슈티어링 7주차

목 차

<13장 객체 검출>

- 템플릿 매칭
- 캐스케이드 분류기와 얼굴 검출
- HOG 알고리즘과 보행자 검출
- QR 코드 검출

템플릿 매칭

● 템플릿 매칭 기법

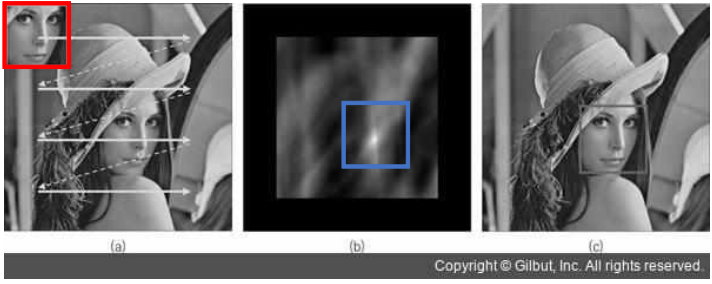
: 입력 영상에서 작은 크기의 부분 영상 위치를 찾아내고 싶은 경우에 사용

●템플릿

: 찾고자 하는 대상이 되는 작은 크기의 영상

●템플릿 매칭

: 작은 크기의 템플릿 영상을 입력 영상 전체 영역에 대해 이동하면서 가장 비슷한 위치를 수치적으로 찾아내는 방식



(a) : 템플릿 영상을 입력 영상 전체 영역에 대해 이동하면서 템플릿 영상과 입력 영상 부분 영상과의 유사도 또는 비유사도 계산

(b) : 입력 영상 모든 위치에서 템플릿 영상과의 유사도를 계산하고, 그 결과를 그레이스케일 영상 형태로 나타난 것

⇒ 밝은 부분 = 가장 유사

(c) : 결과

템플릿 매칭 = matchTemplate() 함수 사용

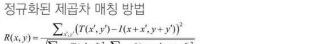
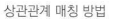
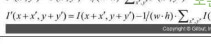
```
void matchTemplate(InputArray image, InputArray templ,
    OutputArray result, int method, InputArray mask = noArray());
```

image	입력 영상. 8비트 또는 32비트 실수형
templ	템플릿 영상. 입력 영상 image보다 같거나 작아야 하며, image와 타입이 같아야 함.
result	(출력) 비교 결과를 저장할 행렬. CV_32FC1 타입
method	템플릿 매칭 비교 방법. TemplateMatchModes 열거형 상수 중 하나를 지정.
mask	찾고자 하는 템플릿의 마스크 영상. mask는 templ과 같은 크기, 같은 타입이어야 함. TM_SQDIFF와 TM_CCORR_NORMED 방법에서만 지원.

- 영상 크기: W x H
- templ 영상 크기: w x h
- result 행렬 크기: (W - w + 1)×(H - h + 1)

- I (x, y): 입력 영상
- T(x, y): 템플릿 영상
- R(x, y): 비교 결과 행렬을 의미

TemplateMatchModes 열거형 상수

TemplateMatchModes 열거형 상수	설명
TM_SQDIFF	제곱차 매칭 방법 : 두 영상이 완벽하게 일치하면 0,서로 유사하지 않으면 0보다 큰 양수 ⇒result 결과 행렬에서 최솟값 위치를 가장 매칭이 잘 된 위치로 선택해야 함 
TM_SQDIFF_NORMED	정규화된 제곱차 매칭 방법 
TM_CCORR	상관관계 매칭 방법 : 두 영상이 유사하면 큰 양수, 유사하지 않으면 작은 값 
TM_CCORR_NORMED	정규화된 상관관계 매칭 방법 
TM_CCOEFF	상관계수 매칭 방법 : 비교할 두 영상 미리 평균 밝기로 보정한 후 상관관계 매칭 수행 : 두 비교 영상이 유사하면 큰 양수, 유사하지 않으면 0에 가까운 양수 또는 음수 
TM_CCOEFF_NORMED	정규화된 상관계수 매칭 방법 

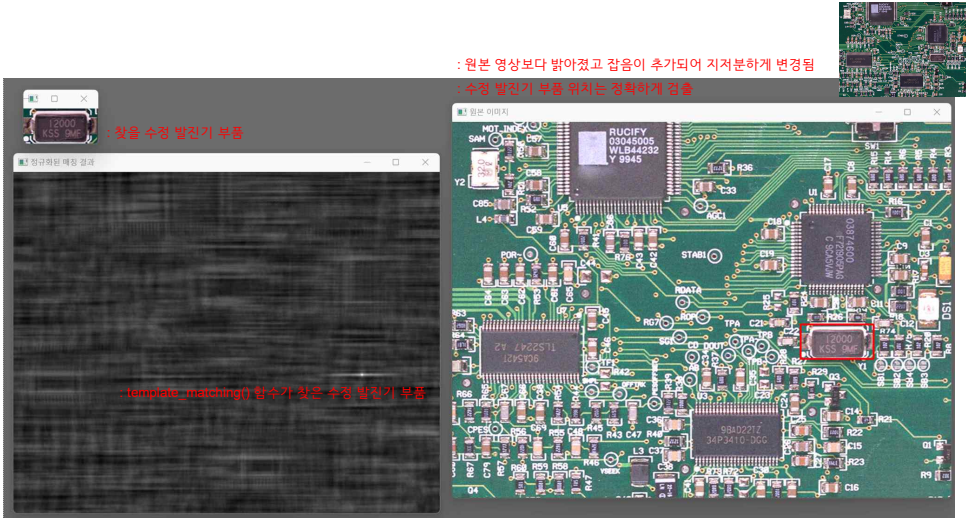
- _NORMED: 각각 영상의 밝기 차이 영향을 줄여 주는 정규화 수식이 추가됨

- result 행렬에서 최솟값/최댓값 위치 : OpenCV의 minMaxLoc() 함수로 쉽게 알아낼 수 있음

result 결과 행렬에서 최댓값 위치가 가장 매칭이 잘 된 위치

템플릿 매칭 예제

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

using namespace cv;
using namespace std;

// 템플릿 매칭을 수행하는 함수 선언
void template_matching();

int main()
{
    template_matching();
    return 0;
}

void template_matching()
{
    // 이미지와 템플릿 이미지 로드
    Mat img = imread("circuit.bmp", IMREAD_COLOR);
    Mat templ = imread("crystal.bmp", IMREAD_COLOR);

    // 이미지나 템플릿 이미지가 로드되지 않았을 경우 에러 출력 후 종료
    if (img.empty() || templ.empty()) {
        cerr << "이미지를 불러오는 데 실패했습니다!" << endl;
        return;
    }

   //실제 영상 획득 과정에서 발생할 수 있는 잡음과 조명의 영향을 시뮬레이션하기 위해 입력 영상의 밝기를 50만큼 증가,표준 편차가 10인 가우시안 잡음을 추가한 후 템플릿 매칭을 수행

    // 이미지의 픽셀값을 밝게 조정
    img = img + Scalar(50, 50, 50);

    // 이미지에 노이즈 추가 :실제 센서의 불완전성이나 전기적 간섭과 같은 요인으로 발생할 수 있는 노이즈 효과를 모방
    Mat noise(img.size(), CV_32SC3);
    randn(noise, 0, 10);
    add(img, noise, img, Mat(), CV_8UC3);

    // 템플릿 매칭 결과 저장할 변수 및 매칭 수행
    Mat res, res_norm;
    matchTemplate(img, templ, res, TM_CCOEFF_NORMED);

    // 매칭 결과를 0~255 범위로 정규화하여 저장
    normalize(res, res_norm, 0, 255, NORM_MINMAX, CV_8U);

    // 매칭 결과에서 최대 값과 위치 찾기
    double maxv;
    Point maxloc;
    minMaxLoc(res, 0, &maxv, 0, &maxloc);
    cout << "최대값: " << maxv << endl;

    // 매칭된 영역에 사각형 그리기
    rectangle(img, Rect(maxloc.x, maxloc.y, templ.cols, templ.rows), Scalar(0, 0, 255), 2);

    // 템플릿 이미지와 매칭 결과, 원본 이미지를 창에 출력
    imshow("템플릿 이미지", templ);
    imshow("정규화된 매칭 결과", res_norm);
    imshow("원본 이미지", img);

    // 키 입력 대기 후 창 닫기
    waitKey();
    destroyAllWindows();
}
```



콘솔 창에는 “maxv: 0.976276” 문자열 출력

⇒ 템플릿 매칭으로 검출된 위치에서 정규화된 상관계수 값, 이 값이 1에 가까운 실수 → 매칭이 잘 되었다고 가늠할 수 있음

캐스케이드 분류기와 얼굴 검출

:OpenCV의 얼굴 검출 기능

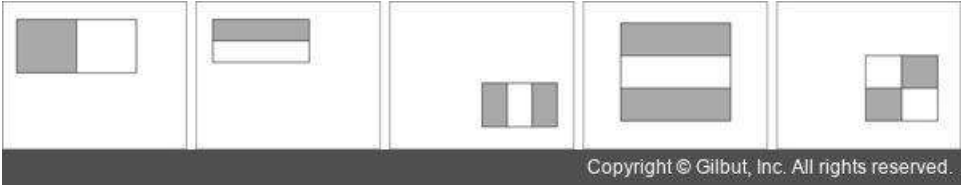
● OpenCV에서 제공하는 얼굴 검출 기능

: 2001년 비올라와 존스가 발표한 부스팅 기반의 캐스케이드 분류기 알고리즘을 기반으로 만들어짐

● 객체 검출 알고리즘

: 다양한 객체를 검출할 수 있지만, 특히 얼굴 검출에 적용되어 속도와 정확도를 인정받은 기술

비올라-존스 얼굴 검출 알고리즘



: 기본적으로 영상을 24×24 크기로 정규화한 후, 유사-하르 필터(Haar-like filter) 집합으로부터 특징 정보를 추출하여 얼굴 여부를 판별

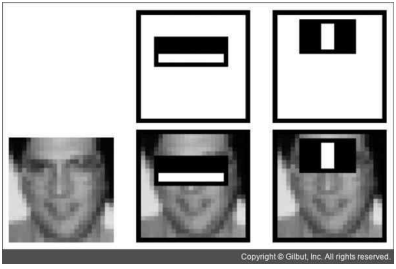
- 유사-하르 필터: 흑백 사각형이 서로 붙어 있는 형태로 구성된 필터.

⇒ 유사-하르 필터 형태에서 흰색 영역 픽셀 값은 모두 더하고, 검은색 영역 픽셀 값은 모두 빼서 하나의 특징 값을 얻을 수 있음

⇒ 사람의 정면 얼굴 형태가 전형적으로 밝은 영역(이마, 미간, 볼 등)과 어두운 영역(눈썹, 입술 등)이 정해져 있기 때문에 유사-하르 필터로 구한 특징 값은 얼굴을 판별하는 용도로 사용할 수 있음

*시간이 오래 걸린다는 문제

에이다부스트알고리즘과 적분 영상



- 에이다부스트 알고리즘

: 수많은 유사-하르 필터 중에서 얼굴 검출에 효과적인 필터를 선별하는 역할

: 24×24 부분 영상에서 검사할 특징 개수가 약 6000개로 감소, 입력 영상 전체에서 부분 영상을 추출하여 검사해야 함 = 여전히 연산량 부담

: 더군다나 나타날 수 있는 얼굴 크기가 다양하기 때문에 보통 입력 영상의 크기를 줄여 가면서 전체 영역에 대한 검사를 다시 수행해야 함

⇒ 대부분의 영상에 얼굴이 한두 개 있을 뿐이고 나머지 대부분의 영역은 얼굴이 아니라는 점에 주목

⇒ 비올라-존스 알고리즘에서는 캐스케이드 구조라는 새로운 방식을 도입하여 얼굴이 아닌 영역을 빠르게 걸러 내는 방식을 사용

캐스케이드 구조

- **1단계**: 얼굴 검출에 가장 유용한 유사-하르 필터 하나를 사용하여, 얼굴이 아니라고 판단 → 이후의 유사-하르 필터 계산은 수행 X
- 1단계를 통과하면 **2단계** : 유사-하르 필터 다섯 개를 사용하여 얼굴이 아닌지를 검사
- 얼굴이 아니라고 판단 → 이후 단계의 검사는 수행 X

∴ 얼굴이 아닌 영역을 빠르게 제거함

⇒ 비올라-존스 얼굴 검출 알고리즘은 동시대의 다른 얼굴 검출 방식보다 약 15배 빠르게 동작하는 성능을 보여줌

캐스케이드 구조

- **1단계**: 얼굴 검출에 가장 유용한 유사-하르 필터 하나를 사용하여, 얼굴이 아니라고 판단 → 이후의 유사-하르 필터 계산은 수행 X
- 1단계를 통과하면 **2단계** : 유사-하르 필터 다섯 개를 사용하여 얼굴이 아닌지를 검사
- 얼굴이 아니라고 판단 → 이후 단계의 검사는 수행 X

∴ 얼굴이 아닌 영역을 빠르게 제거함

⇒ 비올라-존스 얼굴 검출 알고리즘은 동시대의 다른 얼굴 검출 방식보다 약 15배 빠르게 동작하는 성능을 보여줌

CascadeClassifier 클래스를 이용하여 객체를 검출

CascadeClassifier 객체 생성

: 단순히 CascadeClassifier 클래스 타입의 변수를 하나 선언하는 방식으로 생성

```
CascadeClassifier classifier; //CascadeClassifier 타입의 객체 classifier를 선언
```

CascadeClassifier 객체 생성 후

미리 훈련된 **분류기 정보**(XML 파일 형식으로 저장)를 불러올 수 있음

분류기 XML 파일 불러오기 = CascadeClassifier::load() 함수

```
void CascadeClassifier::load(const String& filename); //filename: 불러올 분류기 XML 파일 이름
```

- 불러올 XML 파일이 프로그램 실행 폴더에 있다면 파일 이름만 CascadeClassifier::load() 함수 인자로 전달
- XML 파일이 다른 폴더에 있다면 상대 경로 또는 절대 경로 형태의 문자열을 filename 인자로 전달

  ex) C 드라이브 최상위 폴더에 있는 face.xml 파일을 불러오기 → filename 인자에 "C:\\face.xml" 문자열 전달

* OpenCV는 미리 훈련된 얼굴 검출, 눈 검출 등을 위한 분류기 XML 파일을 제공 → C:\opencv\build\etc\haarcascades

OpenCV에서 제공하는 하르 기반 분류기 XML 파일

XML 파일 이름	검출 대상
haarcascade_frontalface_default.xml haarcascade_frontalface_alt.xml haarcascade_frontalface_alt2.xml haarcascade_frontalface_alt_tree.xml	정면 얼굴 검출
haarcascade_profileface.xml	측면 얼굴 검출
haarcascade_smile.xml	웃음 검출
haarcascade_eye.xml haarcascade_eye_tree_eyeglasses.xml haarcascade_lefteye_2splits.xml haarcascade_righteye_2splits.xml	눈 검출
haarcascade_frontalcatface.xml haarcascade_frontalcatface_extended.xml	고양이 얼굴 검출
haarcascade_fullbody.xml	사람의 전신 검출
haarcascade_upperbody.xml	사람의 상반신 검출
haarcascade_lowerbody.xml	사람의 하반신 검출
haarcascade_russian_plate_number.xml haarcascade_licence_plate_rus_16stages.xml	러시아 자동차 번호판 검출

```
//CascadeClassifier 객체를 생성한 후,
CascadeClassifier classifier;

//CascadeClassifier::load() 함수를 이용하여 정면 얼굴 검출을 위한 XML 파일을 불러오기
classifier.load("haarcascade_frontalface_default.xml");
```

```
//CascadeClassifier 클래스는 객체 생성과 동시에 XML 파일을 불러올 수 있는 생성자 제공 = 한 줄 가능
CascadeClassifier classifier("haarcascade_frontalface_default.xml");
```

* haarcascade_frontalface_default.xml 파일 = 프로그램 실행 시 프로그램과 같은 폴더에 있어야 함

CascadeClassifier::empty() 멤버 함수

```
bool CascadeClassifier::empty() const //분류기 파일을 정상적으로 불러왔는지를 확인
```

- 반환값 분류기 파일을 정상적으로 불러왔으면 false, 그렇지 않으면 true를 반환
- true 반환할 경우 객체 검출을 수행할 수 없으므로 예외 코드 추가하는 것이 안전

CascadeClassifier::detectMultiScale() 멤버 함

```
void CascadeClassifier::detectMultiScale(
    InputArray image,
    vector<Rect>& objects,
    double scaleFactor = 1.1,
    int minNeighbors = 3, int flags = 0,
    Size minSize = Size(),
    Size maxSize = Size()
);
```

image	입력 영상. CV_8U 깊이의 행렬
objects	(출력) 검출된 객체의 사각형 좌표 정보
scaleFactor	검색 윈도우 확대 비율. 1보다 커야 함
minNeighbors	검출 영역으로 선택하기 위한 최소 검출 횟수
flags	현재 사용 X
minSize	검출할 객체의 최소 크기
maxSize	검출할 객체의 최대 크기

CascadeClassifier::detectMultiScale() 멤버 함

: 입력 영상 image에서 다양한 크기의 객체 사각형 영역을 검출

수

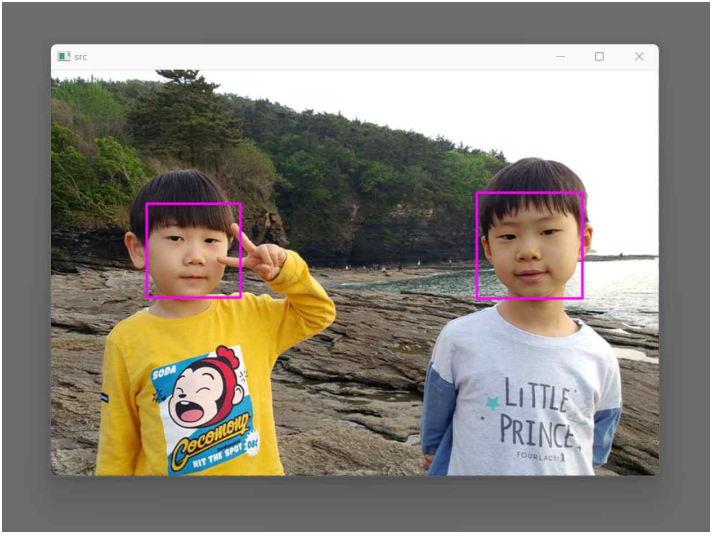
```
void CascadeClassifier::detectMultiScale(
    InputArray image,
    vector<Rect>& objects,
    double scaleFactor = 1.1,
    int minNeighbors = 3, int flags = 0,
    Size minSize = Size(),
    Size maxSize = Size()
);
```

- 만약 입력 영상 image가 3채널 컬러 영상이면
- 함수 내부에서 그레이스케일 형식으로 변환하여 객체를 검출
- 각각의 사각형 영역 정보는 Rect 클래스를 이용하여 표현
- vector<Rect> 타입의 인자 objects에 검출된 모든 사각형 정보가 저장
- scaleFactor 인자는 검색 윈도우의 확대 비율을 지정
- CascadeClassifier::detectMultiScale() 함수는 다양한 크기의 얼굴을 검출하기 위하여 처음에는 작은 크기의 검색 윈도우를 이용하여 객체를 검출
- 이후 scaleFactor 값의 비율로 검색 윈도우 크기를 확대시키면서 여러 번 객체를 검출
- minNeighbors 인자에는 검출할 객체 영역에서 얼마나 많은 사각형이 중복되어 검출되어야 최종적으로 객체 영역으로 설정할지를 지정
- minNeighbors 값을 기본값인 3으로 설정하면 검출된 사각형이 최소 세 개 이상 중첩되어야 최종적으로 객체 영역으로 판단

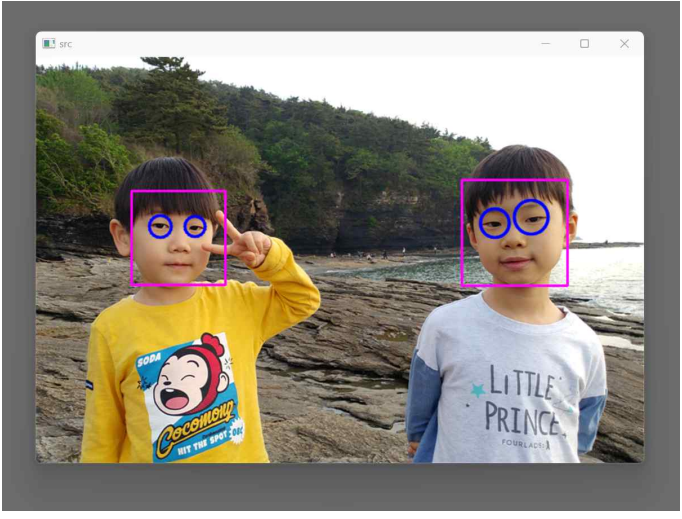
얼굴/눈 검출 예제 프로그램

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

using namespace cv;
using namespace std;

void detect_face();  // 얼굴 검출 함수의 프로토타입 선언
void detect_eyes();  // 눈 검출 함수의 프로토타입 선언

int main()
{
    detect_face();  // 얼굴 검출 함수 호출
    detect_eyes();  // 눈 검출 함수 호출

    return 0;  // 프로그램 종료
}
```

```
void detect_face()
{
    // 이미지 파일을 읽어옵니다.
    Mat src = imread("kids.png");

    if (src.empty()) {
        cerr << "Image load failed!" << endl;  // 이미지 로드 실패 시 오류 메시지 출력
        return;  // 함수 종료
    }

    // 얼굴 검출을 위한 Haar Classifier 객체 생성 및 XML 파일 로드
    CascadeClassifier classifier("haarcascade_frontalface_default.xml");

    if (classifier.empty()) {
        cerr << "XML load failed!" << endl;  // XML 파일 로드 실패 시 오류 메시지 출력
        return;  // 함수 종료
    }

    vector<Rect> faces;  // 얼굴 검출 결과를 저장할 벡터
    classifier.detectMultiScale(src, faces);  // 이미지에서 얼굴 검출

    for (Rect rc : faces) {
        rectangle(src, rc, Scalar(255, 0, 255), 2);  // 검출된 얼굴 주변에 사각형 그리기
    }

    imshow("src", src);  // 결과 이미지를 윈도우에 표시

    waitKey(0);  // 키 입력 대기
    destroyAllWindows();  // 모든 이미지 윈도우 닫기
}
```


src
SODA
LITTLE
PRINCE
FOURLADS

```
void detect_eyes()
{
    Mat src = imread("kids.png");  // 이미지 파일을 읽어옵니다.

    if (src.empty()) {
        cerr << "Image load failed!" << endl;  // 이미지 로드 실패 시 오류 메시지 출력
        return;  // 함수 종료
    }

    CascadeClassifier face_classifier("haarcascade_frontalface_default.xml");  // 얼굴 검출을 위한 Haar Classifier 객체 생성 및 XML 파일 로드
    CascadeClassifier eye_classifier("haarcascade_eye.xml");  // 눈 검출을 위한 Haar Classifier 객체 생성 및 XML 파일 로드

    if (face_classifier.empty() || eye_classifier.empty()) {
        cerr << "XML load failed!" << endl;  // XML 파일 로드 실패 시 오류 메시지 출력
        return;  // 함수 종료
    }

    vector<Rect> faces;  // 얼굴 검출 결과를 저장할 벡터
    face_classifier.detectMultiScale(src, faces);  // 이미지에서 얼굴 검출

    for (Rect face : faces) {
        rectangle(src, face, Scalar(255, 0, 255), 2);  // 검출된 얼굴 주변에 사각형 그리기

        Mat faceROI = src(face);  // 얼굴 영역 추출
        vector<Rect> eyes;  // 눈 검출 결과를 저장할 벡터
        eye_classifier.detectMultiScale(faceROI, eyes);  // 얼굴 영역에서 눈 검출

        for (Rect eye : eyes) {
            Point center(eye.x + eye.width / 2, eye.y + eye.height / 2);  // 눈 중심 계산
            circle(faceROI, center, eye.width / 2, Scalar(255, 0, 0), 2, LINE_AA);  // 눈 중심에 원 그리기
        }
    }

    imshow("src", src);  // 결과 이미지를 윈도우에 표시

    waitKey(0);  // 키 입력 대기
    destroyAllWindows();  // 모든 이미지 윈도우 닫기
}
```


src
SODA
LITTLE
PRINCE
FOUR ACES

객체 검출에 관한 Open CV 실습

3. HOG 알고리즘과 보행자 검출

4. QR 코드 검출

HOG

- Histogram of Oriented Gradients
- 영상의 지역적 그래디언트 방향 정보를 히스토그램으로 표현해서 영상의 형태를 표현하는 방법
- HOG와 SVM 머신러닝을 결합하여 정형화된 객체를 검출하는 알고리즘

HOG 알고리즘

- Open CV에서 기본으로 제공되는 알고리즘
- 이미지 혹은 영상 속 프레임에서 인물(보행자)을 검출하기 위한 목적
- 다만, 검출 성능이 매우 좋지는 않음

HOG 알고리즘 작동 순서

1. 임의의 크기의 사각형을 정의해서 부분 영상을 추출합니다.

2. 추출한 부분 영상의 크기를 정규화 합니다. (64X128)

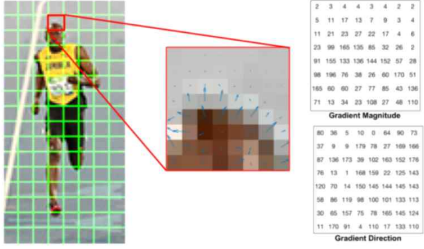
3. 64X128 영상의 그래디언트를 계산하여 방향 성분과 크기 성분을 파악합니다.

HOG 알고리즘 작동 순서

4. 64X128 영상을 8X8 크기의 셀(cell)로 분할합니다.

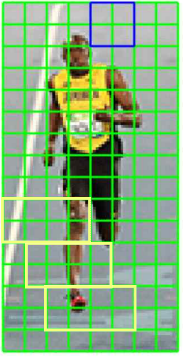
5. 각 셀마다 방향과 크기 성분을 이용하여 방향 히스토그램을 계산합니다.

6. 각각의 셀에서 방향 성분을 9개로 구분하여 9가지 방향에 대한 히스토그램을 생성합니다. (180도를 20도씩 9가지 방향, 대칭하면 360도)



블록 히스토그램 구하기

- 8X8 셀 4개를 하나의 블록으로 지정 => 블록 하나의 크기는 16X16
- 8픽셀 단위 로 이동 (블록 반칸씩 겹쳐서 이동)
- 각 블록의 히스토그램 빈(bin) 개수는 4X9 = 36개 (방향 성분 조합 36가지)
- 하나의 부분 영상 패치에서의 특징 백터 크기는 7 X 15 X 36 = 3780
- 여기에 특징 벡터이므로 또 4를 곱함



| HOG 알고리즘 구현 함수

- OpenCV는 HOG 알고리즘을 구현한 HOGDescriptor 클래스를 제공
- HOGDescriptor 클래스를 이용하면 특정 객체의 HOG 기술자를 쉽게 구현할 수 있다.
- HOGDescriptor 클래스를 이용하려면 먼저 HOGDescriptor 객체를 생성하여야 한다.

  이때, 객체를 생성하기 위해 기본 생성자를 이용하면 되는데, 이 기본 생성자는
  윈도우 검색 크기를 64 x 128로 설정하고, 셀 크기는8 x 8, 블록 크기는 16 x 16,
  그래디언트 방향 히스토그램 빈 개수는 9개로 설정된다.

- 다음 장에서 HOG 알고리즘 구현 함수 세가지 설명

| HOGDescriptor::detecMultiScale() 함수

- 입력 영상img 에서 다양한 크기의 객체 사각형 영역을 검출하고,
  그 결과를 std::vector<Rect> 타입의 인자 foundLocations에 저장한다.

| HOGDescriptor::getDefaultPeopleDetector() 함수

- 64 x 128 크기의 윈도우에서 똑바로 서 있는 사람(보행자)을 검출하는 용도로 훈련된 계수기를 반환한다.

| HOGDescriptor::setSVMDetector() 함수

- HOGDescriptor 클래스를 이용하여 원하는 객체를 검출하기 위해서
  먼저 검출할 객체에 대해 훈련된 SVM분류기 계수를 이 함수에 등록해야 한다.

>> 보행자 검출을 할 경우 HOGDescriptor::getDefaultPeopleDetector() 함수가 반환한 분류기 계수를 HOGDescriptor::setSVMDetector() 함수의 인자로 전달하면 된다.

| 실습코드 1번 (.jpg)

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

using namespace std;
using namespace cv;

int main()
{
    VideoCapture cap("C:/opencv/ch02/Project1/people.JPG");

    if (!cap.isOpened()) {
        cout << "Video Open Fail" << endl;
        return -1;
    }

    HOGDescriptor hog;    // HOGDescriptor 객체 hog 선언
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector()); // 보행자 검출을 위한 미리정의된 SVM 분류기 계수 등록

    Mat frame;
    while(1) {
        cap >> frame;
        if (frame.empty())
            break;

        vector<Rect> detected;
        hog.detectMultiScale(frame, detected);    // 보행자 검출 수행

        for (Rect r : detected) {                 // 검출된 정보를 사각형으로 표시
            Scalar c = Scalar(255, 0, 0);
            rectangle(frame, r, c, 3);
        }

        imshow("frame", frame);
        waitKey();

    }
}
```

실습코드 2번 (.mp4)

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

using namespace cv;
using namespace std;

int main()
{
    VideoCapture cap("vtest.avi");

    if (!cap.isOpened()) {
        cerr << "Video open failed!" << endl;
        return -1;
    }

    HOGDescriptor hog;
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

    Mat frame;
    while (true) {
        cap >> frame;
        if (frame.empty())
            break;

        vector<Rect> detected;
        hog.detectMultiScale(frame, detected);

        for (Rect r : detected) {
            Scalar c = Scalar(rand() % 256, rand() % 256, rand() % 256);
            rectangle(frame, r, c, 3);
        }

        imshow("frame", frame);

        if (waitKey(10) == 27)
            break;
    }

    return 0;
}
```

실습코드 3번 (.avi)

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

using namespace cv;
using namespace std;

int main()
{
    VideoCapture cap("vtest.avi");

    if (!cap.isOpened()) {
        cerr << "Video open failed!" << endl;
        return -1;
    }

    HOGDescriptor hog;
    hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());

    Mat frame;
    while (true) {
        cap >> frame;
        if (frame.empty())
            break;

        vector<Rect> detected;
        hog.detectMultiScale(frame, detected);

        for (Rect r : detected) {
            Scalar c = Scalar(rand() % 256, rand() % 256, rand() % 256);
            rectangle(frame, r, c, 3);
        }

        imshow("frame", frame);

        if (waitKey(10) == 27)
            break;
    }

    return 0;
}
```

객체 검출에 관한 Open CV 실습

3. HOG 알고리즘과 보행자 검출

4. QR 코드 검출

QR코드 인식 기술

QR코드 기능을 이용하기 위해서는 먼저 카메라가 QR코드를 인식하고 검출해야 한다.

QR코드 검출 방법

1. 먼저 QR 코드 세 모서리에 포함된 흑백 정사각형 패턴을 찾는다.

2. QR 코드 전체 영역 위치를 알아낸다.

3. 검출된 QR 코드를 정사각형 형태로 투시 변환한다.

4. QR 코드 내부에 포함된 흑백 격자 무늬를 해석하여 문자열을 추출한다.

>> 위 작업은 매우 복잡하고 정교한 영상 처리를 필요로 하는데,
OpenCV 4.0.0 버전부터 QR 코드를 검추랗고 해석하는 기능을 제공한다.

QR코드 인식 함수

- OpenCV에서 QR 코드를 검출하고 해석하는 기능은 QRCodeDetector 클래스에 구현되어 있다.
- QRCodeDetector 클래스를 이용하여 영상에서 QR 코드를 검출하거나 해석하려면 먼저 QRCodeDetector 객체를 생성해야 한다.
- QRCode 객체를 생성한 후 QRCodeDetector 클래스 멤버 함수를 이용하여 QR 코드를 검출하거나 문자열을 해석할 수 있다.
- 다음 장에서 HOG 알고리즘 구현 함수 세가지 설명

QRCodeDetector::detect() 함수

- 입력 영상 img에서 QR 코드를 검출하고, QR 코드를 감싸는 사각형의 꼭지점 좌표를 반환

QRCodeDetector::detect() 함수

- 입력 영상 img에서 QR 코드의 검출과 해석을 동시에 수행하고, 해석된 문자열을 반환

실습코드 4번 (*노트북 내장 카메라 사용)

```
void decode_qrcode()
{
    VideoCapture cap(0);

    if (!cap.isOpened()) {
        cerr << "Camera open failed!" << endl;
        return;
    }

    QRCodeDetector detector;

    Mat frame;
    while (true) {
        cap >> frame;

        if (frame.empty()) {
            cerr << "Frame load failed!" << endl;
            break;
        }

        vector<Point> points;
        String info = detector.detectAndDecode(frame, points);

        if (!info.empty()) {
            polylines(frame, points, true, Scalar(0, 0, 255), 2);
              putText(frame, info, Point(10, 30), FONT_HERSHEY_DUPLEX, 1, Scalar(0, 0, 25

        }

        imshow("frame", frame);
        if (waitKey(1) = = 27)
            break;
    }
}
```

끝