UW1-271B

Computer Vision: is study of getting the valuable data from stopped video. In other word, 사람이 눈으로 사물을 보고 인지하는 작업을 컴퓨터가 동등하게 수행할 수 있도록 연구하는 학문.

Camera = Human eyes.

Algorithm = brain

Ex: 사과, 배경을 제외하고 가운데 영역에 빨간색 성분이 많고, 둥근 윤곽을 가지고 잇으면 사과라고 인식 하게 만들수있다, 하지만 초록색 사과라면? 색상정보에 초록색 추가

밝기, 색상, 모양, 텍스처 등이 주로 사용된다, 이 정보와 머신러닝 알고리즘을 함께 사용 -> 사물인식.

Image processing

대부분의 스마트폰 카메라에서 지원하는 HDR (High Dynamic Range) 사진 촬영, 인스타그램의 필터 기능 등은 영상의 화질을 개전 하여 보기 좋은 사진을 만드는 용도로 사용되고있습니다.

태양의 가시 광선 또는 특정 광원에서 발생한 빛이 (The lights from Sun reflect from object and that reflected light comes to lens)카메라 lens 를 통해 카메라 내부로 들어옴.

Now the camera curve the light help to gather to image sensor.

Obj -> lens -> sensor -> ISP -> jpg or png

Pixel => width (w-1) start from 0

Height(h-1) start from 0

(행 개수) x (열 개수) in OpenCV

가로 크기 x 세로 크기

Grayscale level: 0 ~ 255 integer (0-black, 255-white).

C/C++ use unsigned char, you can store 0 to 255.

Typedef unsigned char BYTE // windows

Typedef unsigned char uint8\_t; // linux

Typedef unsigned char uchar; // OpenCV lib

Use RGB: EX (255,0,0) == red

Png => Portable Network Graphcis: 무손실 압축

core,videoio,imgcodecs(영상 입출력) -> imgproc, photo (전처리), imgproc,features2d(특징추출), imgprocobjdetect(객체 검출, 영상 분할), calib3d, video, stitching, ml, dnn (분석 객체 인식 포즈 추청, 움직임 분석) -> highgui, ml, dnn (화면 출력 최종판단).

Core module: used for convert into arr or save the file.

Videoio and imgcodecs is used for calling video file or stopped video file.

Imgproc is used for basic 영상 처리 기능 (image processing).

Features2d, objdetect, video is used for identify specific point, moving pattern.

Ml, dnn is used for video recognition and decision.

OpenCV provides all modules in one compact module which is world.

Pros and Cons of OpenCV

Fast download, provide built in DLL and LIB files.

You can modify the code to mathematical optimization.

Con

Only available for window 64 bits, take long time to built.

In OpenCV, the video data is representing using Mat class.

Since Mat class is stated inside namespace cv, therefore cv::Mat is proper way.

Since video’s element consists of 0 to 255 2d arr

To display image

#include <iostream>  
#include “opencv2/opencv.hpp”

int main(void) {  
 cv::Mat img;  
 img = cv::imread(“lenna.bmp”);  
 if (img.empty()) {

std::cerr << “Fail to read Image” << std::endl;  
 return -1;

}  
 cv::namedWindow(“image”);  
 cv::imshow(“image”,img);  
 cv::waitKey();  
 return 0;

}

Without using OpenCV, it would be hundred line of code to read and display bmp file in separate window (console).

cv::Mat  
주요함수: imread(), namedWindow(), imshow(), keyWait()

// file name = the filename, flags = select the imreadModes  
// return type is Mat obj which is the video data  
Mat imread(const String& filename, int flags = IMREAD\_COLOR);  
flags: 영상 파일을 불러올 때 사용할 컬러 모드와 영상 크기를 지정하는 플래그.

Flags 인자에는 ImreadModes 를 지정할수있다. Default value is IMREAD\_COLOR (3channel color BGR).

ImreadModes consists

* IMREAD\_UNCHANGED (그대로 사용).
* IMREAD\_GRAYSCALE (1채널 그레이스케일 영상 변환)
* IMREAD\_COLOR (3채널 bgr 컬러 영상)
* IMREAD\_REDUCED\_GRAYSCALE\_2 (1/2로 크기를 줄인 그레이스케일)

Mat obj 에 저장된 영상데이터를 파일로 저장하기 위해서는 imwirte() 함수를 사용한다.

Bool imwrite(const String& filename, InputArray img, const std::vector<int>&params = std::vector<int>());

* filename: just file name
* img: 저장할 영상 데이터 (Mat obj).
* params: 저장할 영상 파일 형식에 의존적인 파라미터
* return value = bool true if saved false if not.

Ex: 만약 lenna.jpg 파일로 저장할 때 JPEG 압축률 을 95% 저장하고싶다면 다음과 같이 코드 작성

vector<int> params;

// imwrite\_jpeg\_quality 플래그 는 jpeg 압출률을 나타내는 option flag.  
params.push\_back(IMWRITE\_JPEG\_QUALITY);  
params.push\_back(95);  
imwrite(“lenna.jpg”, img, params);

void namedWindow(const String& winname, int flags = WINDOW\_AUTOSIZE);

windowFlag의 열거형 상수

* WINDOW\_NORMAL: 사용자가 자유자제로 크기 변경가능
* WINDOW\_AUTOSIZE: 사용자 크기 변환 제약
* WINDOW\_OPENGL: provide openGL

void destoryWindow(const String& winname);

소멸시킬 창 이름, 운영체제가 자동으로 지워주지만, 프로그램이 동작중일때 지우고 싶을때 사용

창 크기를 바꾸거나 위치를 바꿀수 있는 함수  
moveWindow()   
void moveWindow(const String& winname, int x, int y)

Winname: 위치를 이동할 창 이름  
x: 창이 이동할 위치 좌표  
y: 창이 이동할 위치 좌표

프로그램이 동작중에 영상 출력 창의 크기를 변경하고 싶다면 resizeWindow() 함수 사용

void resizeWindow(const String& winname, int width, int height);

Chapter 3 Basic Class

* 픽셀의 좌표를 표현하는 Point\_ 클래스
* 사각형 영역의 크기를 표현하는 Size\_ 클래스
* 사각형의 위치와 크기 정보를 나타내는 Rect\_ 클래스

Point\_ class

// pt1 = (0,0)  
Point pt1;   
// pt1 = (5,10)  
pt1.x = 5; pt1.y = 10;  
// pt2 = (10,30)  
Point pt2(10,30);

You can also calculate the location of points

// pt1 = [5,10] , pt2 = [10,30]

// 15, 40  
Point pt3 = pt1 + pt2;

// 10, 20  
Point pt4 = pt1 \* 2;

// Point::dot() function is 두점의 내적을 계산 하는 맴머함수  
x 좌표 끼리 곱한 값에 y 좌표 끼리 곱한값을 더한다.  
int d1 = pt1.dot(pt2);  
또한 bool 로 표현이 가능 (두 포인트가 같은지 아닌지 등등)

Bool b1 = (pt1 == pt2);

It also provides standard output, which means std::cout << “pt1” << pt1 << std::endl; works

Size\_ 클래스

// means sz1 = [0 x 0], sz2 = [10 x 20]   
Size sz1, sz2 (10,20);

sz1.width = 5; sz1.height = 10; // sz1 = [5x10]

Point class 와 마찬가지로 사칙연산 가능  
int area1 = sz4.area() 넓이 구해주는 function.

Rect\_ 클래스

사각형의 위치와 크기 정보를 표현할 때에는 Rect\_ 클래스를 사용.

Ex:  
Rect rc1; // rc1 = [0 x 0 from (0,0)]  
Rect rc2(10,10,60,40); // rc2 = [60 x 40 from (10,10)]  
좌표 (10,10) 에서 시작해서 크기가 60x40 사각형.

Basic 사칙연산 가능 with Size class and Point class  
// rc1 = [0 x 0 from (0, 0)], rc2 = [60 x 40 from (10, 10)]

Rect rc3 = rc1 + Size (50,40) //set height and width 40,50 // rc3 = [50 x 40 from (0, 0)]  
Rect rc4 = rc2 + Point (10, 10) // rc4 = [60 x 40 from (20,20)]

// rc3 = [50 x 40 from (0,0)] , rc4 = [60 x 40 from (20,20)]  
&: Rect rc5 = rc3 & rc4; // rc5 = [30 x 20 from (20,20)] (두 사각형이 곂치는 부분)

|: Rect rc6 = rc3 | rc4; // 두 사각형을 모두 포함하는 최소 크기의 사각형 반환

RotatedRect 클래스

RoatateRect 클래스는 회전된 사각형의 중심 좌표를 나타내는 center, 사각형의 가로 및 세로 크기를 나타내는 size 최전 각도 정보를 나타내는angle을 맴버 변수로 가지고 있는다. (시계방향 기준)

RotatedRect rr1(Point2f(40,30), Size2f(40,20), 30.f);  
if you want to know four edges of the rect, then RotatedRect::point();  
Ex:

RotatedRect rr1(Point2f(40,30),Size2f(40,20), 30.f);

Point2f pt[4];

Rr1.points(pts);

// output

Pts[0] = (17.679~~)

Pts[1] = (27.679~~)

Pts[2] = (67.679~~)

Pts[3] = (57.679~~)  
경우에 따라서는 회전된 사각형을 감싸는 최소 크기의 사각형 정보가 필요합니다.

bounding box 라고 한다.  
RotatedRect::boundingRect() 함수사용.  
Rect br = rr1.boundingRect(); 🡨 integer

Rect br = rr1.boundingRect2f(); 🡨 float

Range 클래스

Start to end

[Start, end).

Range::size() return end – start  
Range::empty() return true if start == end;

Ex:  
Mat img[3];

For (int I = 0; I < 3; i++) {  
String filename = fotmat(“test%02d.bmp”, i+1);  
img[i] = imread(filename);  
}

Mat 클래스

Mat 클래스를 이용해서 행렬, 영상 생성, 복사, 부분 행렬 추출, 행렬 정보 참조 등등 설명.  
Mat::dim(): represents Mat array’s dimension, default value is 2-dimension array  
Mat:rows and Mat::cols: represent the size of 2-d array, if the obj is carry a video then  
Mat::rows represents row pixels and Mat::cols represents column pixel.

Mat::data represents the pointer of elements in array, of course if nothing contains then NULL or nullptr.

BGR, 세 개의 색상 성분

성분 요소 == channel  
CV\_8UC1 represent unsigned char and use one channel

CV\_8UC3 represent unsigned char and use all three channel.

행렬의 생성 과 초기화  
기본 생성자: Mat img1; 이렇게 생성된 obj 는 비어 있는 행렬, which means img1.rows and img1.cols is 0 and of course img1.data is NULL.

Mat 객체를 생성함 과 동시에 원소 값 저장을 위한 메모리 공간을 할당하려면 다응ㅁ 생성자를 사용합니다.

Mat::Mat(int rows, int cols, int type)

* Rows: 새로 만들 행렬의 행 개수 (영상의 세로 크기)
* Cols: 새로 만든 행렬의 열 개수 (가로 크기)
* Type: 새로 만들 행렬의 타입

ㅅㅂ 반대임 ㅡㅡ

Mat img2 (480, 640, CV\_8UC1); // unsigned char, 1-channel (세로 480, 가로 640).

Mat img3 (480, 640, CV\_8UC3); // unsigned char, 3-channel (세로 480, 가로 640).

Typically, CV\_8UC1 is use for grayscale and CV\_8UC3 is used for true color.

Instead of using Mat img2(480,640,CV\_8UC1), you can also use

Mat::Mat(Size size, int type);

* Size(cols, rows) or Size(width, height)
* Type 새로만들 정렬 타입

Mat img4(Size(640, 480), CV\_8UC3); // size(width, height) 가로 640, 세로 480

위 처럼 만들면, data 가 Mat obj 에 들어가기 전까진 garbage value 로 채워진다.

그러므로 Mat obj 를 생성함과 동ㅇ시에 모든 원소 값을 특정 값으로 초기화 하는것이 안전하다.

Mat::Mat(int row, int cols, int type, const Scalar& s);  
Mat::Mat(Size size, int type, const Scalar& s);

* S: 행렬 원소 초기값

Scalar 클래스는 4 개의 실수 값을 저장할수 있는 OpenCV 클래스 이다. (주로 픽셀 값을 표현 하는 용도로 사용됨).

보통 grayscale 할때 (scalar 하나의 변수만 사용)  
Mat img5 (480, 640, CV\_8UC1, Scalar(128));

0,0,255 => red

Using of Scalar, the color represent (BGR order).

새로운 행렬을 생성할때, 모든 원소 값을 0 으로 초기화 하는 경우가 많으며, 이러한 경우 Mat 클래스의 행렬 원소 초기값에 Scalar(0) 을 지정해 주면 된다.

또는 Mat::zeros() 사용해 다 초기화

Static MatExpr Mat::zeros(int rows, int cols, int type);  
static MatExpr Mat::zeros(Size size, int type);

Mat mat1 = Mat::zeros(3,3,CV\_32SC1);

행렬의 모든 원소가 1 로 초기화 된 행렬을 생성 하려면 Mat::ones() 함수 사용

또한 행렬 연산에서 자주 사용되는 단위 행렬 (identity matrix) 을 생성하려면 Mat::eye() 사용

Mat 객체를 생성할때, 행렬 원소를 저장할 메모리 공간을 새로 할당하는 것이 아니라, 기존에 이미 할당되어 있는 메모리 공간의 데이터를 행렬 원소 값으로 사용할수있다.

객체 생성이 빠르다.

Mat::Mat(int rows, int cols, int type, void\* data, size\_t step = AUTO\_STEP)

Ex:   
float data[] = {1,2,3,4,5};  
Mat mat4 (2,3,CV\_32FC1, data);

Mat4 = {1,2,3}  
 {4,5,6}

Shallow copy

cv::Mat img1 = imread(“dog.bmp”);

cv::Mat img2 = img1;

Deep copy (픽셀 데이터를 공유 하는것이 아니라 메모리 공간을 새로 할당하여 픽셀 데에터 전체를 복사 하고 싶다면 cv::Mat::clone() 또는 cv::Mat::copyTo() 를 사용  
cv::Mat::clone () return \*this;

cv::Mat::copyTo() return mask

clone() 동일한 객체 완전히 복사 후 반환

ex: cv::Mat img4 = img1.clone();

void MatOp2() {

// original image

cv::Mat img1 = cv::imread("dog.bmp");

// shallow copy

cv::Mat img2 = img1;

cv::Mat img3;

// shallow copy

img3 = img1;

// deep copy

cv::Mat img4 = img1.clone();

cv::Mat img5;

img1.copyTo(img5);

// set 1 to yellow

img1.setTo(cv::Scalar(0, 255, 255));

cv::imshow("img1", img1);

cv::imshow("img2", img2);

cv::imshow("img3", img3);

cv::imshow("img4", img4);

cv::imshow("img5", img5);

cv::waitKey();

cv::destroyAllWindows();

}

위 결과 shallow copy 를 한 녀석들은 original 이 바뀌었을때, 다 같이 바뀜

But deep copy 한 놈들은 건재히 살아있음이 보인다.

부분 형렬 추출

영상에서 사각형 모양의 부분 영상을 추출하거나 참조하는 방법에 대해 공부

특정 사각형 영역 의 부분 행렬을 추출 하고 싶을때에는 Mat 클래스에 정의된 괄호 연산자 재정의 사용.

Mat Mat::operator() (const Rect& roi) const;  
Mat Mat::operator() (Range rowRange, Range colRange) const;

* roi: 사각형 관심 영역
* rowRange (관심 로우)
* colRange (관심 컬름)
* return value = 추출한 부분 행렬 또는 영상. 부분 영상의 픽셀 데이터를 공유.

cv::Mat img1 = imread(“cat.bmp”);  
cv::Mat img2 = img1(Rect(220, 120, 340, 240));

img1(Rect(220, 120, 340, 240)) 코드 는 img1 영상 (220,120) 좌표부터 340x240 크기 만큼의 사각형 부분 영상을 추출하는 코드이다. 그 부분을 img2 에 shallow copy.

부분 영상을 추출할 때 주의할 점은 Mat 클래스의 괄호 연산자를 이용하여 얻은 부분 영상이 독립된 메모리 공간을 확보아여 복사하는 깊은 복사가 아니고, 픽셀 데이터를 공유하는 얕은 복사.

즉 부분 영상을 추출한 후 부분 영상의 픽셀값 변경 -> 원본 변경 으로 이어진다.

영상의 반전: Mat 클래스 타입의 변수 앞에 ~ 연산자를 붙이는 방법.

img2 = ~img2  
위 코드 는 img2 영상을 반전 하여 그 결과를 다시 img2 에 저장.

이렇게부분 영상 참조 기능은 입력 영상에 사각형 모양의 괌심 영역 (ROI, region of interest) 을 설정하는 용도로 사용할 수 있습니다.

사각형이 아닌 임의의 모양의 ROI 를 설정하고 싶은 경우에는 마스크 연산을 응용할 수 있으며, 마스크 연산에 대해는 챕터4

만약 deep copy 를 사용하고싶다면, simply put clone() behind.

cv::Mat img3 = img1(cv::Rect(220, 120, 340, 240)).clone();

cv::Mat 행렬에서 특정 범위의 행 또는 열 부분을 따로 추출할려면

cv::Mat::rowRange() or cv::Mat::colRange() 함수 사용

cv::Mat cv::Mat::rowRange(int startrow, int endrow) const;  
cv::Mat cv::Mat::rowRange(const Range& r) const;

* startrow: 추출할 행 범위 시작 번호 (포함)
* endrow: 끝 번호 (포함하지 않는다).
* r: 추출할 행 범위
* return: 지정한 행 범위에 해당하는 행열

행렬의 원소 값 참조

Opencv 라이브러리 는 다양한 영상 처리에 필요한 기능을 구현한 클래스와 함수 를 제공해줌

하지만 직접 자신만의 알고리즘을 구현 하여 적용해야한다면? 이때 필요한 기능이 영상의 픽셀 값 을 참조하는 기능

픽샐값 참조 기능:

cv::Mat::at() 함수

OpenCV 에서 제공하는 가장 직관적인 행렬 원소 접근 방법. 보통 take params (rows and cols) and used template

Template <typename \_Tp>

\_Tp& cv::Mat::at(int y, int x);

* y:참조할 행 번호
* x:참조할 열 번호
* return value (\_Tp& 타입으로 형변환된) y 번째 행 x 번재 열의 원소값.

Mat::at() 함수는 템플릿 으로 저장되어있기때문에, data type 지정이 필요하다.

CV\_8UC1 이면 uchar 로 지정, CV\_32FC1 타입이면 float.  
CV\_8UC3 (3 channel) 이면 Vec3b 로 지정해야한다.

만약 Mat obj img1 에 그레이스케일 영상이 저장되어 있을경우 (x,y) 좌표 필색값을 참조할려면

img1.at<uchar>(y,x) 형태의 코드를 작성해야한다.

cv::Mat mat1= cv::Mat::zeros(3,4, CV\_8UC1);

for(int I = 0; I < mat1.rows; i++) {

for (int j = 0; j < mat.cols; i++) {

mat1.at<uchar>(I,j)++;

}

}

Mat::ptr() 함수 사용법

행렬 원소 접근 방법

Mat::ptr() 함수는 특정행의 첫번째 원소 주소값 반환.  
Mat::at() 보다 빠르다, 하지만 cols 로만 연산을 수행해서, 임의 자표 원소에 빈번하게 접근해야한다면,

Map::at() 사용.

Template<typename \_Tp>  
\_Tp\* Mat::ptr(int y) // 주소값 반환

* y:참조할 행 번호
* return: y번째 시작 주소

for (int i = 0; I < mat1.rows; i++) {

uchar\* p mat1.ptr<uchar>(i);

for(int j = 0; j < mat1.cols; j++) {

p[i]++;

}

}

MatIterator\_ 반복자

C++ iterator 사용법과 유사하다.

Mat::begin() 함수를 사용해서 arr 의 첫번째 원소 위치를얻을수있다.

Mat::end() 함수로 마지막 원소 바로 다음 위치를 얻을수있다.

MatIterator\_ 를 사용해서 mat1 행렬의 모든 원소값을 1씩 증가 시키는 예제 코드:

CV\_8UC1 == uchar

for(MatIterator\_<uchar> it = mat1.begin<uchar>(); it != mat.end<uchar>(); ++it) {

(\*it)++;

}

Mat::ptr() 보다 느리고, mat::at() 함수처럼 임의의 위치에 자유롭게 접근할수가없다.

행렬 정보 참조하기 3.2.6

For example, calling lenna.bmp picture but you just want to know how big is the bmp pixel is then you may want to use the following code

cv::Mat img1 = img1.imread(“lenna.bmp”);

std::cout << img1.rows << std::endl;  
std::cout << img2.cols << std::endl;

because lenna.bmp contains pixel of 512x512, therefore, it will simply printout

512, 512

Mat class Mat::data is 행렬 원소 데이터가 저장되어 있는 메모리 공간의 시작 주소를 가ㄹ키는 ㅍ인터 입니다.

Mat::data 맴버 변수가 가리키는 메모리주소를 활용 해서 사용자가 직접 참조 가능하다.

하지만 잘못 연산 할시에 문제가 생김으로, 앞에 설명했던 Mat::at(), Mat::ptr() 함수를 사용했다.

Mat::rows and Mat::cols

int cv::Mat::channels() const // 행렬의 채널 수 를 반환한다.

int cv::Mat::depth() const // 행결의 깊이를 반환한다.

**size\_t** Mat::elemSize() const //return each element’s data type (in byte)

bool Mat::empty() const // return true if it is empty or return false if some data is already in the elements.

bool Mat::isContinuous() const // if there are no empty data between in elements in arr, return true.

**Size** Mat::size() const // return the size of the arr

Int Mat::type() const // return the type of the element ex CV\_8UC1 like htis

예를 들어 현재 다루고 있는 영상 객 체가 그레이스케일 영상인지 혹은 3채널 컬러 영상인지를 확인하려면 Mat::type() 을 사용.

In code we can write

cv::Mat img1 = imread(“Something.bmp”);  
if (img1.empty()) {

std::cerr << “Fail To Read Image” << std::endl;

}

If (img1.type() == CV\_8UC1) {

std::cout << “The Image is GrayScale << std::endl;

} else {

std::cout << “The data image is either full or 2-channel << std::endl;

}

static inline

std::ostream& operator << (std:ostream& output, const cv::Mat& mtx);

* output == output, overloading
* reference obj Mat, 출력할 행렬
* return: c++ standard output

float data[] = {2.f, 1.414f, 3.f, 1.732f};

cv::Mat mat1 (2,2, CV\_32FC1,data);

The new line of the arr is contains ; (semicolon);

[2, 1.414;

3, 1732]

It also contains comma.

3-11 MatOp5()

레나 사진 파일에서 불러온 레나 영상 크기 채널 수 타입 정보를 확인후

출력.

그리고 작은 크기의 행렬을 정의하고, cout 연산자를 이용해서 행렬의 모든 원소를 화면에 출력.

void MatOp5 () {

cv::Mat img1 = imread(“lenna.bmp”);

std::cout << “Width” << img1.cols << std::endl;

std::cout << “Height” << img1.rows << std::endl;

std::cout << “Channels” << img1.channels() << std::endl;

// grayscale check

if (img1.type() == CV\_8UC1) {

std::cerr << “The image is gray scale image” << std::endl;

}

}