연 + 재 + 순 + 서
1회 2004.12 | OpenCV 영상처리 프로그래밍의 첫걸음
2회 | OpenCV 고급 영상처리 알고리즘 구현

연 + 재 + 가 + 이 + 드
운영체제 | 윈도우 98/ME/NT/2000/XP
개발도구 | 다이나루스 비수얼 스튜디오 6
기초지식 | C, 비수얼 스튜디오, 영상처리에 대한 기초 개념

이흔진 | hjlee@vision.hanyang.ac.kr
황미란 | mirang@vision.hanyang.ac.kr

한양대학교 영상공학연구실의 동기로 각각 컴퓨터 인식, 위치 파악 등을 연구하였다. 지금은 사회 초년생으로서 열심히 적응 중이며 짬짬이 시간을 내어 함께 공부를 하곤 한다.

OpenCV 라이브러리를 이용한 영상처리 1

쉽고 빠른 OpenCV 영상처리의 첫걸음

우리는 일상의 대부분을 영상처리와 함께 한다. TV를 보고, 사진을 찍고, PC를 사용하는 등 이제는 일상에서 영상처리를 빼놓을 수 없다. 영상처리와 전혀 관계없어 보인다면 다시 한번 살펴보자. 디지털 방송에서 TV 화면 신호는 압축되어 전송되고, 전송된 화면은 복원 뒤 선명한 화질을 보여주기 위해 내부적으로 처리가 된다. 이런 영상 신호의 압축, 화질 개선이 모두 영상처리가 하는 일이다.

또한 디지털 카메라 내부에서도 화질 개선, 노이즈 제거 등을 위해서 영상처리를 하고 있으며, 포토샵을 이용한 보정 작업은 영상처리의 전형이다. 최신의 인텔 기반 노트북은 배터리 사용 시간을 늘리기 위해 DPST(Display Power Saving Technology)라는 영상처리 기술을 구사하기도 한다(참고자료 ❶).

영상처리 라이브러리로서 인텔 OpenCV가 있다. 인터넷에 공개된 다른 영상처리 라이브러리에 비해 체계적으로 개발되어 왔으며 기본적인 영상을 읽고 쓰는 기능부터 고급 영상처리 알고리즘까지 제공한다. 그 동안 OpenCV 라이브러리가 낯설고 사용법이 어려워서 활용하지 못하고 있었다면 이번 기회에 OpenCV 라이브러리를 정복해 보자.

<화면 1> OpenCV를 이용한 영상처리

OpenCV 라이브러리

이렇게 영상처리가 점점 더 생활 속으로 파고들고 있지만, 막상 프로그램으로 영상처리 알고리즘을 개발하려고 하면 장벽이 생긴다. 영상처리 이론을 이해하는 것도 일이지만, 그것을 구현하는 과정 또한 쉽지 않기 때문이다. 이러한 어려움을 해결해 줄 수 있는 것이 바로 인텔의 OpenCV 라이브러리(Open Source Computer Vision Library)다.

OpenCV는 인텔 주도 하에 만들어지는 영상처리 전용 라이브러리다. OpenCV에서 제공하는 알고리즘은 〈표 1〉과 같이 영상 읽기, 출력 등 간단한 기능에서부터 옵티컬 플로우(optical flow), 임베디드 HMM, 카메라 캘리브레이션(camera calibration) 등 고급 알고리즘까지 폭넓게 제공한다. OpenCV를 이용하면 〈화면 1〉과 같이 간단한 이미지 툴에서부터 패턴 인식, 3D 그래픽스 시스템을 구현할 수 있다.

<표 1> OpenCV에서 지원하는 영상처리 알고리즘의 종류

Image functions	Data Structures	Contour Processing	Geometry
Features	Image Statistics	Image Pyramids	Morphology
Background Differencing	Distance Transform	Thresholding	Flood Fill
Camera Calibration	View Morphing	Motion Templates	CAMSHIFT
Active Contours	Optical Flow	Estimators	POSIT
Histogram (recognition)	Gesture Recognition	Matrix	Eigen Objects
Embedded HMMs	Drawing Primatives	System Functions	Utility

다양한 알고리즘을 제공한다는 장점 외에도 OpenCV 라이브러리의 경우 인텔 CPU에 최적화되어 있기 때문에 빠른 실행 속도를 보여준다. 또한 OpenCV의 알고리즘에 대한 소스를 모두 제공하고 있어 사용자의 입맛에 맞게 수정해 사용할 수 있다.

OpenCV는 리눅스처럼 세계 각지의 개발자들에 의해 만들어지고 있으며 OpenCV의 코딩 규칙을 지킨다면 자신이 만든 알고리즘을 OpenCV의 알고리즘에 추가할 수도 있다(참고자료 ❷). OpenCV 라이브러리를 사용하는 경우 인텔의 라이선스 조항을 따라야 하지만 대부분 자유롭게 사용할 수 있다(참고자료 ❸).

OpenCV에 관한 좀 더 자세한 내용은 웹 사이트에서 확인할 수 있다(참고자료 ❹). 여기까지 OpenCV에 대한 소개는 간단히 마치며 다음에서는 OpenCV를 설치하고 영상처리 알고리즘을 직접 실행해 보자.

OpenCV 설치하기

우선, 인텔 사이트를 통해 OpenCV 소스를 다운 받자. 그러나 현재 인텔 사이트에서는 〈화면 2〉와 같이 OpenCV에 대한 소개만 하고 있

<화면 2> 인텔 OpenCV 소개 사이트

으며 소스를 받기 위해서는 소스포지 사이트에 접속해야 한다(http://sourceforge.net/projects/opencvlibrary/). 인텔 사이트에 링크가 제공되므로 클릭하여 소스포지로 이동하자. 참고로 OpenCV 웹 그룹(http://www.yahoogroups.com/group/OpenCV)도 있어서 관련된 정보를 얻을 수 있다.

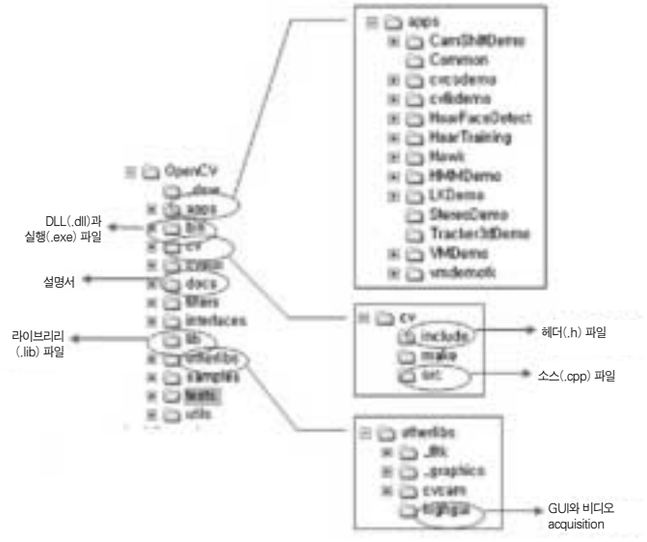
윈도우용 OpenCV 베타 4 버전(OpenCV-win)을 선택하여 다운받아 설치한다. 특별한 설정은 없으며 'Next' 버튼을 열심히 눌러서 설치를 마치면 된다. 설치가 끝나면 Program Files 폴더 밑에 OpenCV 폴더가 설치된다.

〈그림 1〉과 〈그림 2〉는 각각 OpenCV 베타 3.1과 베타 4의 폴더 구성을 보여준다. 최근에 OpenCV 베타 4로 버전업됐는데, OpenCV가 베타 3.1에서 베타 4로 버전업되면서 크게 바뀐 부분은 바로 cv.h 중에서 기본적인 구조체(structure)와 함수가 cxcore.h로 독립되어 나왔다는 것이다. 따라서 기존의 OpenCV 베타 3.1을 사용했다면 cxcore.h, cxcore.lib, cxcore.dll과 설정을 추가해야 한다. 또한, 예제 부분이 조금 달라졌다. 베타 3.1에서는 apps 폴더 밑에 예제 프로젝트가 많이 있었으나 베타 4에서는 사라졌다. 베타 3.1의 apps 폴더 밑에 유용한 예제가 많이 있으니 설치해서 실행해 보길 권한다.

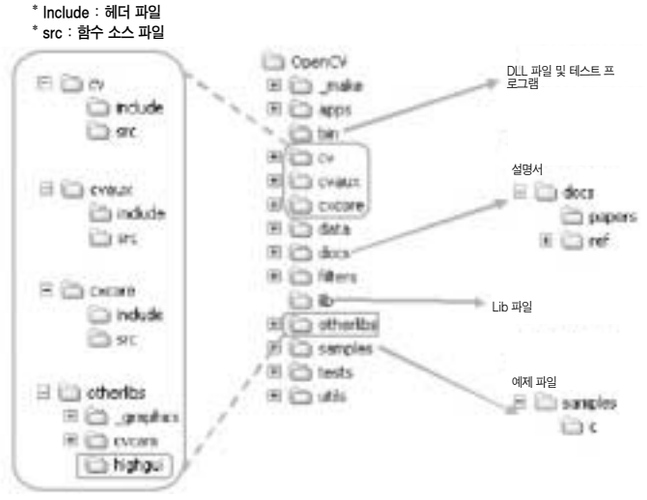
docs는 도움말 파일이 있는 폴더로 문서들을 한 번씩은 읽어봐야 하고, samples 밑의 예제들도 한 번씩은 꼭 실행해 보길 바란다. 자세한 것은 차차 설명하기로 하고 나머지 폴더의 설명은 〈그림 1〉과 〈그림 2〉를 참고하길 바란다.

참고로 OpenCV 라이브러리의 실제 소스들은 OpenCV/cv/src와 같이 src 폴더들에서 찾아볼 수 있다.

<그림 1> OpenCV 3.1 베타 라이브러리의 폴더 구성

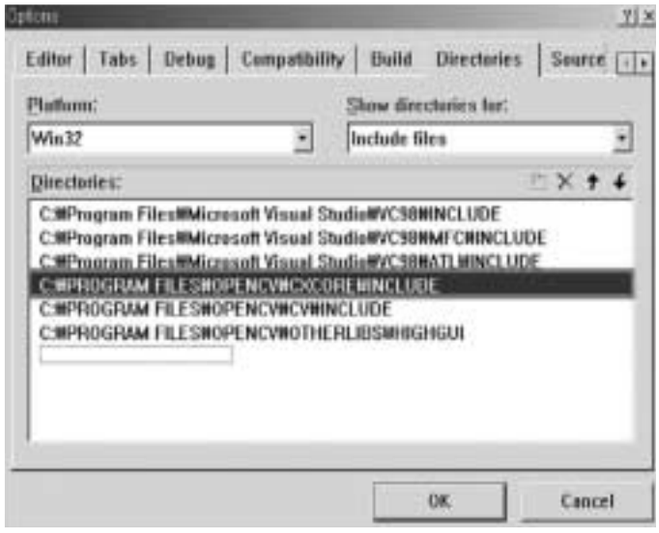


<그림 2> OpenCV 4 베타 라이브러리의 폴더 구성



다른 라이브러리와 마찬가지로 OpenCV도 헤더 파일과 라이브러리 파일의 경로를 비주얼 스튜디오에 포함시켜야만 사용할 수 있다.

비주얼 스튜디오 6.0의 「Tools | Option | Directories」 탭을 이용하여 라이브러리를 추가한다. 즉, Include files 항목에서 OPENCV\CXCORE\INCLUDE' OpenCV/cv/include와 OpenCV/OTHER LIBS/HIGHGUI, OPENCV\CXCORE\INCLUDE의 경로를 추가한다. Library files 항목에서는 OpenCV/lib 경로를 추가한다.



<화면 3> OpenCV 4 베타에서 헤더 폴더 포함시키기

Option 항목에서 소스와 라이브러리의 경로를 설정하면 기본적인 비주얼 스튜디오 환경 설정은 끝난다. OpenCV의 lib 파일과 dll 파일을 연결하는 과정은 비주얼 스튜디오에서 직접 프로그램을 작성해 보면서 알아보자.

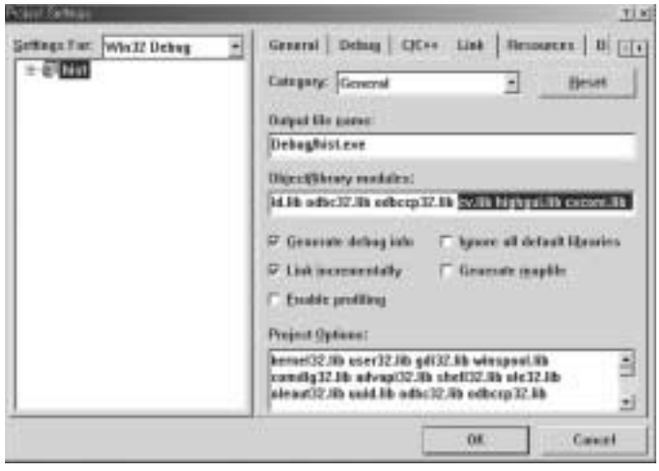
OpenCV를 이용한 영상 출력

우선 비주얼 C++ 6.0을 실행하고 Win32 Console Application 모드로 새로운 프로젝트를 만든다. 새로운 파일을 프로젝트에 추가한 다음 〈리스트 1〉의 코드를 작성하자. 〈리스트 1〉은 image.jpg를 읽어 화면에 출력해 준다.

〈리스트 1〉 영상 출력 프로그램

```
#include "cv.h"
#include "highgui.h"

void main()
{
   IplImage* image;
   image = cvLoadImage( "image.jpg", -1); // 이미지 읽기

   if( image != 0 )
   {
      cvNamedWindow( "test", 0 );     // 이름이 "test"인 윈도우 생성
      cvShowImage( "test", image );   // "Test" 윈도우에 이미지 출력
      cvWaitKey(0);                   // 키보드 입력 기다림
      cvReleaseImage( &image );       // 이미지에 할당된 리소스 해제
   }
}
```

코드를 입력하고 컴파일하면 라이브러리 관련 에러 메시지를 볼 수 있을 것이다. 이것은 OpenCV 라이브러리의 경로는 설정했지만, 실제로 OpenCV 라이브러리를 프로젝트에 포함시키지 않았기 때문에 나타나는 에러다. 현재 프로그램에서는 cxcore.lib, cv.lib, highgui.lib 등 3개의 라이브러리가 필요하다. 라이브러리 추가를 위해서는 〈화면 4〉와 같이 Project Setting의 Link 탭에서 앞의 lib 파일들을 추가한다.



〈화면 4〉 cv.lib과 highgui.lib 라이브러리를 프로젝트에 추가시키는 방법

그리고 cxcore.lib, cv.lib, highgui.lib 등 3가지 라이브러리를 사용하기 때문에 cxcore.dll과 cv.dll, highgui.dll이 있어야 한다. 단, OpenCV 베타 4로 버전업되면서 cxcore096.dll, cv096.dll, highgui096.dll이란 이름으로 변경됐다. 윈도우 프로그램이 실행될 때에는 보통 *.dll 파일을 실행 파일이 있는 폴더에서 찾고, 없으면 Windows\System32 폴더를 찾기 때문에 두 폴더 중 한 곳에는 cxcore096.dll, cv096.dll, highgui096.dll이 있어야 한다. dll 파일들은 OpenCV\bin 폴더에서 찾을 수 있으며 해당 프로젝트의 프로그램이 실행되는 폴더(Debug 혹은 Release 폴더)에 복사하거나 Windows\System32 폴더에 복사하도록 한다.

여기까지 세팅이 끝나면 기본적인 OpenCV 라이브러리에 관한 설정은 끝이 난다. 추후에 OpenCV에서 사용하는 다른 라이브러리를 사용할 경우가 생긴다면 앞에서와 같이 설정하면 된다. 이제 프로그램을 실행하면 image.jpg라는 그림 파일을 〈화면 5〉와 같이 화면에 출력한다.

〈화면 5〉 OpenCV를 이용한 영상 출력하기

〈리스트 1〉과 같이 OpenCV 라이브러리를 이용하면 원하는 영상을 쉽게 화면에 출력할 수 있다. 무엇보다도 MFC나 윈도우 프로그래밍 방법을 알지 못해도 얼마든지 영상을 화면에 출력하고, 그래픽 관련 작업을 할 수 있는 것이 OpenCV의 장점이다. 〈리스트 1〉에서 나온 구조체와 함수 등에 대해 간단히 살펴보자.

IplImage는 OpenCV에서 영상의 정보를 포함하고 있는 구조체다. 영상처리에서 영상을 빼고 생각할 수 없듯이 OpenCV에서도 IplImage를 뺄 수 없을 만큼 필수적인 자료형이다. 영상을 다루는 자료형으로서 중요 인자로는 영상의 크기(width, height)와 색상 정보(nChannels=1,2,3,4), 데이터 크기(depth), 영상 크기(imageSize), 영상 데이터(imageData)가 있다. 특히, 이미지 데이터를 직접 다루려면 imageData라는 원소를 이용해야 한다. 단, OpenCV에서는 픽셀 값을 2차원 배열 대신 1차원 imageData 배열로 다루고 있으며 다음 호에서 설명하겠지만 영상 픽셀 값을 다룰 때에는 width 대신에 widthStep을 사용하는 것에 주의해야 한다.

그리고 OpenCV에서 윈도우 프로그래밍에 대한 특별한 지식 없이도 영상처리를 할 수 있도록 해주는 것이 바로 HighGUI라는 라이브러리다. 〈리스트 1〉에서 나온 cvLoadImage()와 cvReleaseImage(), cvNamedWindow(), cvShowImage()는 모두 HighGUI에서 제공하는 함수들로, 사용하기 위해서는 highgui.h를 소스에 포함시켜야 한다. HighGUI를 이용하면 영상을 읽는 복잡한 과정도 cvLoadImage()라는 함수 한 줄로 대체할 수 있다. 현재 BMP, JPG, PNG, PBM, TIF 등의 영상 파일을 지원한다. 기능을 간단히 살펴보면, cvLoadImage()의 두 번째 인자는 영상을 읽는 방식으로 -1이면 영상 포맷 그대로 읽으며, 0이면 그레이 영상으로, 1이면 컬러 영상으로 변환해서 읽어온다. cvNamedWindow()는 새로운 윈도우를 만드는 기능을, cvShowImage()는 영상을 출력하는 기능을 한다. 그리고 cvLoadImage()로 읽어온 영상은 반드시 cvReleaseImage()를 해줘야만 한다.

OpenCV를 이용한 동영상 출력

OpenCV를 이용하면 동영상 재생이나 PC 카메라로부터 받은 영상을 다룰 수 있다. 〈화면 6〉은 각각 su.avi 동영상을 재생하고, PC 카메라로부터 받은 영상을 출력하는 예제다. 동영상을 제어하기 위해서는 MFC에서 제공하는 AVI 관련 함수나 DirectShow를 익혀야 하지만 OpenCV에서는 〈리스트 2〉와 같이 간단하게 동영상을 출력할 수 있다. 단, DivX와 같은 동영상을 재생할 수 없는 아쉬움이 있다.

동영상 파일을 여는 데는 cvCaptureFromFile() 함수가 사용되며 영상을 얻어오기 위해서는 cvGrabFrame() 함수와 cvRetrieveFrame() 함수가 함께 사용된다. 즉, cvGrabFrame()가 호출될 때마

<화면 6> OpenCV를 이용한 동영상과 PC 카메라 영상 출력

다 내부 버퍼에 현재 프레임이 저장되며 cvRetrieveFrame()을 통해 영상을 포인터 형태로 받아온다. 그러므로 〈리스트 2〉에서 frame과 같이 넘어온 프레임의 메모리를 해제해서는 안 된다. 영상처리는 넘어온 frame에 대해서 하면 된다.

〈리스트 2〉 동영상 출력 예제

```
void COpenCVTestDlg::OnBtnAvi()
{
    CvCapture* capture=NULL;
    IplImage *frame, *frame_copy = 0;
    // 동영상 파일 읽기
    capture = cvCaptureFromFile("d:/su.avi");
    cvNamedWindow( "result", 1 );

    if( capture )
    {
      for(;;)
      {
         // 동영상에서 한 프레임을 가져온다.
              if( !cvGrabFrame( capture ))
                  break;

         // 가져온 프레임을 frame에 넘겨준다.
         frame = cvRetrieveFrame( capture );
         if( !frame )
            break;

         cvShowImage( "result", frame );
         if( cvWaitKey( 10 ) >= 0 )
            break;
      }

     cvReleaseImage( &frame_copy );
     cvReleaseCapture( &capture );

   }else{
      AfxMessageBox("지원되지 않는 파일입니다.");
   }
}
```

PC 카메라에서 영상을 받아오는 방법은 〈리스트 2〉와 동일하며 단지, cvCaptureFromFile() 함수 대신에 cvCaptureFromCAM() 함수를 사용한다는 점이 다르다.

```
// PC 카메라 장치를 열고 PC 카메라 객체의 포인터를 넘겨주는 함수
CvCapture* cvCaptureFromCAM( int index ); // 영상을 받아올 PC 카메라의 인덱스 번호
```

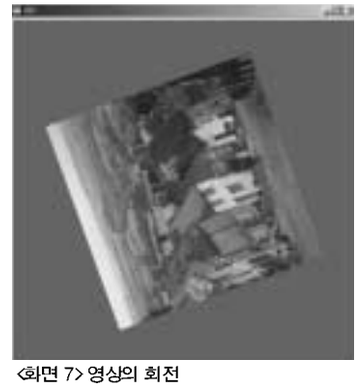
여기서 Index는 PC 카메라가 연결됐을 때 각 카메라를 가리키는 번호다. 보통 연결된 PC 카메라가 하나일 때는 0을 쓰면 되고, 여러 개라면 연결된 번호를 지정하면 그 PC 카메라로부터 영상을 받아올 수 있다. 이것은 cvCaptureFromCAM() 등의 함수가 비디오 캡처 기술인 VFW(Video for Windows)를 기반으로 만들어져 있기 때문이다. VFW에서 PC 카메라 장치를 열 때에 ID를 받아오며 ID는 보통 0에서부터 증가하므로 두 개의 PC 카메라를 사용한다면 0, 1을 지정하면 된다. VFW를 이용하면 〈리스트 2〉와 같이 영상을 재생할 수도 있으며, PC 카메라에서 영상을 받아올 수 있다. 자세한 내용은 MSDN을 참조하기 바란다. 예제와 함께 자세한 설명을 찾아볼 수 있을 것이다.

영상처리 기본 알고리즘의 구현

지금까지 영상 출력에 대한 부분을 알아봤다. 이제 기본적인 영상처리 알고리즘에 대해 하나하나씩 배워보자.

영상의 회전, 크기 변경

우선, 〈화면 7〉과 같이 영상을 회전시키는 방법에 대해 살펴보자. OpenCV에서 영상을 회전시키기 위해서는 cvGetQuadrangleSubPix() 함수를 사용한다. 이 함수를 사용하면 영상의 특정 위치를 중심으로 원하는 만큼 회전시킬 수 있으며 크기도 조정할 수 있다.


<화면 7> 영상의 회전

```
// 영상의 회전 및 크기 변화를 다루는 함수
void cvGetQuadrangleSubPix(
   const CvArr* src,                // 원본 영상
   CvArr* dst,                      // 결과 영상
   const CvMat* map_matrix,         // 회전될 각도 및 회전 중심을 가리키는 행렬
   int fill_outliers=0,             // 회전 후 빈 영역을 채우는 방법
   CvScalar fill_value=cvScalarAll(0)); // fill_outliers=1일 때, 빈 영역을 채우는 색상
```

여기서 중요한 것은 세 번째 인자 const CvMat* map_matrix다. CvMat* map_matrix는 다음 같은 형태로 회전 행렬을 정의하며 회

전 각도는 θ, 영상의 확대/축소 배율을 a, 회전 중심을 (x, y)로 나타내고 있다.

$$\beta_\theta = a \times \begin{bmatrix} \cos\theta & \sin\theta & x \\ -\sin\theta & \cos\theta & y \end{bmatrix}$$

그러므로 영상의 (10, 10)을 중심으로 30도만큼 회전하고 두 배 확대하고 싶다면 다음과 같이 표현할 수 있다.

$$\beta_\theta = 2 \times \begin{bmatrix} \cos\frac{\pi}{6} & \sin\frac{\pi}{6} & 10 \\ -\sin\frac{\pi}{6} & \cos\frac{\pi}{6} & 10 \end{bmatrix}$$

네 번째 인자 fill_outliers를 이용해 원본 영상에 해당하지 않는 영역을 채워주는 방법을 정하는데, 기본값 fill_outliers=0을 사용하면 된다. 다음 예제는 OpenCV의 도움말 파일에서 제공하는 예제다. OpenCV 도움말이 빈약하긴 하지만, 다음과 같은 예제가 종종 나오므로 참고하면 큰 도움이 된다.

〈리스트 3〉에서는 CvMat M = cvMat(2, 3, CV_32F, m);를 이용해 32비트 소수형의 2×3 행렬을 만들어 앞에서 언급한 방법으로 회전 행렬을 만든다. 또한, factor는 영상의 크기를 조절하여 영상이 회전하면서 크기도 변하게 한다.

영상의 임계 값 연산

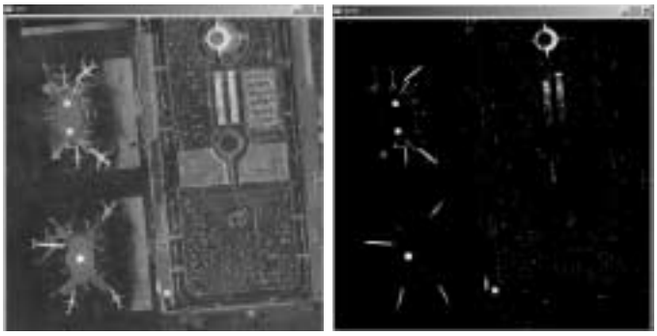
영상에서 임계 값(threshold)을 이용하여 영상의 밝기를 구분하는 방법에 대해 알아보겠다. 〈화면 8〉은 영상에서 임계 값 이상의 값만을 표시한 결과다. 임계 값을 이용하는 방법은 영상처리의 많은 부분에서 활용되는 기본적인 방법이므로 개념을 이해해두면 도움이 될 것이다.

```
void cvThreshold( const CvArr* src,      // 원본 영상, 단 그레이 영상만 가능
                  CvArr* dst,            // 결과 영상, src와 동일한 종류의 영상
                  double threshold,      // 임계값 : 보통 0~255 사이의 값
                  double max_value,      // threshold_type에서 채울 색상값 : 0~255
                  int threshold_type ); // 임계값 적용 종류
```

〈리스트 4〉와 같이 cvThreshold() 역시 사용에 어려운 점은 없다. 단, 네 가지 임계값 방법에 따라 결과가 조금씩 다른 것을 직접 확인해 보길 바란다. CV_THRESH_BINARY와 CV_THRESH_BINARY_INV, CV_THRESH_TRUNC, CV_THRESH_TOZERO 방법이 있으며 〈화면 8〉은 임계 값 이상 되는 부분만을 표시하기 위해 CV_THRESH_TOZERO를 사용했다.

〈리스트 3〉 영상의 회전, 크기 변경

```
void COpenCVTestDlg::OnBtnRotation()
{
   IplImage* src = cvLoadImage("dog.jpg", -1);
   if(src!=0)
   {
      IplImage* dst = cvCloneImage( src );
      int delta = 1;
      int angle = 0;

      cvNamedWindow( "src", 1 );
      cvShowImage( "src", src );

      for(;;)
      {
         float m[6];
         double factor = (cos(angle*CV_PI/180.) + 1.1)*3; // 영상의 크기 변경
         CvMat M = cvMat( 2, 3, CV_32F, m );
         int w = src->width;
         int h = src->height;

         m[0] = (float)(factor*cos(-angle*2*CV_PI/180.));
         m[1] = (float)(factor*sin(-angle*2*CV_PI/180.));
         m[2] = w*0.5f; // x 회전 중심
         m[3] = -m[1];
         m[4] = m[0];
         m[5] = h*0.5f; // y 회전 중심

         cvGetQuadrangleSubPix( src, dst, &M, 1, cvScalarAll(0));
         cvNamedWindow( "dst", 1 );
         cvShowImage( "dst", dst );

         if( cvWaitKey(5) == 27 )
            break;

         angle = (angle + delta) % 360;
      }
   }else{
         AfxMessageBox("파일을 찾을 수 없습니다");
   }

         cvReleaseImage(&src);
}
```

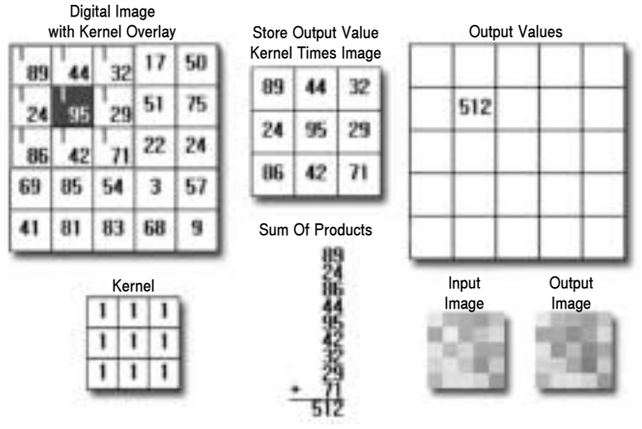


<화면 8> 영상에 임계 값 이하의 값을 0으로 준 결과

〈리스트 4〉 영상의 임계 값 연산

```
void COpenCVTestDlg::OnBtnThreshold()
{
   IplImage* src=cvLoadImage("dog.jpg", 0);
   IplImage* dst_image=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

   cvThreshold(src, dst_image, 150, 100, CV_THRESH_TOZERO);

   // 여기에서 cvNamedWindow(), cvShowImage()를 이용한 결과 화면 출력,
   // IplImage의 리소스 해제
}
```

<그림 3> 마스크를 convolution하는 과정



필터를 이용한 영상처리

필터를 이용한 영상처리 방법에는 영상을 흐릿하게 만들어주는 블러(blur) 방법 및 영상의 경계를 구하는 edge, 영상을 날카롭게 만들어 주는 방법이 있다. 이러한 필터를 이용한 영상처리 방법을 영상처리에서는 공간 영역(spatial domain)에서의 영상처리라고 부르며, 커널(마스크)을 원본 영상과 convolution하는 방법을 사용한다. 참고로, 공간 영역에 대응되는 영역을 주파수 영역(frequency domain)이라 부르며 푸리에 변환(fourier transform)을 통해 영상을 주파수 영역으로 변화시킬 수 있다. 또한, 주파수 영역에서 필터링하는 방법도 존재한다.

〈그림 3〉은 커널을 convolution하는 방법을 보여준다. 커널은 원본 영상의 좌측 상단 영역에서 우측 하단 영역까지 1픽셀씩 움직이며, 커널과 곱하고 합한 결과를 커널의 중심에 해당하는 픽셀에 저장한다. 이 과정은 영상 전체 영역에 대해 적용되며 공간 영역에서 필터 연산을 하는 방법이다. convolution에 관한 자세한 내용은 관련 서적 혹은 인터넷 사이트의 예제를 참조하자(참고자료 ❹).

```
void cvSmooth(const CvArr* src, CvArr* dst,  // 원본 영상, 결과 영상
        int smoothtype=CV_GAUSSIAN,     // 블러 방법 선택
        int param1=3, int param2=0, double param3=0 ); // 각 블러 방법의 파라미터

void cvFilter2D(const CvArr* src, CvArr* dst,     // 원본 영상, 결과 영상
                const CvMat* kernel,              // 커널 종류
                CvPoint anchor=cvPoint(-1,-1));   // 커널 중앙의 위치,
```

OpenCV에서는 cvSmooth()라는 영상을 블러시키는 함수를 제공한다. 블러 방법의 종류는 smoothtype = CV_BLUR_NO_SCALE, CV_BLUR, CV_GAUSSIAN, CV_MEDIAN, CV_BILATERAL 등이 있다. param1, param2, param3은 smoothtype에 따라 정해지며 주로 커널 크기와 관련이 있다. 〈화면 9〉는 CV_BLUR를 이용한 간단한 블러 방법으로 크기가 3×3(param1=3, param2 =3)인 〈그림 4〉의 커널을 이용한 결과다.

<그림 4> 3 ×3 크기의 CV_BLUR 커널

1/9	*1/9*	*1/9*
1/9	*1/9*	*1/9*
1/9	*1/9*	*1/9*

다음으로 cvFilter2D()를 이용하면 공간 영역에서 사용되는 모든

<화면 9> CV_BLUR를 적용한 방법

<그림 5> 커널 종류

(a) Sobel horizontal

-1	-2	-1
0	0	0
1	2	1

(b) Sobel vertical

-1	0	1
-2	0	1
-1	0	1

(c) Composite Laplacian

0	-1	0
-1	5	-1
0	-1	0

(d) second composite

-1	-1	-1
-1	9	-1
-1	-1	-1

<리스트 5> 필터를 이용한 영상처리

```
void COpenCVTestDlg::OnBtnFilter()
{
   IplImage* src=cvLoadImage("dog.jpg", -1);
   IplImage* dst_image=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 3 );

   CvMat* Mat = cvCreateMat(3, 3, CV_32FC1);

   // Composite Laplacian
   Mat->data.fl[0]= 0; Mat->data.fl[1]=-1; Mat->data.fl[2]= 0;
   Mat->data.fl[3]=-1; Mat->data.fl[4]= 5; Mat->data.fl[5]=-1;
   Mat->data.fl[6]= 0; Mat->data.fl[7]=-1; Mat->data.fl[8]= 0;

   cvFilter2D(src, dst_image, Mat);//, cvPoint(1,1));; // Gaussian

   //Simple Blur - 1보다 큰 홀수여야 함.
   //cvSmooth(src, dst_image, CV_BLUR, 3, 3);

   // 여기에서 cvNamedWindow(), cvShowImage()를 이용한 결과 화면 출력 및 IplImage
의 리소스 해제

}
```

<화면 10> Composite Laplacian Filter를 적용한 결과

종류의 필터를 구현할 수 있다. const CvMat* kernel에 원하는 커널 모양을 만들어주면 해당 연산을 해줄 수 있다. 즉, 앞의 cvSmooth()의 CV_BLUR 기능을 cvFilter2D()를 이용하여 구현하고자 한다면 마스크의 모양만 〈그림 4〉와 같이 만들어주면 된다. 〈그림 5〉는 영상처리에서 사용되는 마스크의 일부를 나타내고 있다. 마스크의 종류에 관한 더 자세한 내용은 관련 서적에서 찾아볼 수 있다.

cvFilter2D() 함수 사용에 있어서는 const CvMat* kernel을 적절한 마스크로 정의하는 것 외에는 어려움이 없다. 〈화면 10〉은 Composite Laplacian 필터를 적용한 결과로 영상이 좀더 날카로워진 것을 확인할 수 있다.

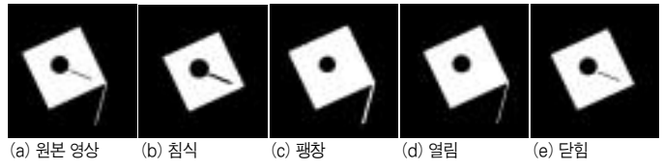
모폴로지 기법

모폴로지 기법(morphological operation)에 대한 정의는 다양하지만 형태 처리 또는 형태론 등으로 번역되고 있다. 모폴로지 기법은 어떤 영상의 형태적인 면을 조작하는 과정이다. 경계, 골격, 블럭과 같은 영역 형태를 표현하거나 서술하는 데 있어서 유용한 영상 요소들을 추출하기 위한 도구로 수학적 형태론의 개념으로 사용된다.

주로 영상 전처리 작업이나 초기 객체 분류 또는 이러한 처리 뒤에 이어서 물체의 내재된 구조를 명확히 하는 데 사용된다. 이것은 최외각선의 한 픽셀을 빼거나 더하는 작업을 통해 물체의 외곽선이나 골격선을 간단하게 함으로써 이루어진다.

모폴로지 기법 중에서 침식(erosion), 팽창(dilation), 열림(opening), 닫힘(closing)에 대해 살펴보자. 각각의 연산 결과는 〈화면 11〉과 같다.

<화면 11> 모폴로지 기법



(a) 원본 영상 (b) 침식 (c) 팽창 (d) 열림 (e) 닫힘

◆ 침식 연산은 물체에 대해 배경을 확장시키고 물체의 크기를 축소하는 역할을 한다. 침식 마스크를 이용하여 연산시 흰 물체의 둘레로부터 한 픽셀을 없애는 효과를 가진다. 노이즈와 같은 불필요한 디테일을 제거하고자 할 때 사용할 수 있다.

◆ 팽창 연산은 물체의 최외각 픽셀을 확장하는 역할을 한다. 따라서 물체의 크기는 확장되고 배경은 축소된다. 픽셀 확장을 하기 때문에 끊어진 부분을 이을 때 사용할 수 있다.

◆ 열림 연산은 침식으로 축소한 뒤 팽창 연산을 수행한다. 물체의 외곽을 부드럽게 하며 협부를 끊거나 얇게 이어진 부분을 제거하는 역할을 한다.

◆ 닫힘 연산은 팽창으로 물체의 확장을 수행한 뒤 침식 연산으로 다시 축소 연산을 수행한다. 열림처럼 물체의 외곽을 부드럽게 하지만, 이어진 부분을 보존하며 갈라진 틈이나 작은 구멍, 갭을 메우는 역할을 한다.

〈리스트 6〉 모폴로지 연산 : 팽창, 침식, 열림, 닫힘

```
void COpenCVTestDlg::OnBtnMorphologicalOperation()
{
   IplImage* src=cvLoadImage("dog.jpg", 0);
   IplImage* dst_dilation=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* dst_erosion =cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* dst_opening=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* dst_closing =cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* temp= cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* dst_closing2=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
   IplImage* dst_closing3=cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

   // 팽창
   cvDilate(src, dst_dilation);

   // 침식
   cvErode(src, dst_erosion);

   // 열림
   cvErode (src, temp);
   cvDilate(temp,dst_opening);

   // 닫힘
   cvDilate(src, temp);
   cvErode (temp,dst_closing);

   // 침식 - 사용자 정의된 엘리먼트로 침식한 결과
    IplConvKernel* element = cvCreateStructuringElementEx( 5, 5, 2, 2,
CV_SHAPE_RECT, 0 );
   cvDilate(src, temp,element,1);
   cvErode(temp,dst_closing2,element,1);

   // cvMorphologyEx를 이용한 열림 결과
   cvMorphologyEx(src, dst_closing3, temp, element, CV_MOP_CLOSE, 1);

   // 여기에서 cvNamedWindow(), cvShowImage()를 이용한
   // 결과 화면 출력 및 IplImage의 리소스 해제
}
```

〈화면 12〉 모폴로지 기법 적용 결과

(a) 원본 영상
(b) 팽창
(c) 침식
(d) 열림
(e) 닫힘
(f) 사용자 정의된 엘리먼트를 이용한 닫힘
(g) cvMorphologyEx() 함수를 이용한 닫힘

〈화면 12〉는 〈화면 8〉의 결과 영상에 대해서 모폴로지 연산을 적용시켜 본 결과다. 침식, 팽창, 열림, 닫힘의 결과를 확인할 수 있고, 특히, 침식과 열림의 경우 위의 설명한 바와 같이 노이즈를 제거할 때 사용될 수 있다. 모폴로지 기법에 관한 함수로는 침식과 팽창에 관한 함수 두 가지와 확장 기능을 수행하는 두 개의 함수로 구성되어 있다.

```
// 영상에서 침식 연산을 수행하는 함수
void cvErode( const CvArr* src,            // 원본 영상
              CvArr* dst,                  // 결과 영상
              // 엘리먼트 종류 지정, 기본은 3×3 사각형 엘리먼트
              IplConvKernel* element=NULL,
              int iterations=1 );          // 연산 적용 반복 횟수

// cvErode와 동일
void cvDilate( const CvArr* src, CvArr* dst, IplConvKernel* element=NULL,
int iterations=1 );

// 앞의 엘리먼트를 생성해주는 함수
IplConvKernel* cvCreateStructuringElementEx(
            int cols,      // 엘리먼트의 열 개수
            int rows,      // 엘리먼트의 행 개수
            int anchor_x, // 엘리먼트 중심의 x 좌표
            int anchor_y, // 엘리먼트 중심의 y 좌표
            int shape,    // 엘리먼트의 모양, CV_SHAPE_RECT, CROSS 등
            // 엘리먼트 내부의 모양을 정의. 2차원 엘리먼트를 1차원 배열로
            // 표현한 것으로 0이 아닌 값은 엘리먼트 영역을 가리킨다.
            int* values=NULL );
```

```
// 열림, 닫힘 연산 외에 확장된 모폴로지 연산을 수행한다.
void cvMorphologyEx(
            const CvArr* src,       // 원본 영상
            CvArr* dst,             // 결과 영상
            CvArr* temp,            // 특정 연산에서 사용되는 임시 영상
            IplConvKernel* element,  // structured 엘리먼트
            // 연산 종류 : CV_MOP_OPEN, CLOSE, GRADIENT, TOPHAT, BLACKHAT 등
            int operation,
            int iterations=1 ); // 연산 적용 반복 횟수
```

cvErode() 함수와 cvDilate() 함수의 인자는 동일하며 cvCreate StructuringElementEx()에서 생성된 사용자 지정 structured 엘리먼트를 사용할 수 있다. 〈리스트 6〉과 같이 열림과 닫힘 연산은 침식과 닫힘 연산을 순차적으로 적용하여 구현할 수 있다. cvCreateStructuring ElementEx()를 할 때, structured 엘리먼트를 생성할 때 크기는 홀수로 하는 것이 중심인 anchor_x, anchor_y의 위치를 잡기가 편하다. 예를 들면, 〈리스트 6〉에서는 크기가 5×5이므로 중심은 2×2인 도형 structural 엘리먼트를 생성한다. Shape은 사각형, 십자형, 타원형 외에 사용자가 직접 정의할 수 있으며 values에 따라 모양이 변한다. CvMorphologyEx() 함수는 열림, 닫힘 연산 외에 추가 연산을 지원한다.

백문이 불여일견

이번 호에는 OpenCV 라이브러리 소개 및 설치 방법, 기본적인 영상처리 알고리즘에 대해 알아봤다. OpenCV를 통해 자주 사용되는 영상처리 알고리즘들을 쉽고 빠르게 수행할 수 있다는 것을 알 수 있었을 것이다. 백문이 불여일견이란 말도 있듯이 OpenCV의 samples 폴더와 apps 폴더의 예제들을 한 번씩은 실행해 보길 권한다. 또한, '이달의 디스켓'으로 제공되는 소스에 지면상 다루지 못했던 3개의 함수도 추가했으니 참고하길 바란다. 다음 호에는 OpenCV에서 특별한 기능을 하는 함수에 대해 알아보기로 하고 이번 호를 마치겠다.

정리 | 강경수 | elegy@korea.cnet.com

이 + 달 + 의 + 디 + 스 + 켓

openCVTest.zip http://www.imaso.co.kr

참 + 고 + 자 + 료

❶ 인텔 DPST에 관한 설명 :
http://www.intel.com/pressroom/kits/centrino/cmt_backgrounder_doth.pdf
❷ OpenCV에 코드를 등록하기 위한 코딩 스타일 및 테스트 방법 :
http://www.intel.com/research/mrl/research/opencv/Coding_Style/CodingStyle.htm
❸ OpenCV 라이브러리 라이선스 :
http://www.intel.com/research/mrl/research/opencv/license.htm
❹ 인텔 OpenCV 사이트 : http://www.intel.com/research/mrl/research/opencv/
❺ Convolution 예제(마스크 연산) :
http://www.olympusmicro.com/primer/java/olympusmicd/digitalimaging/kernelmaskoperations/