 색 표현이 가능함
- 그레이 이미지는 이미지를 흑백으로 처리하여 0~255 사이 숫자로 0은 Black, 255는 White임
- 결국, 이미지는 2차원 혹은 3차원 행렬임
- RGB 외에 HSV(Hue, Saturation, Value) 즉, 색파장(빨강색을 0도, 보라색을 360도), 채도(진한 정도), 밝기로 나타내기도 함





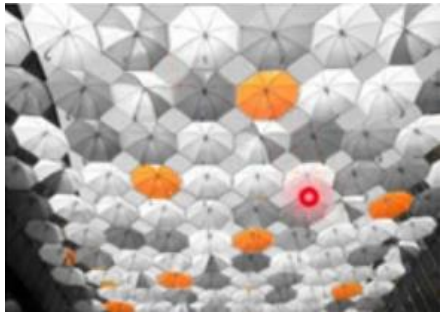
2.1 이미지 자료

- RGB는 색의 3원색이 결합되어 색깔과 강도를 나타내는데 HSV는 H는 색상, S는 채도, V는 밝기를 나타냄(채도: 선명함)
- 일반적으로 이미지 변환을 할 때, HSV로 변환 후에 처리하는 경우가 많음
- 아래 그림은 (1) 원본 (2) V를 낮춤 (3) 특정 색 이외의 S 값을 0으로 바꿈 (4) H 값에 임의의 수를 가감한 결과임

(1)

(2)

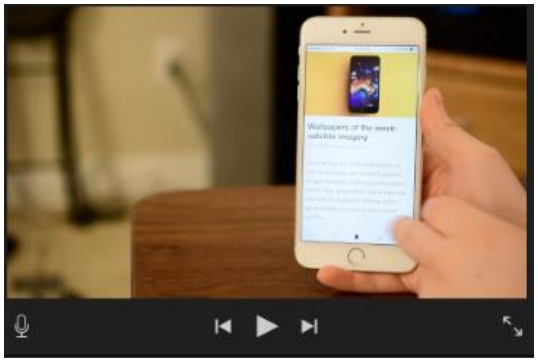
(3)



(4)

2.2 비디오 자료

- 비디오는 이미지를 시간순서에 따라 배열한 Sequence 집합임
- 비디오의 품질은 이미지 해상도와 Frame rate fps 에 의해 좌우됨
- fps(프레임/초)는 초당 이미지의 수를 말함. 많을 수록 부드럽지만, 데이터 량이 커짐
- 지연시간은 1000 / fps 로 계산되며 이는 비디오 파일의 재생속도를 조절하는 용도로 사용함
  예를 들어, 1초(1000ms) 에 20 fps 만큼 재생하려면 이미지와 이미지 사이에 50ms 만큼 지연하여야 함
- 비디오 자료에서 fps를 적당히 조정해야 계산량도 줄이고 적절한 품질을 만들 수 있음
- 비디오 자료 처리에 이미지 샘플링을 해서 처리하는 것이 일반적임
- 파일 대신 캠코더 입력을 실시간으로 처리하는 것도 가능함 videoCam.py videoRecord.py



2.3 Binary & Gray Image

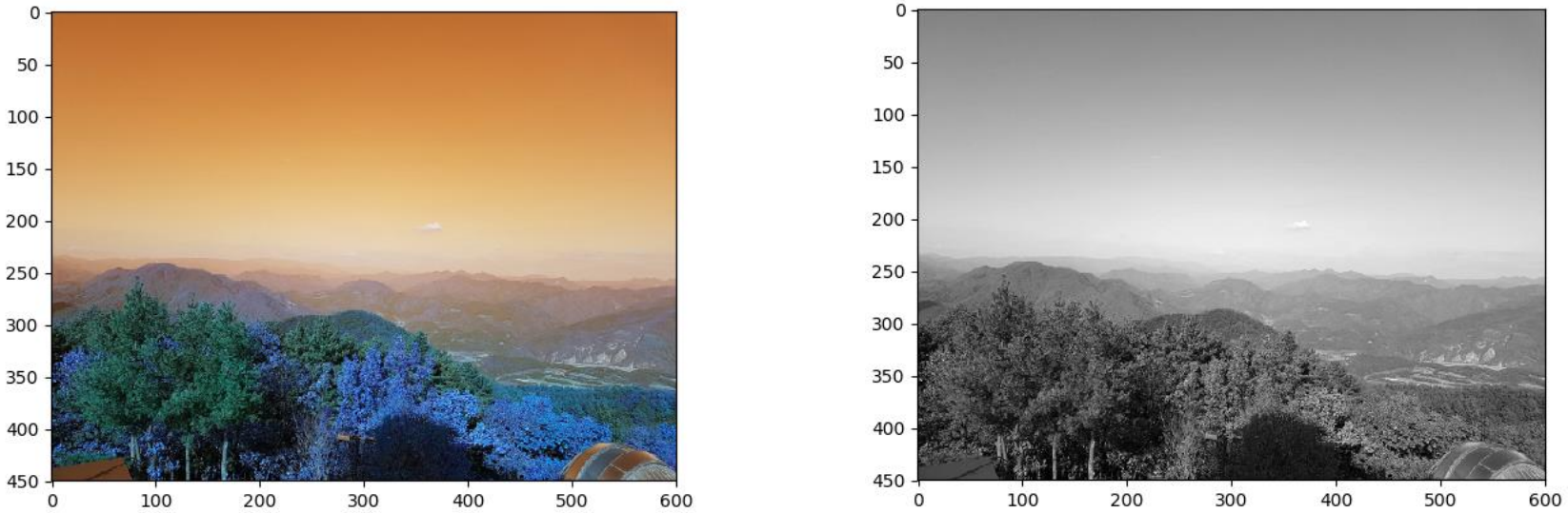
- 흑백 이미지는 0과 255만 있는 이미지로 0~255사이의 숫자에서 특정 Threshold 값을 기준으로 0 혹은 255로 맵핑하여 얻을 수 있음
- 아래의 이미지는 이보다 복잡한 알고리즘(기우시안 필터)를 이용해 얻은 흑백사진임
- 이 방법은 Threshold을 픽셀 인접값의 가중치 평균으로 결정하는 방법임

adativeThreshold.py

2.3 Binary & Gray Image

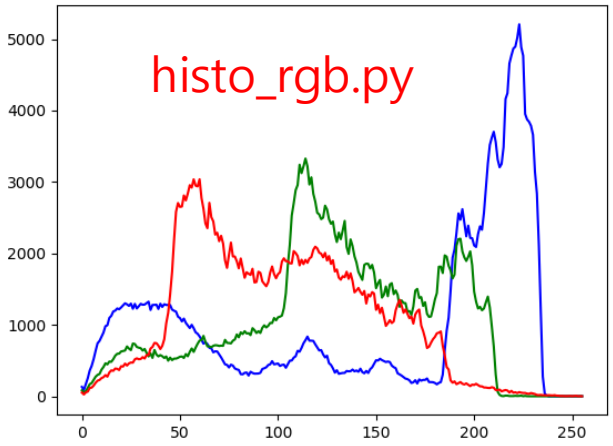
- 0.21 R + 0.72 G + 0.07 B의 계산으로 그레이 이미지를 얻을 수 있음

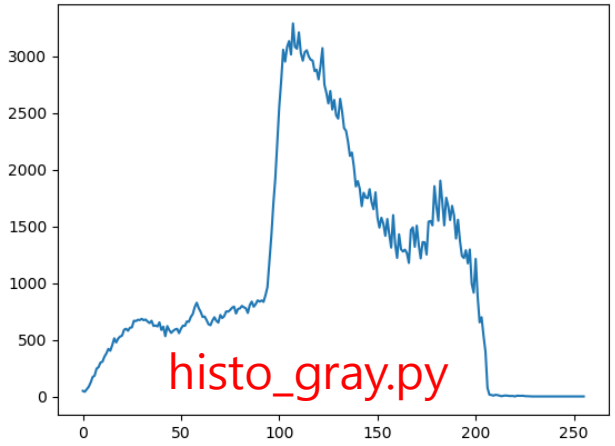
GrayImage.py



2.4 Image Histogram

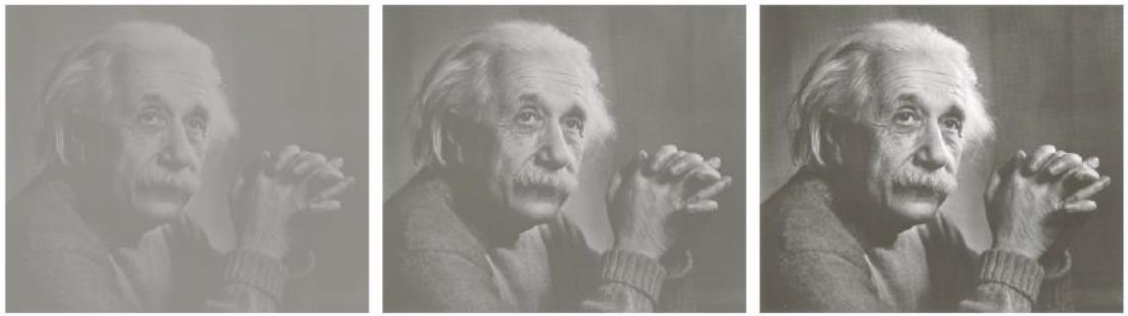
- 이미지는 숫자이므로 숫자 히스토그램을 그려 색의 분포를 표현할 수 있음
- 컬러사진은 R, G, B 채널에 대해 각각 그릴 수 있고, 그레이 사진은 1차원으로 표현 가능함
- 히스토그램을 통해 특정색의 값과 분포높이를 파악할 수 있음



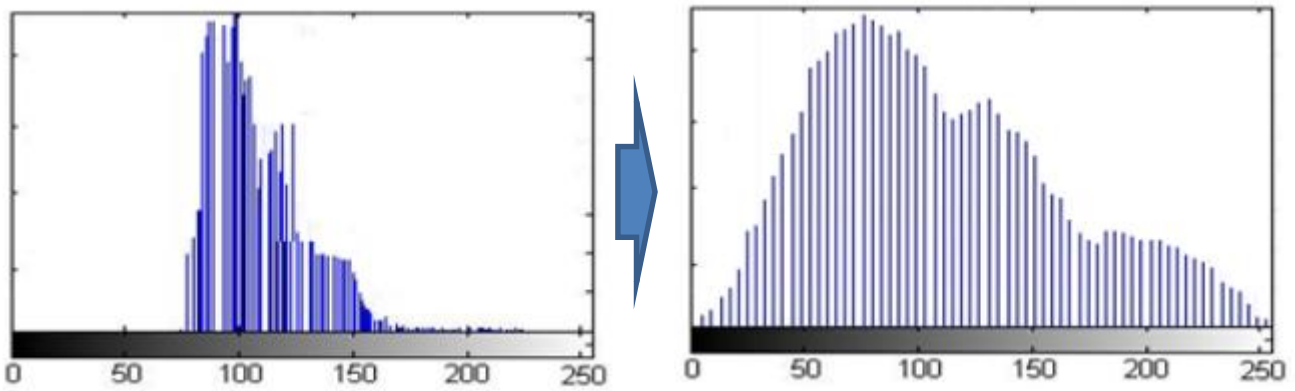


2.4 Image Histogram

- 이미지 히스토그램이 한쪽으로 치우쳐 있는 경우, 정규화를 할 수 있음
- 정규화는 채널별 최대, 최소값을 구해 0과 255 사이로 재계산하는 것임
- 평활화는 히스토그램의 분포의 표준편차를 크게 하여 최대한 색의 차이가 벌어져 Contrast가 커지도록 하는 것임



$\sigma = 14.3$ $\sigma = 31.6$ $\sigma = 49.2$

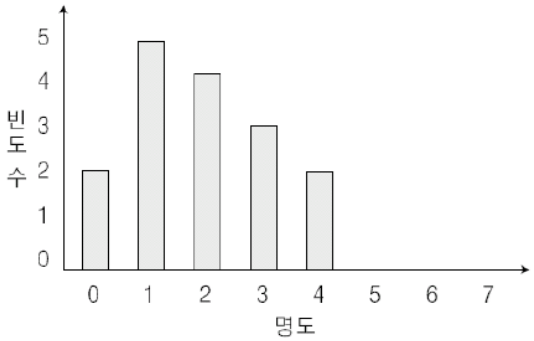


2.4 Image Histogram

- Histogram Stretching은 $x_{ij} = \frac{x_{ij} - X_{min}}{X_{max} - X_{min}} * 255$, 강제로 (0, 255)로 늘려주는 것임
- Histogram Equalization은 C.D.F.(누적분포)의 개념을 이용한 방법임

2	4	4	3
2	1	3	3
1	0	1	2
0	1	1	2

(a) 입력 영상

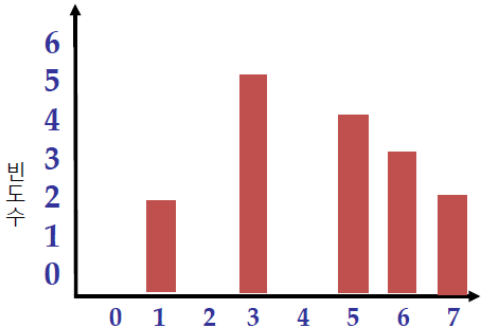


(b) 히스토그램

pixel값	누적빈도	CDF	CDF*max(pixel)	rounding
0	2	0.125	0.875	1
1	7	0.4375	3.0625	3
2	11	0.6875	4.8125	5
3	14	0.875	6.125	6
4	16	1	7	7
5	16	1	7	7
6	16	1	7	7
7	16	1	7	7

5	7	7	6
5	3	6	6
3	1	3	5
1	3	3	5

(a) 결과 영상



(b) 평활화 된 히스토그램

2.4 Image Histogram

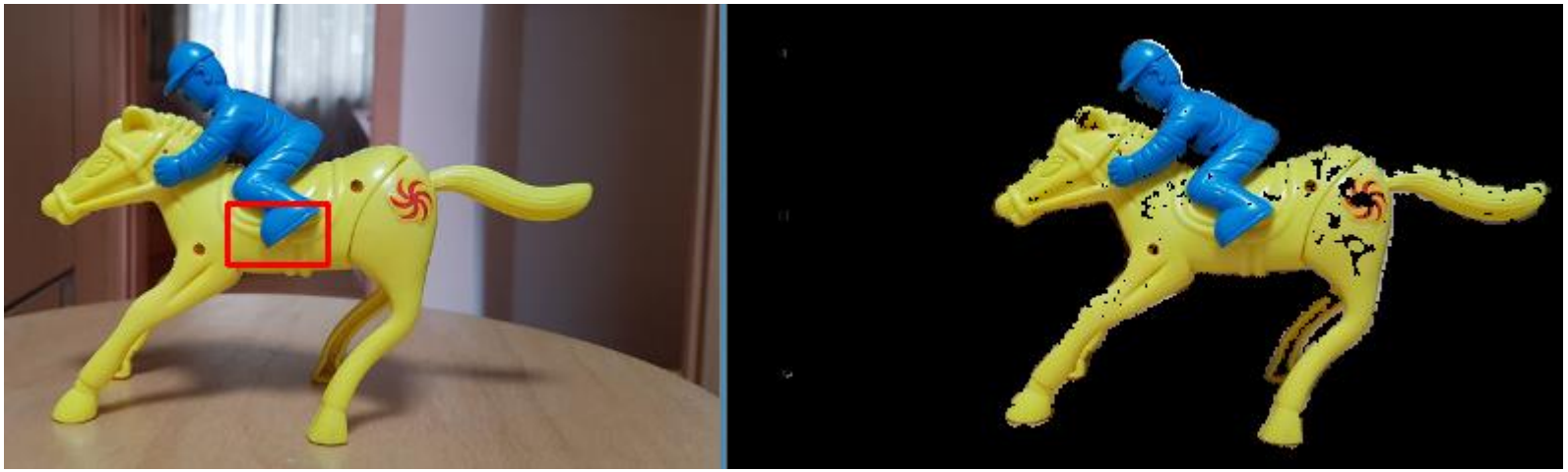
- 좌측이 원본 이미지이고, 우측이 평활화된 이미지임

histo_normalize.py

2.4 Image Histogram

- 이미지 RGB 값을 HSV로 변환 후, H는 색을 의미하므로 이를 하나의 히스토그램으로 표현 가능함
- 히스토그램 높이는 색에 대한 빈도를 나타내는 값이고, 빈도의 합이 1이 되도록 변환하면 밀도가 됨
- 즉, 색깔별로 0~1사이의 밀도 값을 얻을 수 있으므로 아래 그림에서 노랑/파랑은 밀도값이 크고 나머지 색은 거의 밀도가 0이 될 것임
- 역 투영을 통해 원하는 부분만 Crop/Masking이 가능함
- 아래는 좌측 이미지에서 노랑/파랑색 부분을 ROI로 지정하고, ROI 이미지의 히스토그램을 전체 이미지에 역 투영하여 얻은 결과임
- ROI에 없는 색은 빈도가 없으므로 검정색으로 변함

histo_backproject.py



2.5 Image Blending

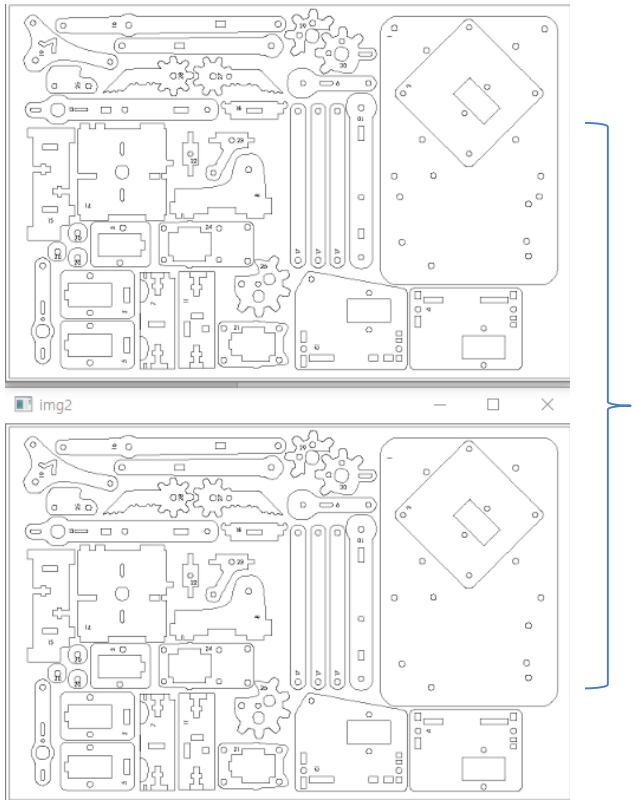
- 술의 브랜딩과 유사하게 이미지도 브랜딩이 가능함

가중평균

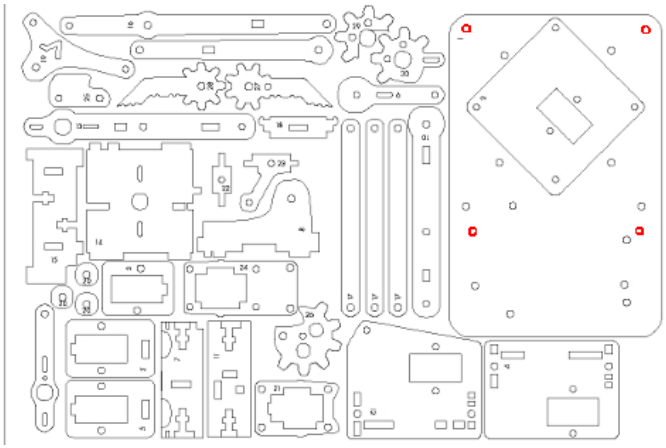
blending_alpha_trackbar.py

2.6 Image Differencing

- 두 이미지의 차이는 틀린 그림 찾기 임(어렵다)
- 예를 들어, 두 도면의 차이를 찾거나, 전자제품 PCB의 결함을 찾는데도 응용 가능함
  혹은 블랙박스 주차모드에서 촬영시작 여부를 인지할 수 있음



diff_absolute.py



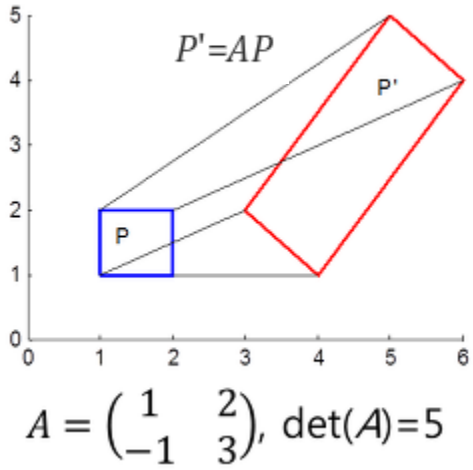
2.7 Chroma Key

- 아래는 두 이미지에서 전경과 배경을 섞는 예제임
- 영화에서 많이 사용함
- 인물사진에서 초록색 배경을 제거하고 나머지 부분의 값만 추출한 후, 배경사진에 더해 주면 됨

chromakey.py

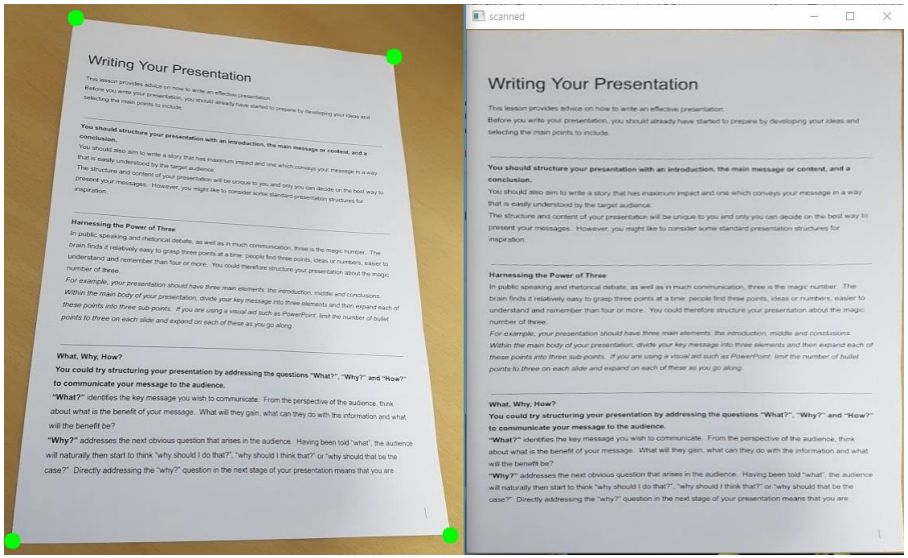
2.8 Image 원근변환

- 이미지는 촬영하는 각도에 다를 수 있음
- 요즘, 휴대폰 카메라 성능이 좋아져 카메라로 문서 스캔을 대신하는 경우가 많음
- 가까운 것은 크게 보이고, 먼 곳은 작게 보이는 문제를 해결하는 것이 원근변환임
- 이러한 변환은 이미지 좌표계 행렬에 대한 Y = AX 와 같은 선형 변환을 통해 새로운 좌표를 만들어 내는 원리임



$$A = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \det(A)=5$$

perspective_scan.py



2.9 Image Blurring

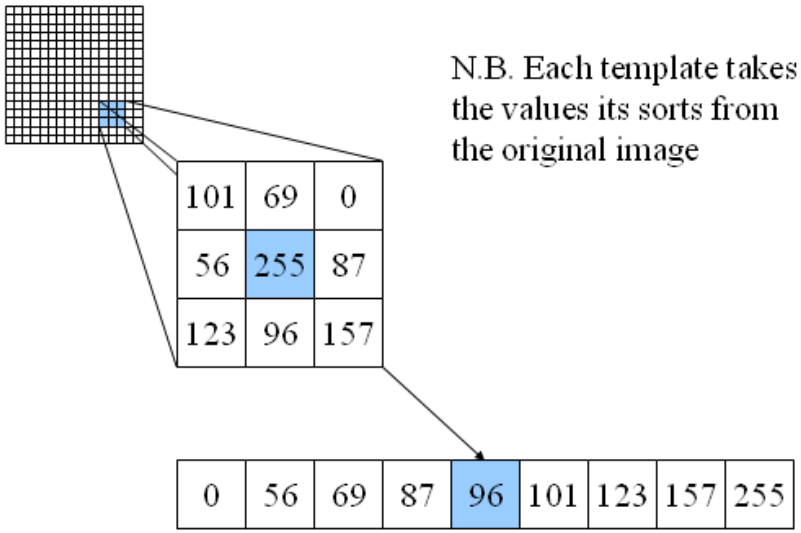
- 흔히, 말하는 뽀샵임
- 블러링은 우측과 같은 필터를 통해 구현함

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

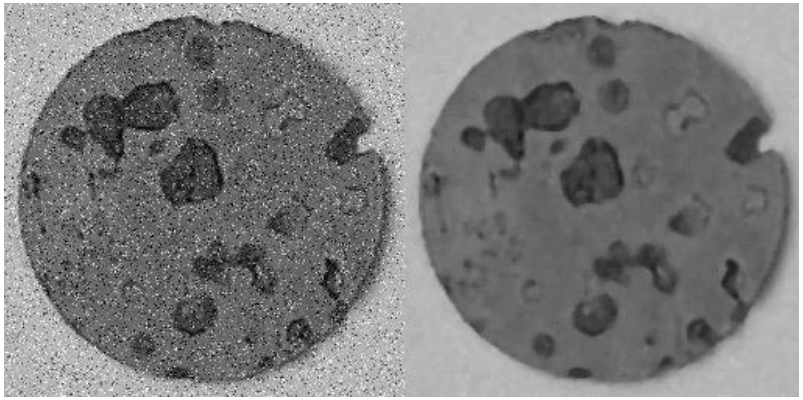
blur_gaussian.py

2.10 Noise Filter

- 가우시안 필터는 가중평균 개념이라면 median filter를 사용하여 노이즈를 제거할 수 있음
- salt and pepper noise(0, 255 잡음)를 탁월하게 제거함
- Average Filter에 비해 연산량이 많음

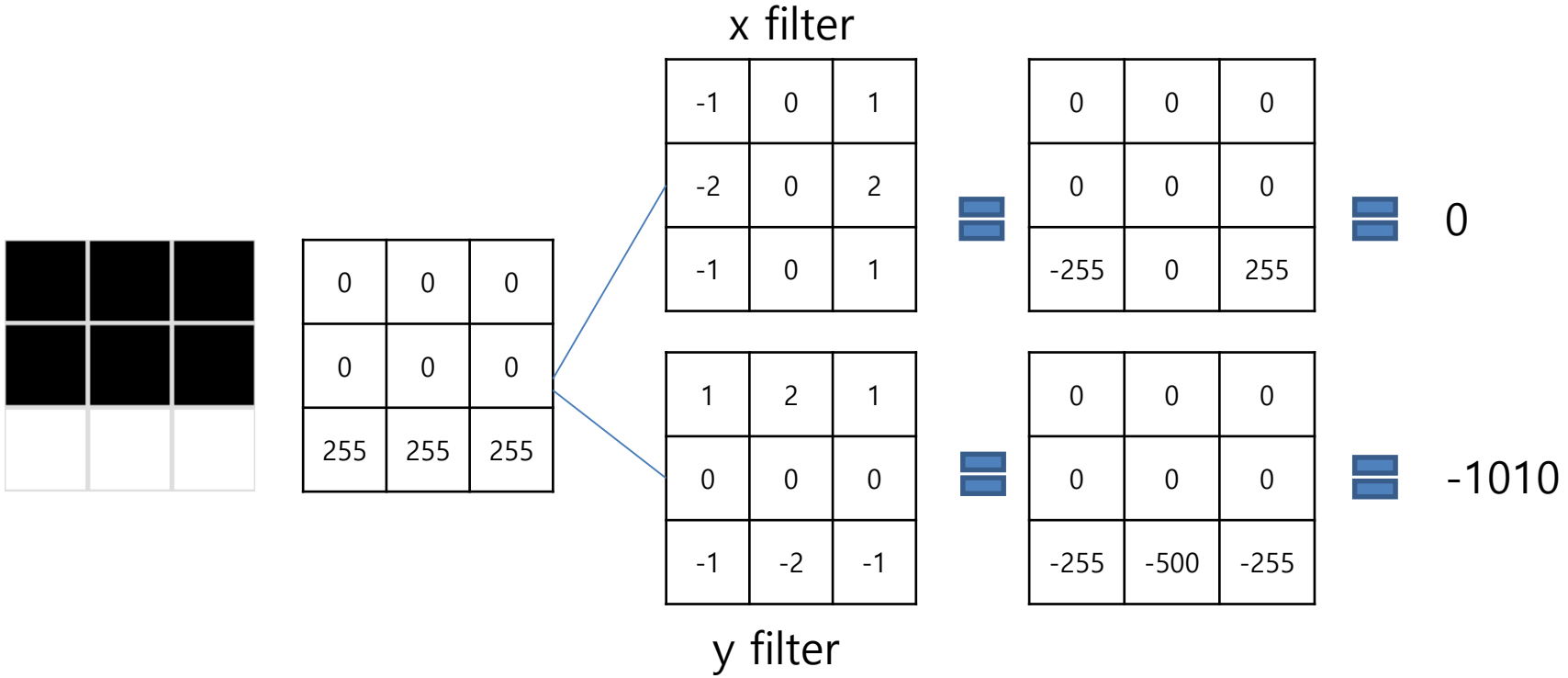


blur_median.py



2.11 edge detection

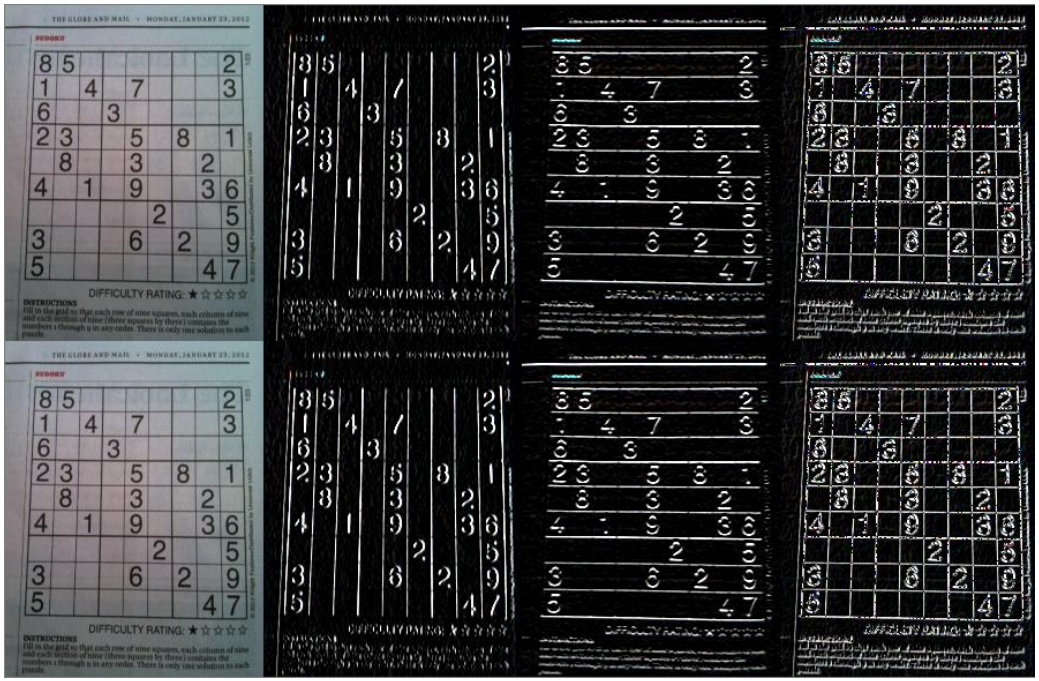
- 이미지에서 영역 경계를 찾는 것은 전경과 배경을 구분하는데 많이 사용하는 기법임
- 기본적인 아이디어는 값의 변화가 큰 곳(2차 미분 값이 큰 곳)을 찾아 강조 시키는 것임
- 실무적으로 1968년 Sobel이 제안한 Sobel Filter를 많이 사용함
- 아래의 그림은 x 축은 변화가 없고, y축으로 변화가 있음
- Sobel 필터 적용결과, x필터 값은 0이고, y 필터 값은 크게 나옴(절대값 기준)



2.11 edge detection

- Sobel 필터를 이용하여 이미지 영역의 경계를 찾는 예임

edge_sobel.py



2.12 sketch camera

- 아래는 컬러 이미지를 그레이로 바꾸고 엣지를 얻은 다음, 엣지에 대해 팽창연산을 해서 얻은 결과임
- 이러한 기술을 활용한 스케치 어플이 있음

workshop_painting_cam.py

2.13 morphology

- 모폴로지는 형태학이란 뜻으로 노이즈 제거, 경계 이어주기 등 이미지의 결점을 보완해주는 변환임
- 침식(255→0), 팽창(0→255) 변환이 기본적인 연산이고 이를 조합한 Open(부분침식) $A^{\circ}B$, Close(부분팽창) $A \cdot B$ 연산을 사용함

Erosion returns 1 if all nine pixels are 1 and 0 otherwise.

Dilation returns 0 if all nine pixels are 0 and 1 otherwise.

$A \ominus B$

original eroded once eroded twice

$A \oplus B$

original dilated once dilated twice

$A \circ B = (A \ominus B) \oplus B$

original epened

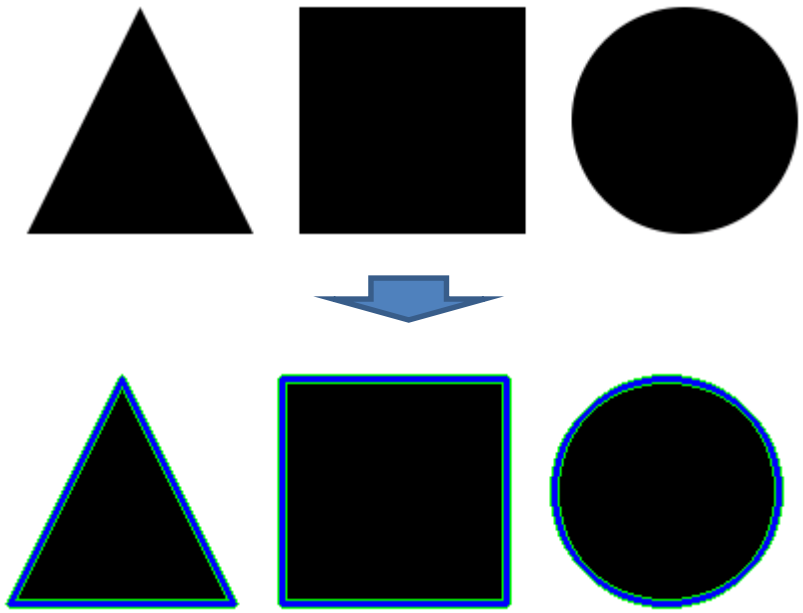
$A \bullet B = (A \oplus B) \ominus B$

original closed

morph_open_close.py

2.14 Image Contour

- 등고선은 지도에서 높이가 같은 곳을 선으로 이어주는 그림임
- 이미지에서 객체는 같은 색으로 이어지는 경향이 있으므로 같은 색으로 이어지는 등고선으로 궤적을 찾을 수 있음



cntr_find.py

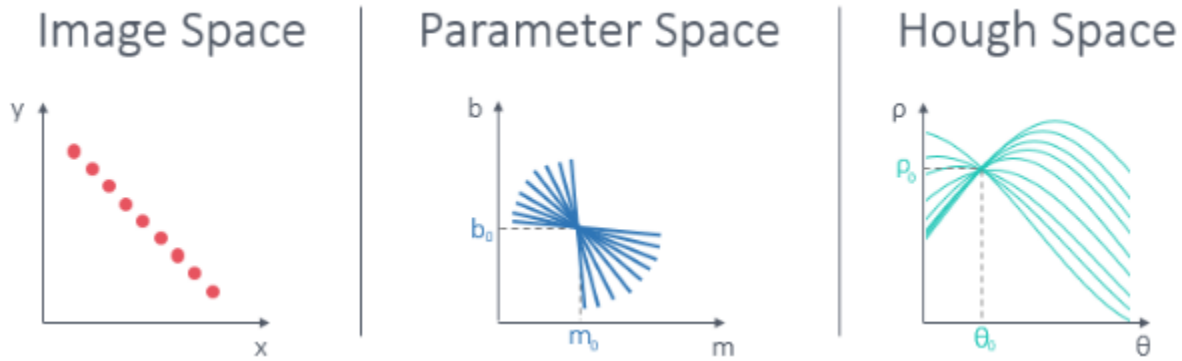
2.15 차선 검출

- Hough Transform은 이미지에서 사각형이나 원을 찾는데 사용함
- 자율주행 자동차가 차선을 인지하는 것은 이미지에서 직선을 검출하는 것임
- 같은 방식으로 이미지 상에 원 형태를 검출하는 것도 가능함
- 알고리즘은 이미지 상에서 직선상에 존재하는 픽셀좌표들이 그 직선의 절편과 기울기로 이루어진 평면에서는 각 점들이 각각 직선으로 표현되고 이 직선이 모두 한점을 통과하는 원리를 이용함

laneDetect.py

첨부: Hough Transform

- 평면 이미지에서 (x_1, y_1) 픽셀을 지나는 직선은 $y_1 = mx_1 + b$가 성립함
  여기서, (m, b)는 무수히 많음. 하지만, $b = y_1 - mx_1$ 조건 때문에 (m,b)평면에서 직선 상 존재
- (m,b) 피라미터 공간의 직선 식은 $b = y_1 - mx_1$임
- 평면 이미지에서 (x_2, y_2) 픽셀을 지나는 직선의 기울기와 절편(m, b)이 $y_1 = mx_1 + b$의 (m,b) 인 (m_0, b_0)와 같다고 하면 $b = y_2 - mx_2$ 이고 이 직선은(m_0, b_0)를 통과함
- 마찬가지로 $b = y_1 - mx_1$ 도(m_0, b_0)를 통과함
- 같은 원리로 이미지 공간에서 직선 상에 존재하는 픽셀은 모두 파라미터 공간에서 (m_0, b_0)를 통과함
- 이미지 각 픽셀에 대해 파라미터 공간의 직선을 만들고 이 직선들이 만나는 점의 숫자를 count하여 특정 점에서 많이 만나면 그 절편과 기울기가 이미지 공간에서 직선임
- 다만, Y 축에 평행한 직선은 기울기가 무한대이므로 파라미터 공간에 표현이 불가능함
- Hough 공간은 (m_0, b_0)을 극좌표 변환을 통해 (θ_0, ρ_0)로 변환한 것임. θ는 0과 2π 사이값임



2.16 유사 이미지 검색

- 가장 간단한 아이디어는 Hash Matching 방법임
- 이 방법은 이미지를 Gray로 변환한 후, 일정한 크기로 변환함(예: 16*16)
- 변환된 값 > 이미지 픽셀 평균값 이면 1, 아니면 0을 가지는 이진화된 이미지를 만든다.
- 이 이미지 Sequence는 모든 이미지에 대해 같은 크기의 이진수를 가진다.
- 앞에서 구한 이진수를 16진수로 변환하여 얻은 코드를 서로 비교하여 비슷한 이미지를 찾는다.
- 코드의 유사도를 비교하기 위해 Hamming Distance를 많이 사용함
- 아래는 좌측 권총 이미지를 101_objectCategories에서 찾은 결과를 보여줌
- http://www.vision.caltech.edu/Image_Datasets/Caltech101/#Download

avg_hash_matching.py

감사합니다