슈티어링 6주차

목 차

컬러 영상 처리

컬러 영상 다루기

● 픽셀 값 참조

img 객체에서 (0, 0) 위치의 픽셀 값을 참조

```
Vec3b& pixel = img.at<Vec3b>(0, 0);
```

pixel에 저장된 파란색(B), 녹색(G), 빨간색(R) 색상 성분 값 확인

```
uchar b1 = pixel[0];

uchar g1 = pixel[1];

uchar r1 = pixel[2];
```

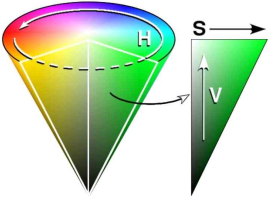
실습

```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

void color_inverse()
{
Mat src = imread("butterfly.jpg", IMREAD_COLOR);

if (src.empty()) {
cerr << "Image load failed!" << endl;
return;

}

Mat dst(src.rows, src.cols, src.type());

for (int j = 0; j < src.rows; j++) {
for (int i = 0; i < src.cols; i++) {
Vec3b& p1 = src.at<Vec3b>(j, i);
Vec3b& p2 = dst.at<Vec3b>(j, i);

p2[0] = 255 - p1[0]; // B
p2[1] = 255 - p1[1]; // G
p2[2] = 255 - p1[2]; // R

}

}

imshow("src", src);
imshow("dst", dst);
waitKey();
destroyAllWindows();
}
```

컬러 영상 다루기

- cvtColor()함수 - 영상의 색공간을 다른 색 공간으로 변환

- 색 변환 코드 COLOR_
  - BGR2GRAY 색공간변환 코드는 BGR컬러영상을 그레이스케일 영상으로 변환
  - GRAY2BGR 색공간 변환코드는 그레이스케일 영상을 BGR로 변환
  - HSV 색 모델은 색상, 채도, 명도를 색으로 표현하는 방식
  - HSV 색 공간은 원뿔 모양으로 표현



컬러 영상 다루기

- 색상 채널 나누기

- 컬러 영상을 다루다 보면 빨간색 성분만을 이용하거나 HSV 색 공간으로 변환한 후 H 성분만을 이용하는 경우

-> 3채널 Mat 객체를 1채널 Mat 객체 세 개로 분리해서 다루는 것이 효율적

- split() 함수 - 다채널 행렬을 1채널 행렬 여러 개로 변환
- merge() 함수 - 1채널 행렬 여러 개를 합쳐서 다채널 행렬 하나를 생성

실습

```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

void color_split();

int main(void)
{
color_split();
return 0;
}

void color_split() {
Mat src = imread("candies.jpg", IMREAD_COLOR);

if (src.empty()) {
cerr << "Image load failed!" << endl;
return;
}
vector<Mat> bgr_planes;
split(src, bgr_planes);

imshow("src", src);
imshow("B_plane", bgr_planes[0]);
imshow("G_plane", bgr_planes[1]);
imshow("R_plane", bgr_planes[2]);

waitKey();
destroyAllWindows();
}
```

컬러 영상 처리 기법

- 컬러 히스토그램 평활화
- 입력 영상을 밝기 정보와 색상 정보로 분리한 후,

  밝기 정보에 대해서만 히스토그램 평활화를 수행

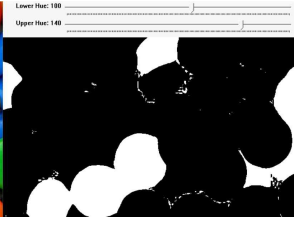
컬러 영상 처리 기법

- 색상 범위 지정에 의한 영역 분할
- 특정 색상 영역을 추출 필요
- inRange() 함수 - 행렬의 원소 값이 특정 범위 안에 있는지 확인

```cpp
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int lower_hue = 40, upper_hue = 80;
Mat src, src_hsv, mask;

void on_hue_changed(int, void*);

int main(int argc, char* argv[])
{
src = imread("candies.jpg", IMREAD_COLOR);

if (src.empty()) {
cerr << "Image load failed!" << endl;
return -1;

}

cvtColor(src, src_hsv, COLOR_BGR2HSV);

imshow("src", src);

namedWindow("mask");
createTrackbar("Lower Hue", "mask", &lower_hue, 179, on_hue_changed);
createTrackbar("Upper Hue", "mask", &upper_hue, 179, on_hue_changed);
on_hue_changed(0, 0);

waitKey(0);
return 0;
}

void on_hue_changed(int, void*)
{
Scalar lowerb(lower_hue, 100, 0);
Scalar upperb(upper_hue, 255, 255);
inRange(src_hsv, lowerb, upperb, mask);

imshow("mask", mask);
}
```

실습

- 빨강: 0, 14
- 초록: 40, 70
- 파랑 : 100, 140



컬러 영상 처리 기법

● 히스토그램 역투영

● 사람의 피부색처럼 미세한 변화가 있거나 색상 값을 수치적으로 지정하기 어려운 경우

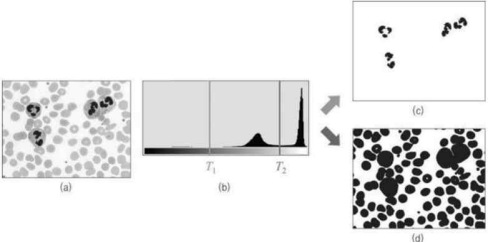
● 객체의 기준 영상을 미리 가지고 있다면 컬러 히스토그램 정보를 이용하여 비슷한 색상 영역을 찾을 수 있음

● 히스토그램 역투영 - 기준 영상으로부터 찾고자 하는 객체의 컬러 히스토그램을 미리 구하고, 주어진 입력 영상에서 해당 히스토그램에 부합하는 영역을 찾아내는 방식 & 히스토그램 모델과 일치하는 픽셀을 찾아내는 기법

이진화와 모폴로지

영상의 이진화

- 이진화
- 영상의 각 픽셀을 두 개의 부류로 나누는 작업
- 임계값을 어떻게 설정하는지에 따라 서로 다른 의미를 갖는 이진화 영상을 얻을 수 있음
- threshold() 함수 사용

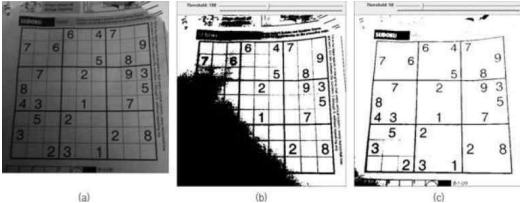


영상의 이진화

● 적응형 이진화

● 불균일한 조명 성분을 가지고 있는 영상에 대해서는 하나의 임계값으로 객체와 배경을 제대로 구분하기 어려움

-> 각 픽셀마다 서로 다른 임계값을 사용하는 적응형 이진화 기법을 사용하는 것이 효과적



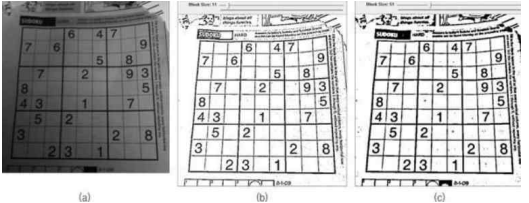
(a) (b) (c)

영상의 이진화

- 적응형 이진화

- 적응형 이진화 - 영상의 모든 픽셀에서 정해진 크기의 사각형 블록 영역을 설정하고, 블록 영역 내부의 픽셀 값 분포로부터 고유의 임계값을 결정하여 이진화하는 방식

- adaptiveThreshold() 함수 이용



모폴로지

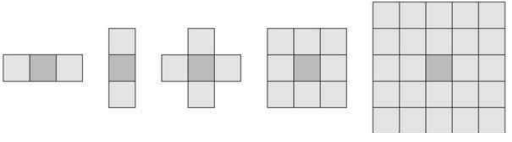
- 형태 또는 모양에 관한 학문
- 영상 처리 분야에선 영상 속 객체의 형태 및 구조를 다룸

모폴로지 연산

- 영상 내부 객체의 형태와 구조를 분석하고 처리하는 기법
- 수학적 모폴로지라고도 함
- 그레이스케일 영상과 이진 영상에 모두 적용 가능
  But! 주로 이진 영상에서 객체의 모양을 변형하는 용도로 사용
  예) 객체의 모양을 단순화, 잡음제거

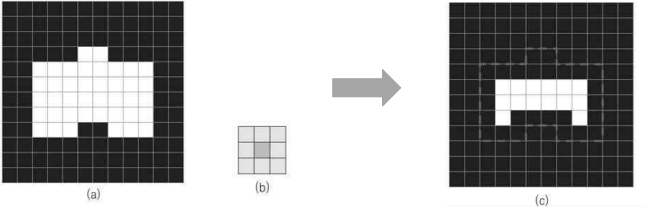
구조 요소

- 모폴로지 연산을 정의하기 위해선 구조 요소를 먼저 정의해야 함
- 구조 요소는 모폴로지 연산의 동작을 결정하는 작은 크기의 행렬
  ≒ 필터링에서 사용되는 마스크
- 다양한 크기와 모양으로 정의할 수 있지만 대부분 3x3 정방형 구조 요소 사용
- 모폴로지 연산 결과가 저장될 위치를 나타내는 고정점 존재
- 고정점은 대부분 구조 요소의 중심이 됨



이진 영상의 침식

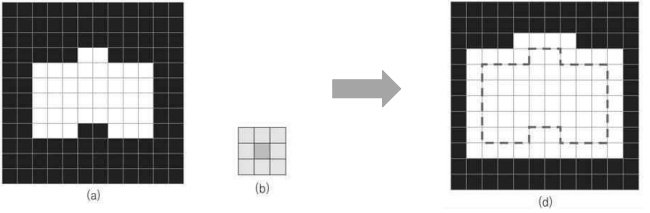
- 침식 연산은 객체 외곽을 축소하는 연산
- 전체적으로 객체 영역은 축소, 배경 영역은 확대
- 구조 요소를 영상 전체에 대해 스캔하여 구조 요소가 객체 영역 내부에 완전히 포함될 경우 고정점 위치 픽셀을 255로 설정



원본과 구조 요소 침식 연산 수행 결과

이진 영상의 팽창

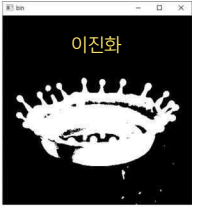
- 팽창 연산은 객체 외곽을 확대하는 연산
- 전체적으로 객체 영역은 확대, 배경 영역은 축소
- 구조 요소를 영상 전체에 대해 이동시켜서 구조 요소와 객체 영역이 한 픽셀이라도 만날 경우 고정점 위치 픽셀을 255로 설정

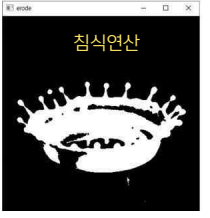


원본과 구조 요소 팽창 연산 수행 결과

OpenCV에서 침식과 팽창

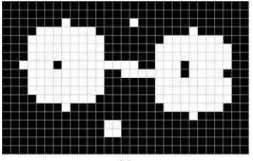
- 원소 값이 0 또는 1로 구성된 Mat 행렬로 구조 요소를 표현, 이때 값이 1인 원소만 구조 요소의 모양을 결정
- 이러한 구조 요소 행렬을 생성할 수 있도록 getStructuringElement() 함수 제공
- 침식 연산은 erode() 함수를 이용하여 수행
- 팽창 연산은 dilate() 함수를 이용하여 수행



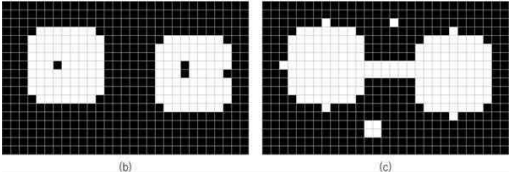


이진 영상의 열기와 닫기

- 열기 연산은 입력 영상에 대해 침식 연산을 수행한 후, 다시 팽창 연산을 수행하는 연산
- 닫기 연산은 입력 영상에 대해 팽창 연산을 수행한 수, 다시 침식 연산을 수행하는 연산



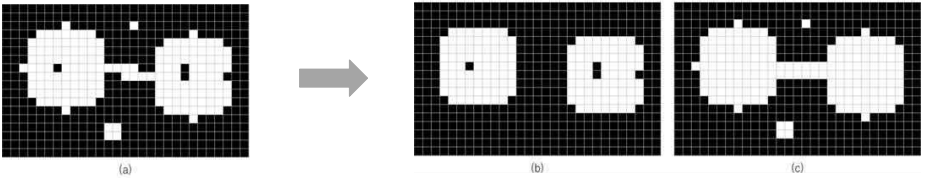
원본



열기 연산과 닫기 연산

이진 영상의 열기와 닫기

- 객체 영역의 크기는 크게 변화하지 않지만, 적용하는 순서에 따라 효과가 달라짐
- 열기 연산은 침식 연산을 먼저 수행하여 영역이 조금 제거된 후 팽창 연산이 수행되므로 이진 영상에 존재하는 점과 같은 작은 크기의 객체가 효과적으로 제거
- 닫기 연산은 팽창 연산을 먼저 수행하여 객체의 작은 구멍 등이 메워진 후 침식 연산이 수행되므로 객체 내부의 작은 구멍이 효과적으로 제거

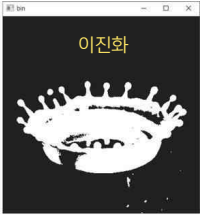


원본

열기 연산과 닫기 연산

OpenCV에서 열기와 닫기

- 열기와 닫기 연산을 morphologyEx() 함수를 통해 수행

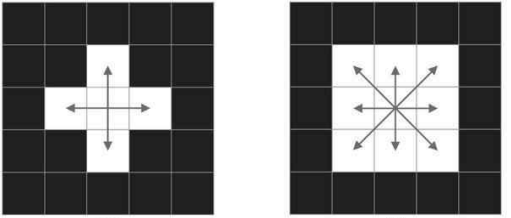


레이블링

- 이진 영상에서 흰색으로 구분된 각 객체 영역에 고유의 번호를 저장하는 기법
- 영상 내부에 있는 객체의 위치, 크기, 모양 등 특징을 분석함
- 객체 인식을 위한 전처리 과정으로 자주 사용
- 픽셀 값이 0인 것은 배경으로, 픽셀 값이 0이 아닌 것은 객체로 간주
- 하나의 객체를 구성하는 모든 픽셀에 같은 레이블 번호를 지정

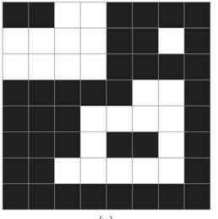
레이블링 수행시 픽셀의 연결 관계

- 픽셀 간의 연결 관계는 두 가지 방법으로 정의
- 첫 번째, 특정 픽셀의 상하좌우로 붙어있는 픽셀끼리 연결되어 있다고 정의하는 4-방향 연결성
- 두 번째, 상하좌우뿐만 아니라 대각선 방향으로 인접한 픽셀도 연결되어 있다고 간주하는 8-방향 연결성



레이블맵

- 레이블링 수행으로 각각 객체 영역에 고유 번호가 매겨져 생성된 2차원 정수 행렬



(a)

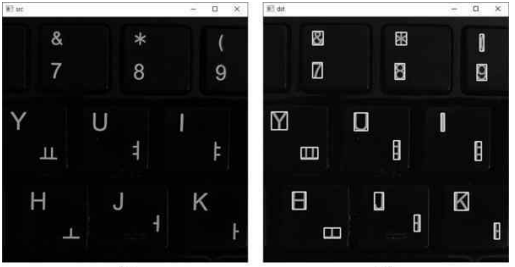
0	0	1	1	0	0	0	0
1	1	1	1	0	0	2	0
1	1	1	1	0	0	0	0
0	0	0	0	0	3	3	0
0	0	0	3	3	3	3	0
0	0	0	3	0	0	3	0
0	0	3	3	3	3	3	0
0	0	0	0	0	0	0	0

(b)

OpenCV에서 레이블링

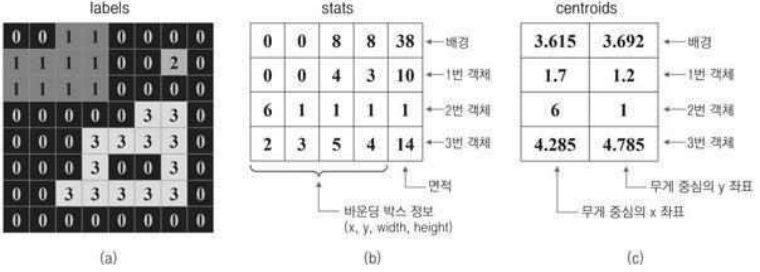
- 3.0.0 버전부터 레이블링을 수행하는 connectedComponents() 함수 제공

실습 – 키보드 문자 영역 추출하기



(a) (b)

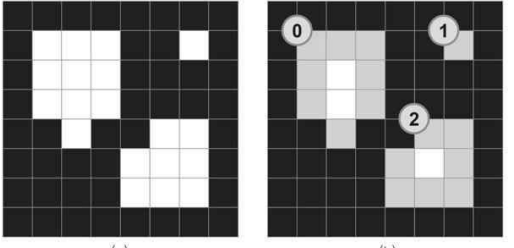
```cpp
//src 영상을 오츠 알고리즘으로 이진화하여 bin에 저장
Mat bin;
threshold(src, bin, 0, 255, THRESH_BINARY | THRESH_OTSU);

//bin 영상에 대해 레이블링을 수행하고 각 객체 영역의 통계 정보를 추출
Mat labels, stats, centroids;
int cnt = connectedComponentsWithStats(bin, labels, stats, centroids);
```



```cpp
//src 영상을 3채널 컬러 영상 형식으로 변환하여 dst에 저장
Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);

//배경 영역을 제외하고 흰색 객체 영역에 대해서만 반복문 수행
for (int i = 1; i < cnt; i++) {
    int* p = stats.ptr<int>(i);

    //객체의 픽셀 개수가 20보다 작으면 잡음으로 간주하고 무시
    if (p[4] < 20) {
        continue;
    }

    //검출된 객체를 감싸는 바운딩 박스를 노란색으로 그림
    rectangle(dst, Rect(p[0], p[1], p[2], p[3]), Scalar(0, 255, 255), 2);
}
```

외곽선 검출

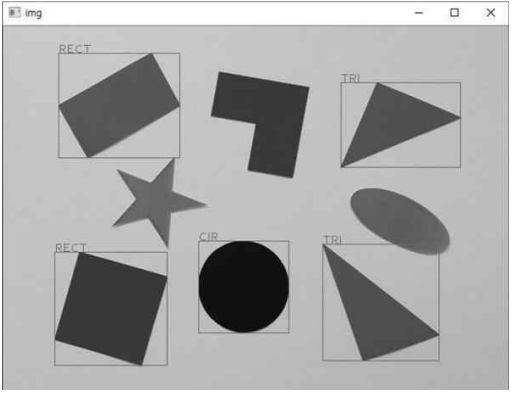
- 이진 영상에서 객체의 위치 및 크기 정보를 추출하는 방법 중 하나
- 객체의 외곽선은 객체 영역 픽셀 중에서 배경 영역과 인접한 일련의 픽셀을 의미
- 객체 내부에 홀이 존재한다면 홀을 둘러싼 픽셀들도 외곽선으로 검출 가능
- 따라서 객체의 외곽선은 객체 바깥쪽 외곽선과 안쪽 홀 외곽선으로 구분



OpenCV에서 외곽선 검출

- 영상 내부 객체들의 외곽선을 검출하는 findContours() 함수 제공
- 외곽선 정보로부터 객체의 크기를 알 수 있는 바운딩 박스를 구하는 boundingRect() 함수 제공
- 외곽선이나 점을 감싸는 최소 크기의 회전된 사각형을 구하는 minAreaRect() 함수, 최소 크기의 원을 구하는 monEnclosingCircle() 함수 제공
- 임의의 외곽선 정보를 가지고 있을 때 외곽선이 감싸는 영역의 면적을 알려주는 contourArea() 함수 제공

실습 – 다각형 인식 프로그램



```cpp
//img 영상에서 pts 외곽선 주변에 바운딩 박스를 그리고 label 문자열을 출력하는 함수
void setLabel(Mat & img, const vector<Point>&pts, const String & label)
{
    //pts 외곽선을 감싸는 바운딩 박스를 구함
    Rect rc = boundingRect(pts);

    //바운딩 박스를 주황색으로 표시
    rectangle(img, rc, Scalar(0, 0, 255), 1);

    //바운딩 박스 좌측 상단에 label 문자열 출력
    putText(img, label, rc.tl(), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255));
}
```

```
//img 영상을 그레이스케일 형식으로 변환하여 gray에 저장
Mat gray;
cvtColor(img, gray, COLOR_BGR2GRAY);

//gray 영상을 오츠 알고리즘으로 자동 이진화하여 bin에 저장
Mat bin;
threshold(gray, bin, 200, 255, THRESH_BINARY_INV | THRESH_OTSU);

//bin 영상에서 모든 객체의 바깥쪽 외곽선을 검출
vector<vector<Point>> contours;
findContours(bin, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);
```

```cpp
//검출된 각 객체의 외곽선 좌표를 pts 변수로 참조하여 반복문 수행
for (vector<Point>& pts : contours) {

    //외곽선이 감싸는 면적이 400보다 작으면 잡음으로 간주하고 무시
    if (contourArea(pts) < 400) {
        continue;
    }

    //pts 외곽선을 근사화하여 approx에 저장
    vector<Point> approx;
    approxPolyDP(pts, approx, arcLength(pts, true) * 0.02, true);

    //approx에 저장된 외곽선 점의 개수를 vtc에 저장
    int vtc = (int)approx.size();

    //근사화된 외곽선의 꼭지점 개수가 3이면 삼각형으로 인식
    if (vtc == 3) {
        setLabel(img, pts, "TRI");
    }

    //근사화된 외곽선의 꼭지점 개수가 4이면 사각형으로 인식
    else if (vtc == 4) {
        setLabel(img, pts, "RECT");
    }

    //근사화된 외곽선의 꼭지점 개수가 4보다 크고 객체의 면적 대 길이 비율을 조사하여 원에 가까우면 원으로 인식
    else if (vtc > 4) {
        double len = arcLength(pts, true);
        double area = contourArea(pts);
        double ratio = 4. * CV_PI * area / (len * len);

        if (ratio > 0.8) {
            setLabel(img, pts, "CIR");
        }
    }
}
```

더글라스-포이커 알고리즘