영상처리_1장 요약본

1. 영상처리 용어 정리

영상처리(Image processing)

: 전자적으로 영상을 얻은 후, 컴퓨터로 영상을 처리해 원하는 출력영상을 얻는 기술분야

향상	시각적 개선, 대비증가, 컬러/흑백변환, 경계탐지
복원	훼손된 영상회복, 잡음제거, 신호처리
압축	영상의 질 보장 데이터 비트 축소, 정지영상/동영상

영상분석(Image Analysis)

: 고수준의 정보를 얻기위해 영상을 다루는 알고리즘

영역 분할	객체에 속한 화소 구분 과정, 상향식 그룹화
분류	이전에 만들어진 모델에 화소 결정, 하향식 비교
모양 복원	정면에서 3차원 구조 복원, x는(비디오, 스테레오)

※ 영상처리/ 영상분석 비교 ※

	영상처리	영상분석
입출력	영상 -> 영상	영상 -> 정보
알고리즘	영상변환, 선형/비선형 필터링, 주파수 영역 처리	선형대수, 통계분석, 투상기하, 함수 최적화
주요문제	향상, 복원, 압축	영역분할, 분류, 모양복원

※ 머신비전/ 컴퓨터비전 비교 ※

컴퓨터가 생기기 전 다른 기계가 대신함 -> 머신비전

	환경	센서	알고리즘	출력
영상처리	any	any	low-level(2D)	이미지
영상분석	any	any	low-to high-level	정보
머신비전	산업	카메라	low-level(2D)	정보
컴퓨터비전	매일	카메라	mid-to high-level	정보

※ 카메라를 제외한 센서의 종류 - 적외선, 소리

머신비전(Machine vision)

: 컨베이어벨트 상의 상품을 카메라로 불량 검출

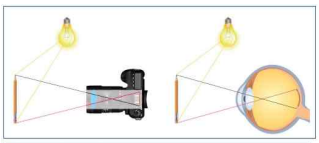
컴퓨터비전(Computer vision)

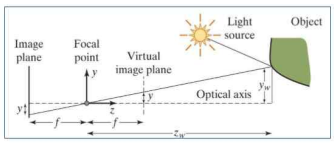
: 휴대폰, 이동로봇, 자동차 등의 카메라에서 처리

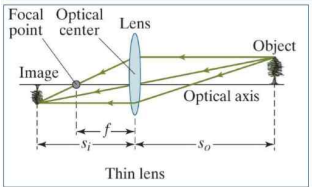
2. 영상 생성 원리

① 핀홀 카메라(pinhole camera)

: 사람의 눈과 비슷한 원리







$$\frac{1}{f} = \frac{1}{s_o} + \frac{1}{s_i}$$

f : 초점거리

s_0 : 망막에 맺히는 거리

s_1 : 이미지에 맺히는 거리

① CCD& CMOS 센서 - 반도체

CCD (Charge Coupled Device)

CMOS (Complimentary Metal Oxide Semiconductor)

※ 비교 ※ CMOS가 성능이 더 좋다!!

	CCD	CMOS
센서 출력 신호	아날로그 전기신호	디지털 전기신호
시스템 크기	큼	작음
시스템 구조	복잡	간단
반응속도	약간 떨어짐	조금 우수함
어두운 곳에서의 균일성	높음	조금 떨어짐
작동 속도	우수함	매우 우수함 (연구/의료용 사용)
전력 소비	큼	작음
시스템 내구성	오랜 기간 개선을 통해서 좋아짐	매우 우수함

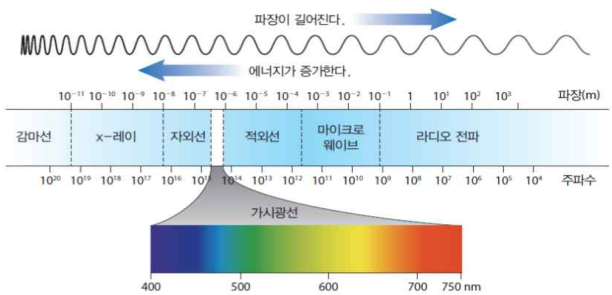
참고) 빛 광자를 전하량으로 바꾸는 것 - 광다이오드

③ 영상 생성 원리

- 전자기 에너지

파장 길어짐

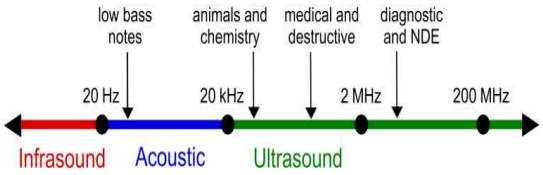
방해물 따라 잘 감



수식 : $E = hf = \frac{h \cdot c}{\lambda}$ 파장과 진동수 역수관계!!

f : 진동수 λ : 파장
카메라는 가시광선만 찍을 수 있음
영상처리는 모든 종류를 볼 수 있어야 함

- 음향



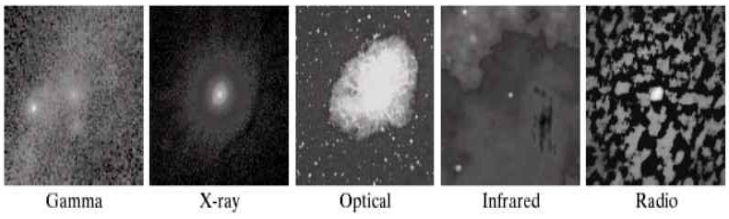
초음파 : 동물, 의료 영역(태아 진단)
음파 : 가청 대역
초저주파 : 지진/화산 등 자연 모니터링, 의료 영역
(심장 역학)

- 컴퓨터에 의한 생성
프랙탈(Fractal): 유사성 규칙, 자연 생성 원리 해석
3차원 모델링, 렌더링
해상도 / 공간 해상도 (DPI : dots per inch)

④ 전자기 스펙트럼 영상

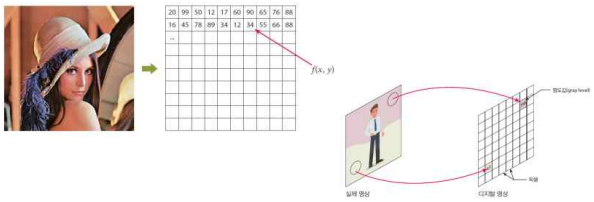
감마선 (Gamma-rays)	감염/종양 진단, 천체 관찰
엑스레이(X-rays)	물체 투영 관찰, 용도별 세기 조절
자외선 (Ultra-Violet)	농작물 상태 검출, 천체 관찰
적외선(Infrared)	열 효과 측정, 근육 치료 및 멸균
가시광선 (Visible light)	약 400~700nm 범위 가시광선의 영상: 컬러영상 - R,G,B 스펙트럼(맥스웰) - 광다이오드 : CCD, CMOS 활용
마이크로파 (Microwave)	약 300Ghz ~ 300Mhz 안테나 통신, 레이다, 전자레인지
라디오파 (Radiowave)	자기공명영상 : 조직에 따라 방사된 수소 원자 반응 펄스를 영상화 (MRI:Magnetic Resonance Imaging)

1. 다중 스펙트럼 영상 활용
2. 같은 물체에 대해서도 다른 영상을 얻음
3. 주어진 영상 정보뿐만 아니라 추가적 정보를 모두 활용해 최적의 정보를 얻음



3. 디지털 영상

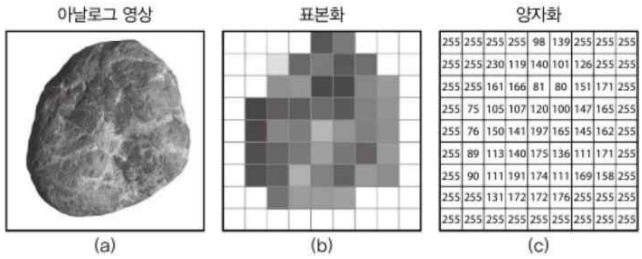
① 디지털 영상
: 수학적으로는 2차원 함수로 정의
함수 값 = 그 점에서의 영상 밝기



② 표본화(샘플링, Sampling)
: 공간 영역에서 픽셀의 개수를 제한하는 것

③ 해상도/공간 해상도(Spatial Resolution)
: 표본화에 의한 식별 가능한 상세도
(단위 거리 당) 선/점(픽셀,화소)수, 인치 당 점 수(dpi)
dpi - dots per inch
샘플링 많이 할수록 고해상도

④ 양자화(Quantization)
: 화소의 값을 정해진 몇 단계의 밝기로 제한하는 과정
가로축 : 표본화 세로축 : 양자화



⑤ 밝기 해상도(Intensity Resolution)
: 밝기 레벨 : 이산적 레벨 [0, L-1], $L = 2^k$
k비트 영상: $L = 2^k$ 레벨 8비트 영상 -> $2^8 = 256$
예) $N \times N$ 해상도, L 레벨 비트 용량


[그림 1.11] 다양한 양자화에 따른 영상의 변화(256, 129, 64, 32, 16, 8, 4, 2)

4. 영상의 표현

※ 영상 형태 ※

① binary Image
- 흑백 영상 0과1만 표현

N/k	1 (L = 2)	2 (L = 4)	3 (L = 8)	4 (L = 16)	5 (L = 32)	6 (L = 64)	7 (L = 128)	8 (L = 256)
32	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
64	4,096	8,192	12,288	16,384	20,480	24,576	28,672	32,768
128	16,384	32,768	49,152	65,536	81,920	98,304	114,688	131,072
256	65,536	131,072	196,608	262,144	327,680	393,216	458,752	524,288
512	262,144	524,288	786,432	1,048,576	1,310,720	1,572,864	1,835,008	2,097,152
1024	1,048,576	2,097,152	3,145,728	4,194,304	5,242,880	6,291,456	7,340,032	8,388,608
2048	4,194,304	8,388,608	12,582,912	16,777,216	20,971,520	25,165,824	29,369,128	33,554,432
4096	16,777,216	33,554,432	50,331,648	67,108,864	83,886,080	100,663,296	117,440,512	134,217,728
8192	67,108,864	134,217,728	201,326,592	268,435,456	335,544,320	402,653,184	469,762,048	536,870,912

- 2^1개, 1bit

② grayscale Image
- 밝기(명암도)에 따라 0~255로 표현
- 2^8개, 8bit(1byte)

③ color Image BGR 순서로 들어가있다.
- Red, Green, Blue로 표현
- $2^8 \cdot 2^8 \cdot 2^8$개, 24bit(3byte)

※ 영상 데이터 표현과 접근 ※
2차원 배열을 1차원 배열로 메모리 저장
- 행 주요 순서(row major order)

인덱스 주소 구하기
$i = y \cdot width + x$
좌표 구하기
$x좌표 = i - y \cdot width = \mathrm{mod}(i, width)$
$y좌표 = i/width$

5. 영상처리 응용
① 운송, 자율 주행 자동차
- 화소 변화 측정, 배경과 전경의 분리 변화 측정

예) 교통량 및 도로 점유 정도 측정 신호 제어, 공항 활주로 낙하물 탐지, 자율 주행의 차선 이탈 및 보행자 감지, 자율 주차

② 공장 자동화
- 산업 검사(Industrial inspection) : 머신 비전 시스템, 제품 결함 찾기

예) 반도체 웨이퍼 결함, 약봉지 알약여부, 기계 부품 조사, 이물질 검사

③ 문서 자동 인식

예) 우체국 편지 문자 주소 분석 분류, 자동차 번호판 인식, 가짜 지폐 구분

④ 과학/의료 이미징
- 과학적 이미징(scientific imaging)
  ㉠ 과학 분야의 현상 연구 측정
  예) 세포 수준 현미경 영상, 뿌리 성장 측정, 천문 이미징 기술
- 의료 이미징(medical imaging)
  ㉠ 생체인식(지문/얼굴) 시스템
  ㉡ 의료 분야 연구 활동
  예) 종양 감지, 골절, 뇌 신경 활동, 3차원 신체 모델, 혈관 이미징

⑤ 원격 감시
- 여러 스펙트럼 대역의 데이터 획득 및 처리

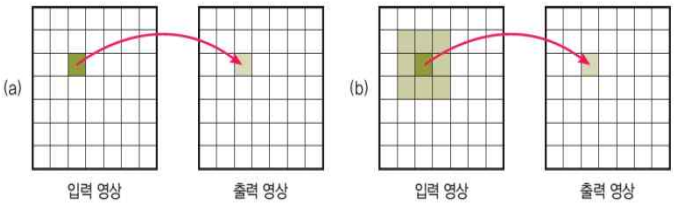
예) 식물 양 측정, 광산 위치, 수질 온도, 해수면 변화, 무인 감시

화소처리_3장 요약본

1. 화소 처리의 이해

① 전통적인 영상처리 ※ 비교 ※

화소 처리	위치는 같은데 화소(pixel)만 바뀜
공간 필터링	주변값을 통해 위치 생성

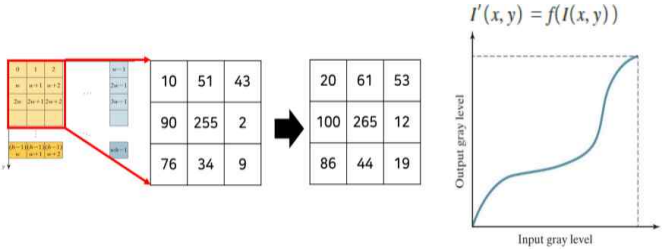


[그림 4.1] 화소 처리와 공간 필터링의 비교 (a) 화소 처리 (b) 공간 필터링

② 화소 변환 (point transformation)

- 화소의 위치는 변경하지 않고, 화소의 값을 변경

$8bit$: $f : Z_{0:255} \rightarrow Z_{0:255}$ (Z는 양수)



2. 밝기 및 콘트라스트 처리

※ 1. 화소를 하나씩 처리하는 방법 ※

㉠ at(y,x) y: 행, x: 열

- 영상에서 임의의 화소값을 가져오거나 수정 가능
- at(y,x)의 인수로 화소의 행, 열 번호 전달하면 됨

```
Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
imshow("Original Image", img);

for (int r = 0; r < img.rows; r++)
    for (int c = 0; c < img.cols; ++c)
        img.at<uchar>(r, c) = img.at<uchar>(r, c) + 30;

imshow("New Image", img);
waitKey(0);
return 0;
```

<오류 원인>

화소 값에 30이 더해지면서 0~255까지 표현할 수 있는 범위를 넘어서서 오버플로우 발생

img.at<uchar>(r,c)= saturate_cast<uchar>(img.at<uchar>(r,c)+30);

포화도 변환 ↳ 255가 넘어갈 경우 포화도 만큼으로 셋팅해라

* underflow와 overflow 현상 발생

㉡ 덧셈 연산

$$I'(x,y) = I(x,y) + b$$

- 오버/언더플로 가능성 클램프가 더 넓은 의미

클램프	해당 비트 표현에 가장 가까운 유효 값 설정
포화 산술	$I'(x,y) = \min(\max(I(x,y)+b, 0), 255)$

```
int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    for (int r = 0; r < img.rows; r++) {
        uchar *p = img.ptr<uchar>(r);
        for (int c = 0; c < img.cols; ++c) {
            p[c] = saturate_cast<uchar>(p[c] + 30);
        }
    }
    imshow("New Image", img);
    waitKey(0);

    return 0;
}
```

pointer / 행 시작주소 / 화소처리 / p[c] = *(p+c)

㉢ convertTo()

- 영상의 밝기를 증가시키는 것

```
int main()
{
    Mat img = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img);

    Mat oimage;
    img.convertTo(oimage, -1, 1, 30);

    imshow("New Image", oimage);
    waitKey(0);
    return 0;
}
```

output / 30을 더해라

```
void convertTo(OutputArray m, int rtype, double alpha = 1, double beta = 0)
```

초기값

convertTo (oimage , -1) ⇒ 복사

매개 변수	설명
m	출력 행렬. 작업 전에 적절한 크기나 유형이 없으면 다시 할당된다.
rtype	원하는 출력 행렬 유형을 지정한다. rtype이 음수면 출력 행렬은 입력과 동일한 유형을 갖는다.
alpha	화소값에 곱해지는 수
beta	화소값에 더해지는 수

2. 콘트라스트(Contrast)

Gray 레벨 이미지 기준: 이미지 상의 객체들을 얼마나 잘 구분해 낼 수 있는 정도

높은 콘트라스트: 많은 구분 가능한 세기값(intensity)

낮은 콘트라스트: 적은 세기값의 사용

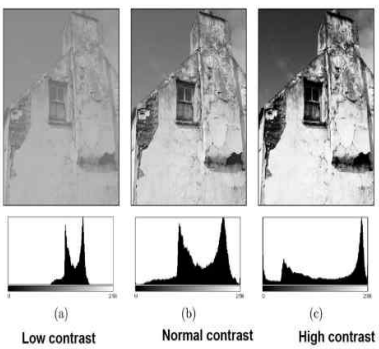
Good Contrast

- 넓게 분포되어 있는 세기값 +
- 작은 값과 큰값의 큰 차이

Contrast Equation: 다양한 종류 존재

- 예) Michalson's equation

$$C(I) = \frac{\max(I) - \min(I)}{\max(I) + \min(I)}$$



산술 연산 : 곱셈과 덧셈

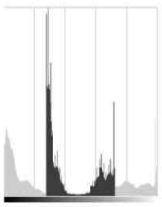
$I'(x,y) = c \cdot I(x,y) + b$: c 게인(gain), b 바이어스(bias)

- 비교: TV/모니터에서 대비/ 블랙레벨 콘트롤

$g(x,y) = \alpha \cdot f(x,y) + \beta$

α : 밝기값이 어떻게 퍼지는지를 결정한다

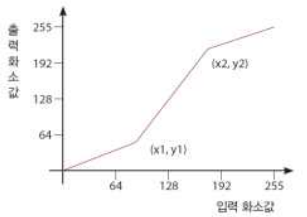
β : 화소에 더해지는 값



[그림 4.3] 밝은 회색은 원본 영상의 히스토그램, 어두

```
int main()
{
    double alpha = 1.0;
    int beta = 0;
    Mat image = imread("d:/contrast.jpg");
    Mat oimage = Mat::zeros(image.size(), image.type());
    cout << "알파값을 입력하시오: [1.0-3.0]: "; cin >> alpha;
    cout << "베타값을 입력하시오: [0-100]: ";   cin >> beta;
    for (int y = 0; y < image.rows; y++) {
        for (int x = 0; x < image.cols; x++) {
            for (int c = 0; c < 3; c++) {
                oimage.at<Vec3b>(y, x)[c] =
                    saturate_cast<uchar>(alpha*(image.at<Vec3b>(y, x)[c]) + beta);
            }
```

선형 콘트라스트 확대

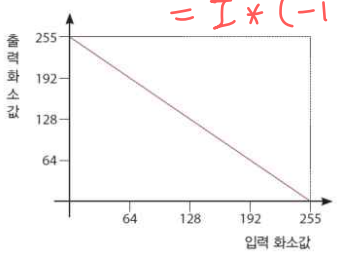


```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int contrastEnh(int input, int x1, int y1, int x2, int y2)
{
    double output;
    if (0 <= input && input <= x1) {
        output = y1 / x1 * input;
    }
    else if (x1 < input && input <= x2) {
        output = ((y2 - y1) / (x2 - x1)) * (input - x1) + y1;
    }
    else if (x2 < input && input <= 255) {
        output = ((255 - y2) / (255 - x2)) * (input - x2) + y2;
    }
    return (int)output;
}
```

반전

$I^* = 255 - I$

$= I * (-1.0) + 255$



```
int main()
{
    Mat src;
    src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("원 영상", src);

    Mat dst;
    dst = 255 - src;
    imshow("변경된 영상", dst);

    waitKey(0);
    return 0;
}
```

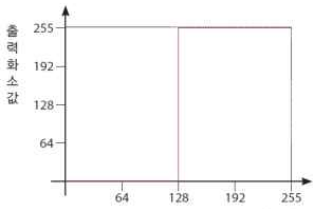
3. 이진화 : 임계치 적용(Thresholding)

- 입력 그레이 레벨이 특정 임계치보다 높으면 1
  낮으면 0

$$I'(x,y) = \begin{cases} 1 & \text{if } I(x,y) > \tau \\ 0 & \text{otherwise} \end{cases}$$

① 이진화

- 영상의 **전경(관심있는객체)**과 **배경(관심없는객체)** 분리



```
double threshold(InputArray src, OutputArray dst, double thresh,
        double maxval, int type)
```

매개 변수	설명
src	입력 영상. 1채널이어야 한다(8비트 또는 32-bit floating point).
dst	출력 영상
thresh	임계값
maxval	가능한 최대 출력값
type	이진화 종류. 우리는 THRESH_BINARY만 사용한다.

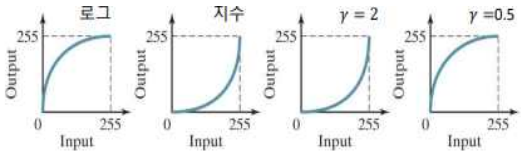
```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace std;
using namespace cv;

int main()
{
    Mat image = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    Mat dst;
    int threshold_value = 127;
    threshold(image, dst, threshold_value, 255, THRESH_BINARY);
    imshow("Original Image", image);
    imshow("New Image", dst);
    waitKey(0);
    return 0;
}
```

② LUT(Look-up Table) 사용 방법


```
#include "opencv2/opencv.hpp"
using namespace cv;
using namespace std;

int main()
{
    Mat img1 = imread("d:/Lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Original Image", img1);

    Mat table(1, 256, CV_8U);    // ①

    uchar* p = table.ptr();
    for (int i = 0; i < 256; ++i)
        p[i] = (i / 100) * 100;

    Mat img2;
    LUT(img1, table, img2);

    imshow("New Image", img2);
    waitKey(0);

    return 0;
}
```

③ 분석적 변환(analytic transformation)

1. 분석 함수(analytic function)



$$I'(x,y) = \log(I(x,y))$$ 로그함수

$$I'(x,y) = \exp(I(x,y))$$ 지수함수

$$I'(x,y) = (I(x,y))^{\gamma}$$

2. Gamma 보정

CRT : 밝기와 전압의 관계가 power 함수 관계

전처리로 모니터가 색상 표시를 균일하게 유지

기기마다 감마보정 기능 내장

```
int main()
{
	Mat src1, src2, dst;
	double gamma = 0.5;

	src1 = imread("d:/gamma1.jpg");
	if (src1.empty()) { cout << "영상을 읽을 수 없습니다." << endl; return -1; }

	Mat table(1, 256, CV_8U);
	uchar * p = table.ptr();
	for (int i = 0; i < 256; ++i)
		p[i] = saturate_cast <uchar> (pow(i / 255.0, gamma) * 255.0);
	LUT(src1, table, dst);
	imshow("src1", src1);
	imshow("dst", dst);
	waitKey(0);
	return 0;
}
```

float형으로 바뀌므로 <uchar> 사용

Look-up Table 방식

$$g(x,y) = \left(\frac{f(x,y)}{255}\right)^{\gamma} \times 255$$

3. 영상 합성

① 선형영상합성

- 두 영상의 선형 가중치 조합

$$I'(x,y) = w_A I_A(x,y) + w_B I_B(x,y)$$

$$w_A + w_B = 1$$

가중치의 합은 무조건 1

$$= \alpha * f_1(x,y) + (1-\alpha) * f_2(x,y)$$

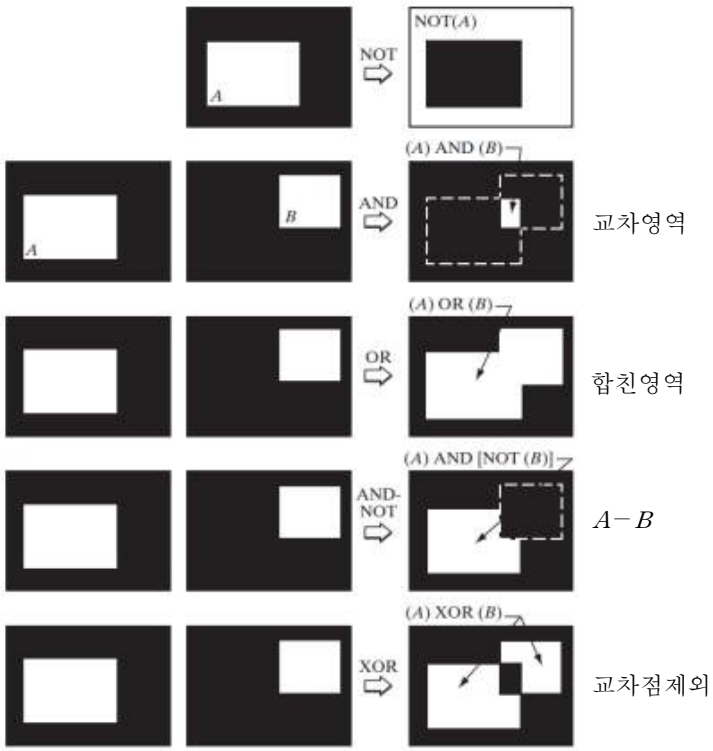
$$g(x,y) = (1-\alpha) * f_1(x,y) + \alpha * f_2(x,y)$$

```
void addWeighted(InputArray src1, double alpha, InputArray src2,
	double beta, double gamma,  OutputArray dst, int dtype=-1)
```

매개 변수	설명
src1	첫 번째 입력 영상
alpha	첫 번째 영상의 가중치
src2	두 번째 입력 영상
beta	두 번째 영상의 가중치
gamma	화소의 합계에 더해지는 값
dst	출력 영상

```
#include "opencv2/opencv.hpp"
#include <iostream>
using namespace cv;
using namespace std;

int main()
{
	double alpha = 0.5; double beta; double input;
	Mat src1, src2, dst;
	cout << "알파값을 입력하시오[0.0-1.0]: ";
	cin >> input;
	src1 = imread("d:/test1.jpg");
	src2 = imread("d:/test2.jpg");
	if (src1.empty()) { cout << "영상1을 로드할 수 없습니다." << endl; return -1; }
	if (src2.empty()) { cout << "영상2을 로드할 수 없습니다." << endl; return -1; }
	beta = (1.0 - alpha);
	addWeighted(src1, alpha, src2, beta, 0.0, dst);
	imshow("Original Image1", src1);
	imshow("Original Image2", src2);
	imshow("선형 합성", dst);
	waitKey(0);
	return 0;
}
```

② 논리 연산 정의



③ 논리적인 영상 합성

- 이진/부울 영상을 mask로 활용
- 2개의 영상을 가지고 비트별로 논리적인 연산 적용


```
int main()
{
	Mat img1, mask;

	img1 = imread("d:/scene.jpg", IMREAD_COLOR);
	if (img1.empty()) { cout << "영상1을 로드할 수 없습니다." << endl; return -1; }
	mask = imread("d:/mask.png", IMREAD_COLOR);
	if (mask.empty()) { cout << "영상2을 로드할 수 없습니다." << endl; return -1; }

	Mat dst = img1.clone();
	imshow("img1", img1);
	imshow("mask", mask);

	bitwise_and(img1, mask, dst);
	imshow("dst", dst);
	waitKey(0);
	return 0;
}
```

④ 용해(dissolve)

$$I'(x,y) = w_A I_A(x,y) + w_B I_B(x,y)$$

$w_A + w_B = 1$ 가중치의 합은 무조건 1

- 두 영상의 가중치 조합
- 가중치 조절 : 중간 영상 생성

배경 감산(background Subtraction)

- 참조/배경 영상으로부터 관심/전경 객체를 분리

$$I'(x,y) = |I(x,y) - B(x,y)|$$ 절대차

Background

Image

Absolute difference

절대차

Thresholded

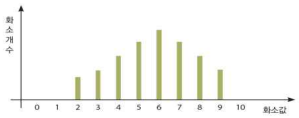
이진화임계치

4장 히스토그램

1. 히스토그램 이해

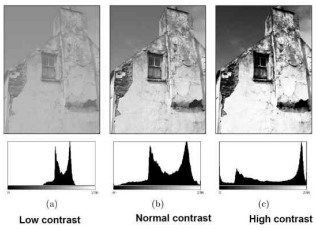
① 히스토그램이란?

- 특정한 값을 가진 화소가 영상 안에 몇 개 있는지 막대그래프로 표시한 것



화소값의 분포를 한눈에 볼 수 있음

=> 콘트라스트 개념 다시 생각

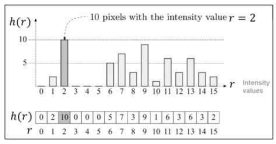


대비▲ => 히스토그램 넓음

대비▼ => 히스토그램 좁음

② 히스토그램 계산하기

- 데이터가 있는 공간(x축)을 빈(bins)으로 나누고, 각 빈이 발생한 횟수를 y축에 기록



• 히스토그램 알고리즘

```
for each pixel of the image
    value = intensity(pixel)
    histogram[value]++
end
```

```
int main()
{
    Mat src = imread("d:/lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Input Image", src);
    int histogram[256] = { 0 };

    for (int y = 0; y <src.rows; y++)
        for (int x = 0; x < src.cols; x++)
            histogram[(int)src.at<uchar>(y, x)]++;

    for (int count : histogram)
        cout << count << ",";
    waitKey(0);
    return 0;
}
```

③ 히스토그램 그리기

```
// 히스토그램을 받아서 막대그래프로 그린다.
void drawHist(int histogram[])
{
    int hist_w = 512;    // 히스토그램 영상의 폭
    int hist_h = 400;    // 히스토그램 영상의 높이
    int bin_w = cvRound((double)hist_w / 256); // 빈의 폭

    // 히스토그램이 그려지는 영상(컬러로 정의)
    Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(255, 255, 255));

    // 히스토그램에서 최대값을 찾는다.
    int max = histogram[0];
    for (int i = 1; i < 256; i++) {
        if (max < histogram[i])
            max = histogram[i];
    }

    // 히스토그램 배열을 최대값으로 정규화한다(최대값이 최대 높이가 되도록).
    for (int i = 0; i < 255; i++) {
        histogram[i] = floor(((double)histogram[i] / max)*histImage.rows);
    }

    // 히스토그램의 값을 빨강색 막대로 그린다.
    for (int i = 0; i < 255; i++) {
        line(histImage, Point(bin_w*(i), hist_h),
            Point(bin_w*(i), hist_h - histogram[i]),
            Scalar(0, 0, 255));
    }
    imshow("Histogram", histImage);
}

int main()
{
    Mat src = imread("lenna.jpg", IMREAD_GRAYSCALE);
    imshow("Input Image", src);
    int histogram[256] = { 0 };

    for (int y = 0; y <src.rows; y++)
        for (int x = 0; x < src.cols; x++)
            histogram[(int)src.at<uchar>(y, x)]++;

    drawHist(histogram);
    waitKey(0);
    return 0;
}
```

2. 히스토그램 확장

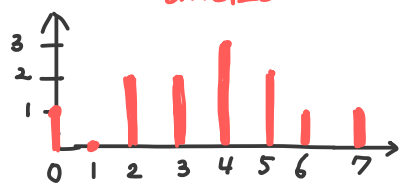
① 정규화 히스토그램 Normalized Histogram

각 빈도수를 전체 화소 개수로 나누어 계산한 히스토그램

3bit 흑백 영상 : [0~7], 크기 : 4(W) x 3(H)

전체 화소 크기 = 12

$$I = \begin{bmatrix} 7 & 4 & 2 & 0 \\ 4 & 2 & 4 & 5 \\ 3 & 3 & 5 & 6 \end{bmatrix}$$



히스토그램 $h = [1, 0, 2, 2, 3, 2, 1, 1]$

정규화 히스토그램 (해당 화소 빈도수 / 전체 화소 크기)

$\bar{h} = [\frac{1}{12}, \frac{0}{12}, \frac{2}{12}, \frac{2}{12}, \frac{3}{12}, \frac{2}{12}, \frac{1}{12}, \frac{1}{12}]$

② 누적 히스토그램 Cumulative Histogram

지정된 빈까지 누적 값 계산

히스토그램 $h = [1, 0, 2, 2, 3, 2, 1, 1]$

누적히스토그램 $hc = [1, 1, 3, 5, 8, 10, 11, 12]$ (1+0, 0+1+2, 0+1+2+2, ...)

③ 확률 밀도 함수 Probability density function

$\bar{h} = [0.083, 0, 0.167, 0.167, 0.25, 0.167, 0.083, 0.083]$

Intesity 0인 화소가 나올 확률

$\bar{h}(K) = P(K) = \frac{h(K)}{\text{전체화소크기}}$ 예) K=5 일때 $\bar{h}(5) = P(5) = \frac{h(5)}{12} = \frac{2}{12} = 0.167$

④ 누적 분포 함수 Cumulative distribution function

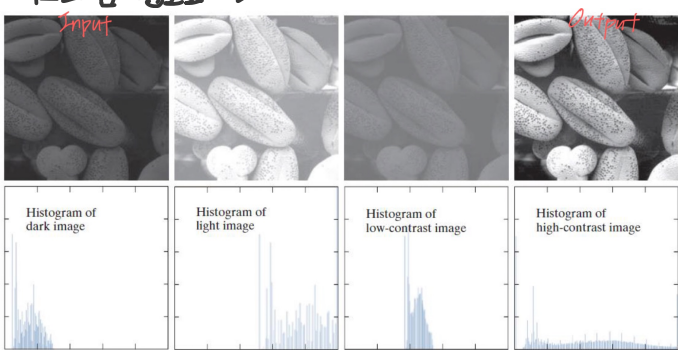
- 확률변수 까지의 누적된 확률 값

$hc = [1, 1, 3, 5, 8, 10, 11, 12]$

$cdf = [\frac{1}{12}, \frac{1}{12}, \frac{3}{12}, \frac{5}{12}, \frac{8}{12}, \frac{10}{12}, \frac{11}{12}, 1]$

* 영상대비와 히스토그램 *

- 히스토그램 분포로 밝고 어두움, 대비 (높/낮) 분석



왼쪽에 histogram 몰림 : 어두움

오른쪽에 histogram 몰림 : 밝음

히스토그램 좁음 : 대비↑

〃 넓음 : 대비↓

선형대비 늘리기 적용

Linear Contrast Stretching

하한 (min)과 상한 (max) 영역에 따른 변환 함수의 조합

$$I'(x,y) = \frac{g'_{max} - g'_{min}}{g_{max} - g_{min}} (I(x,y) - g_{min}) + g'_{min}$$

① $I(x,y) = g_{min} \longrightarrow I'(x,y) = g'_{min}$

② $I(x,y) = g_{max} \longrightarrow I'(x,y) = g'_{max}$

- 좁게 분포되어 있는 것을 넓게

$$I' = \begin{cases} 0 & \text{if } I < r_1 \\ M \times \dfrac{I - r_1}{r_2 - r_1} & \text{if } r_1 \le I \le r_2 \\ M & \text{if } I > r_2 \end{cases}$$

M : 가장 큰 Intensity

r_1과 r_2는 사용자가 지정

예) M : 255 $r_1 = 10$, $r_2 = 70$ 일때

하소값 20 ⟶ ?

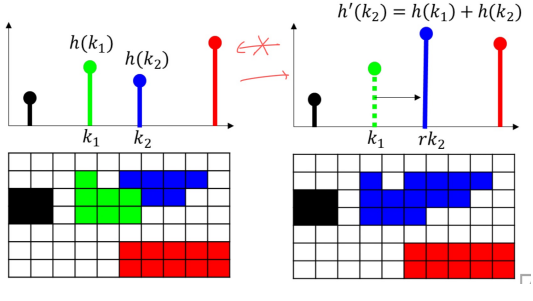
$$I' = M \times \frac{I - r_1}{r_2 - r_1} = 255 \times \frac{20 - 10}{70 - 10}$$

$$= 255 \times \frac{10}{60} = 43$$

20인 하소가 43으로 변환됨

* 콘트라스트 조절 특성 *

- 픽셀단위가 아닌 히스토그램 상에서 변경
- 역으로 조절 불가능



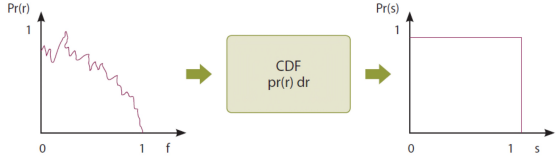
3. 히스토그램 평활화

① 히스토그램 평활화 Histogram Equalization

- 히스토그램을 균일하게 되도록 변환하는 처리

② 평활화 개념

- Pdf (확률밀도함수) 관계로 해석
- 균등 밀도 함수 (uniform Pdf)로 변환



Cdf를 변환 함수로 사용

$$S_k = T(r_K) = (L-1) \sum_{j=0}^{k} P_r(r_j)$$

평활화 절차

CDF 구하기 → CDF로부터 하소변환

→ 히스토그램 평활화 결과

히스토그램 평활화 예) $I' = ROUND((2^l - 1) \cdot \bar{c}(l))$

$$I = \begin{bmatrix} 7 & 4 & 2 & 0 \\ 4 & 2 & 4 & 5 \\ 3 & 3 & 5 & 6 \end{bmatrix}$$

결과 히스토그램: $h = [1\ 0\ 2\ 2\ 3\ 2\ 1\ 1]$

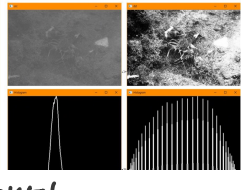
정규화된 히스토그램: $\bar{h} = [0.083\ 0.00\ 0.167\ 0.167\ 0.250\ 0.167\ 0.083\ 0.08$

$\bar{c} = [0.083\ \ 0.083\ \ 0.250\ \ 0.417\ \ 0.667\ \ 0.834\ \ 0.917\ \ 1.00]$

- Level: 8 → 7
- $ROUND(7 \cdot \bar{c}[l])$: 0→1, 1→1, 2→2, 3→3, 4→5, 5→6, 6→6, 7→7
  - 0 : 7x0.083 = 0.581 → 1
  - 1 : 7x0.083 = 0.581 → 1
  - 2 : 7x0.250 = 1.75 → 2
  - …

$$I = \begin{bmatrix} 7 & 4 & 2 & 0 \\ 4 & 2 & 4 & 5 \\ 3 & 3 & 5 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 7 & 5 & 2 & 1 \\ 5 & 2 & 5 & 6 \\ 3 & 3 & 6 & 6 \end{bmatrix}$$

```
int main()
{
    Mat src = imread("d:/crayfish.jpg", IMREAD_GRAYSCALE);
    if (src.empty()) { return -1; }

    Mat dst;
    equalizeHist(src, dst);

    imshow("Image", src);
    imshow("Equalized", dst);
    waitKey(0);
    return 0;
}
```

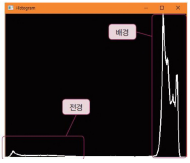


대비 향상에 최적은 아니다.

4. 전경 / 배경 분리

전경 : 관심 객체

배경 : 나머지



전경과 배경의 히스토그램을 분리할 수 있음

```
using namespace std;
using namespace cv;

int main()
{
    Mat src, dst;

    src = imread("d:/plane.jpg", IMREAD_GRAYSCALE);
    imshow("Image", src);
    if (!src.data) { return -1; }

    Mat threshold_image;
    threshold(src, threshold_image, 100, 255, THRESH_BINARY);
    imshow("Thresholded", threshold_image);
    waitKey(0);
    return 0;
}
```

오츄 알고리즘을 이용해 분리

→ threshold 함수 호출

↗ 범위 주지 않아도 배경/전경 분리 가능

threshold (src, threshold_image, 0, 255,
CV_THRESH_BINARY | CV_THRESH_OTSU);