

```

// (2)폰트 구조체를 초기화한다
for (i = 0; i < 16; i += 2) {
    cvInitFont (&font[i], font_face[i / 2], 1.0, 1.0);
    cvInitFont (&font[i + 1], font_face[i / 2] | CV_FONT_ITALIC, 1.0, 1.0);
}

// (3)폰트를 지정하고, 텍스트를 그리기 한다
for (i = 0; i < 16; i++) {
    irandom = cvRandInt (&rng);
    rcolor = CV_RGB (irandom & 255, (irandom >> 8) & 255, (irandom >> 16) & 255);
    cvPutText (img, "OpenCV sample code", cvPoint (15, (i + 1) * 30), &font[i], rcolor);
}

// (4)이미지의 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Text", CV_WINDOW_AUTOSIZE);
cvShowImage ("Text", img);
cvWaitKey (0);

cvDestroyWindow ("Text");
cvReleaseImage (&img);

return 0;
}

```

// (1)이미지를 확보해 초기화한다

폭 400, 높이 500 픽셀의 이미지 영역을 확보해, 초기화(제로 클리어)한다.

// (2)폰트 구조체를 초기화한다

이번은, 16 종류의 폰트와 2 종류의 자체(노멀, 이탤릭)를 지정하고, 폰트 구조체를 초기화한다. 사이즈의 비율은, 폭, 높이 모두 1.0.

// (3)폰트를 지정하고, 텍스트를 그리기 한다

함수 cvPutText()(을)를 이용하고, 텍스트의 그리기를 실시한다. 이 때에, 초기화한 폰트 구조체와 색을 지정한다. 또, OpenCV-1.0.0 시점에서 이용할 수 있는 폰트 세트에서는, 일본어의 그리기는 할 수 없다.

// (4)이미지의 표시, 키가 밀렸을 때에 종료

이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

## 실행 결과예



## IplImage 구조체 정보의 보존 cvWrite, cvWriteComment

IplImage 구조체의 정보를 파일에 보존한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char *argv[])
{
    IplImage *color_img, *gray_img;
    CvRect roi;
    CvFileStorage *fs;

    // (1) 이미지를 읽어들인다
    if (argc != 2 || (color_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;

    // (2) ROI의 설정과 2치화 처리
    gray_img = cvCreateImage (cvGetSize (color_img), IPL_DEPTH_8U, 1);
    cvCvtColor (color_img, gray_img, CV_BGR2GRAY);
    roi = cvRect (0, 0, color_img->width / 2, color_img->height / 2);
    cvSetImageROI (gray_img, roi);
    cvSetImageROI (color_img, roi);
    cvThreshold (gray_img, gray_img, 90, 255, CV_THRESH_BINARY);

    // (3) xml 파일에의 써내
    fs = cvOpenFileStorage ("images.xml", 0, CV_STORAGE_WRITE);
    cvWriteComment (fs, "This is a comment line.", 0);
    cvWrite (fs, "color_img", color_img);
    cvStartNextStream (fs);
    cvWrite (fs, "gray_img", gray_img);
    cvReleaseFileStorage (&fs);
}
```

```

cvReleaseImage (&color_img);
cvReleaseImage (&gray_img);

return 0;
}

```

// (1)이미지를 읽어들인다

함수 cvLoadImage 에 의해, 보존하기 위한 칼라 이미지를 읽어들인다.

// (2)ROI 의 설정과 2 치화 처리

함수 cvCvtColor 에 의해 칼라 이미지를 그레이 스케일 이미지로 변환해, 함수 cvSetImageROI 에 의해 ROI(을)를 설정한다. 또, 변환한 이미지에 대해서 2 치화 처리를 가한다.

// (3)xml 파일에의 써내

함수 cvWrite 에 의해, 입력된 칼라 이미지와 변환·처리된 그레이 스케일 이미지를 파일에 써낸다. 함수 cvSaveImage 과는 달리, 오브젝트의 정보를 데이터로서 보존한다. 또, 여기에서는, 둘의 IpImage 구조체의 데이터를 하나의 파일에 써내고 있지만, 하나의 데이터를 하나의 파일에 써내는 경우는,

```

cvSave("image.xml", color_img);

```

(와)과 같이 해 보존해도 좋다.함수cvSave의 내부에서는, 함수cvWrite하지만 불러 간다. 함수cvStartNextStream()(은), 파일 기입중의 스트림을 새롭게 하는 함수이며, 이 장소에서

```

<!-- next stream -->

```

그렇다고 하는 코멘트와 개행이 삽입되는 것만으로 있다(이번 경우, 표준 타입의 데이터를 톱 레벨에 쓸 뿐(만큼)이므로). 물론, 복수의 파일을 쓸 때에 필수라고 하는 것은 아니다.

### 실행 결과에

[images.xml.zip](#)하지만 실제로 써내진 파일이다. 이하의 snapshot로부터도 알 수 있듯이,ROI등의 정보도 보존되고 있다.



## 맵의 순서를 보존 cvStartWriteStruct, cvEndWriteStruct

두 개의 엔트리를 가지는 맵의 순서를 파일에 보존한다

### 샘플 코드

#### 표시의 변환

```

#include <cv.h>
#include <highgui.h>

```

```

#include <time.h>

int
main (int argc, char *argv[])
{
    int size = 20;
    int i;
    CvFileStorage *fs;
    CvRNG rng = cvRNG (time (NULL));
    CvPoint *pt = (CvPoint *) cvAlloc (sizeof (CvPoint) * size);

    // (1)점렬의 작성
    for (i = 0; i < size; i++) {
        pt[i].x = cvRandInt (&rng) % 100;
        pt[i].y = cvRandInt (&rng) % 100;
    }

    // (2)맵의 순서로서 점렬을 보존
    fs = cvOpenFileStorage ("sequence.yml", 0, CV_STORAGE_WRITE);
    cvStartWriteStruct (fs, "points", CV_NODE_SEQ);
    for (i = 0; i < size; i++) {
        cvStartWriteStruct (fs, NULL, CV_NODE_MAP | CV_NODE_FLOW);
        cvWriteInt (fs, "x", pt[i].x);
        cvWriteInt (fs, "y", pt[i].y);
        cvEndWriteStruct (fs);
    }
    cvEndWriteStruct (fs);
    cvReleaseFileStorage (&fs);

    return 0;
}

```

// (1)점렬의 작성

랜덤인 값을 가지는, CvPoint 형태의 배열(요소수 20)(을)를 작성한다.

// (2)맵의 순서로서 점렬을 보존

레퍼런스 메뉴얼에 있어서의 함수 [GetHashedKey\(\)](#)의 설명의 항에 있는 것 같은, 두 개의 엔트리를 가지는 맵의 순서로서 보존한다. 함수 cvStartWriteStruct()(을)를 이용하고, "x"(와)과 "y"의 엔트리를 가지는 맵의 기입을 실시한다. 한층 더 그러한 맵을, 마지막 인수에 CV\_NODE\_SEQ(을)를 준 함수 cvStartWriteStruct()그리고 정리하는 것으로, 순서로서 쓴다. 이번은, 레퍼런스 메뉴얼의 예에 배워, YAML 형식의 파일로서 보존한다.

## 실행 결과예



## IpImage 구조체 정보의 읽기 cvRead, cvGetFileNodeByName

IpImage 구조체의 정보를 파일로부터 읽어들인다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char *argv[])
{
    IpImage *color_img, *gray_img;
    CvRect roi_color, roi_gray;
    CvFileStorage *fs;
    CvFileNode *node;

    // (1)파일을 읽어들인다
    if (argc != 2 || (fs = cvOpenFileStorage (argv[1], 0, CV_STORAGE_READ)) == 0)
        return -1;

    node = cvGetFileNodeByName (fs, NULL, "color_img");
    color_img = (IpImage *) cvRead (fs, node);
    node = cvGetFileNodeByName (fs, NULL, "gray_img");
    gray_img = (IpImage *) cvRead (fs, node);
    cvReleaseFileStorage (&fs);

    // (2)ROI 정보를 취득해 구형을 그린 후, 해방
    roi_color = cvGetImageROI (color_img);
    roi_gray = cvGetImageROI (gray_img);
    cvResetImageROI (color_img);
    cvResetImageROI (gray_img);
    cvRectangle (color_img, cvPoint (roi_color.x, roi_color.y),
                 cvPoint (roi_color.x + roi_color.width, roi_color.y + roi_color.height),
                 CV_RGB (255, 0, 0));
    cvRectangle (gray_img, cvPoint (roi_gray.x, roi_gray.y),
                 cvPoint (roi_gray.x + roi_gray.width, roi_gray.y + roi_gray.height),
                 cvScalar (0));
```

```
// (3)이미지를 그리기
cvNamedWindow ("Color Image", CV_WINDOW_AUTOSIZE);
cvShowImage ("Color Image", color_img);
cvNamedWindow ("Grayscale Image", CV_WINDOW_AUTOSIZE);
cvShowImage ("Grayscale Image", gray_img);
cvWaitKey (0);

cvDestroyWindow ("Color Image");
cvDestroyWindow ("Grayscale Image");
cvReleaseImage (&color_img);
cvReleaseImage (&gray_img);

return 0;
}
```

// (1)파일을 읽어들인다

커멘드 인수로 지정된 파일을 오픈해, 함수 cvOpenFileStorage()에 의해 파일 스토리지 포인터를 얻는다. 함수 cvGetFileNodeByName()에 의해,"color\_img"라고"gray\_img"(이)라는 이름으로부터 노드를 취득해, 함수 cvRead()에 의해 데이터를 읽어들인다.

// (2)ROI 정보를 취득해 구형을 그린 후, 해방

함수 cvGetImageROI()에 의해, 읽어들였다 IplImage 구조체로부터 ROI의 정보를 추출해, 그 구형을 그린다. 또, 그 때문에(위해) 이미지의 ROI(을)를 해방해 둔다.

// (3)이미지를 그리기

실제로, 파일로부터 읽힌 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

여기서 읽고 있는 파일은 [IplImage구조체 정보의 보존](#) 그리고 보존되었다.xml파일이다.



## 맵의 순서를 읽어들인다 cvGetHashedKey, cvGetFileNode

두 개의 엔트리를 가지는 맵의 순서를 파일로부터 읽어들인다.이것은, 참조 설명서내의 샘플과 거의 동일하다.

### 샘플 코드

표시의 변환

```

#include <cv.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    // (1)파일 스토리지의 오픈, 핫슈드키의 계산, 순서 노드의 취득
    CvFileStorage *fs = cvOpenFileStorage ("sequence.yml", 0, CV_STORAGE_READ);
    CvStringHashNode *x_key = cvGetHashedKey (fs, "x", -1, 1);
    CvStringHashNode *y_key = cvGetHashedKey (fs, "y", -1, 1);
    CvFileNode *points = cvGetFileNodeByName (fs, 0, "points");

    // (2)순서 리더를 초기화, 각 노드를 차례차례 취득
    if (CV_NODE_IS_SEQ (points->tag)) {
        CvSeq *seq = points->data.seq;
        int i, total = seq->total;
        CvSeqReader reader;
        cvStartReadSeq (seq, &reader, 0);

        for (i = 0; i < total; i++) {
            CvFileNode *pt = (CvFileNode *) reader.ptr;

        #if 1
            // (3)고속 버전
            CvFileNode *xnode = cvGetFileNode (fs, pt, x_key, 0);
            CvFileNode *ynode = cvGetFileNode (fs, pt, y_key, 0);
            assert (xnode && CV_NODE_IS_INT (xnode->tag) && ynode && CV_NODE_IS_INT (ynode->tag));
            int x = xnode->data.i;    /* 혹은 x = cvReadInt( xnode, 0 ); */
            int y = ynode->data.i;    /* 혹은 y = cvReadInt( ynode, 0 ); */

        #elif 1
            // (4)저속 버전.x_key(와)과 y_key(을)를 사용하지 않는다
            CvFileNode *xnode = cvGetFileNodeByName (fs, pt, "x");
            CvFileNode *ynode = cvGetFileNodeByName (fs, pt, "y");
            assert (xnode && CV_NODE_IS_INT (xnode->tag) && ynode && CV_NODE_IS_INT (ynode->tag));
            int x = xnode->data.i;    /* 혹은 x = cvReadInt( xnode, 0 ); */
            int y = ynode->data.i;    /* 혹은 y = cvReadInt( ynode, 0 ); */

        #else
            // (5)한층 더 저속이지만, 사용하기 쉬운 버전
            int x = cvReadIntByName (fs, pt, "x", 0);
            int y = cvReadIntByName (fs, pt, "y", 0);
        #endif
        }
    }
}

```

```

#endif
    // (6)데이터를 표시해, 다음의 순서 노드를 취득
    CV_NEXT_SEQ_ELEM (seq->elem_size, reader);
    printf ("%d: (%d, %d)\n", i, x, y);
}
}
cvReleaseFileStorage (&fs);
return 0;
}

```

// (1)파일 스토리지의 오픈, 핫슈드키의 계산, 순서 노드의 취득  
 최초로, 파일명을 지정해 파일 스토리지를 오픈한다. 또, 각 순서의 두 개의 엔트리의 키이다"x","y"에 대한 독특한 값을 함수 cvGetHashedKey()에 의해 계산하고,CvStringHashNode 형태의 포인터를 취득한다. 여기서, 함수 cvGetHashedKey()의 제 3 인수는, 키의 이름의 길이이므로, 여기에서는,"1"(을)를 지정해도 좋지만,0 이하의 값을 지정하는 것으로 자동적으로 문자열의 길이가 계산된다.  
 오픈된 파일 노드로부터,"points"(이)라는 이름을 가지는 순서 노드의 포인터를 취득한다.

// (2)순서 리더를 초기화, 각 노드를 차례차례 취득  
 함수 cvStartReadSeq()(을)를 이용하고, 순서 리더에 의한 연속 읽기 처리를 초기화해, 차례차례 노드를 읽어들인다. 노드의 취득은, 초기화된 순서 리더가 가리키는 노드(ptr 멤버)를 캐스트 하는 것으로 행해진다. 또, 이러한 처리 전에, 미리 취득한 파일 노드의 종류가 순서인 것을 확인하기 위해서, 매크로 CV\_NODE\_IS\_SEQ()(을)를 이용하고 있다. 노드의 멤버 tag(을)를 이용하고, 그 노드의 종류를 아는 매크로는,CV\_NODE\_IS\_\*그렇다고 하는 형태로,cxtypes.h 안에서 정의되고 있다.

// (3)고속 버전  
 미리 계산되었다 CvStringHashNode 형태의 포인터를 이용하고,"x","y"의 이름을 가지는 노드를 취득한다. 이것은, 키의 문자열의 내용을 비교하는 것이 아니라, 독특한 포인터를 비교하는 것으로써 노드를 요구하기 위해, 후술의 함수 cvGetFileNodeByName()(와)과 비교하면 약간 고속으로 동작한다. 또, 취득한 노드의 값을 직접 참조하는 것이, 함수 cvReadInt()(을)를 이용하고 값을 읽어내는 것보다도 약간 고속으로 있다.

// (4)저속 버전.x\_key(와)과 y\_key(을)를 사용하지 않는다  
 함수 cvGetFileNodeByName()에 의해 노드를 취득하는 일 이외는, 전술의 예와 같이.

// (5)한층 더 저속이지만, 사용하기 쉬운 버전  
 함수 cvReadIntByName()(은)는, 그 내부에서 함수 cvGetFileNodeByName 라고 함수 cvReadInt()(을)를 호출하고 있다. 함수 cvReadInt()의 호출 분의 오버헤드가 걸리기 위해, 상술의 예보다 한층 더 약간 저속이 된다.

// (6)데이터를 표시해, 다음의 순서 노드를 취득  
 상술의 몇개의 방법으로 취득된 값을 표시한다. 또, 매크로 CV\_NEXT\_SEQ\_ELEM()에 의해, 순서 리더가 가리키는 노드를 요소의 사이즈분만큼 진행하고(혹은 블록을 변환이라고), 다음의 노드를 취득한다.



## 실행 결과예

여기서 읽히고 있는 파일은,맵의 순서를 보존그리고 보존되었다YAML파일이다.



## K-means 법에 따르는 클러스터링 cvKMeans2

K-means 법으로, 샘플의 클러스터링을 실시한다.이것은,OpenCV 부속의 샘플 코드(와 거의 동일)이다.

## 샘플 코드

### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <time.h>

#define MAX_CLUSTERS (5)

int
main (int argc, char **argv)
{
    CvScalar color_tab[MAX_CLUSTERS] =
        { CV_RGB (255, 0, 0), CV_RGB (0, 255, 0), CV_RGB (100, 100, 255), CV_RGB (255, 0,
255), CV_RGB (255, 255, 0) };
    IplImage *img = cvCreateImage (cvSize (500, 500), IPL_DEPTH_8U, 3);
    CvRNG rng = cvRNG (time (NULL));
    CvPoint ipt;

    while (1) {
        int c;
        int k, cluster_count = cvRandInt (&rng) % MAX_CLUSTERS + 1;
        int i, sample_count = cvRandInt (&rng) % 1000 + MAX_CLUSTERS;
        CvMat *points = cvCreateMat (sample_count, 1, CV_32FC2);
        CvMat *clusters = cvCreateMat (sample_count, 1, CV_32SC1);

        // (1)복수의 가우시안으로부터 완성되는 랜덤 샘플을 생성한다
        for (k = 0; k < cluster_count; k++) {
            CvPoint center;
            CvMat point_chunk;
            center.x = cvRandInt (&rng) % img->width;
            center.y = cvRandInt (&rng) % img->height;
            cvGetRows (points, &point_chunk, k * sample_count / cluster_count,
                k == cluster_count - 1 ? sample_count : (k + 1) * sample_count /
```

```

cluster_count, 1);
    cvRandArr (&rng, &point_chunk, CV_RAND_NORMAL,
               cvScalar (center.x, center.y, 0, 0), cvScalar (img->width * 0.1, img-
>height * 0.1, 0, 0));
}

// (2)랜덤 샘플을 상출 한다
for (i = 0; i < sample_count / 2; i++) {
    CvPoint2D32f *pt1 = (CvPoint2D32f *) points->data.fl + cvRandInt (&rng) %
sample_count;
    CvPoint2D32f *pt2 = (CvPoint2D32f *) points->data.fl + cvRandInt (&rng) %
sample_count;
    CvPoint2D32f temp;
    CV_SWAP (*pt1, *pt2, temp);
}

// (3)K-menas 법에 따르는 클러스터링
cvKMeans2 (points, cluster_count, clusters, cvTermCriteria (CV_TERMCRIT_EPS +
CV_TERMCRIT_ITER, 10, 1.0));

// (4)클러스터마다 색을 바꾸어 샘플을 그리기 한다
cvZero (img);
for (i = 0; i < sample_count; i++) {
    int cluster_idx = clusters->data.i[i];
    ipt.x = (int) points->data.fl[i * 2];
    ipt.y = (int) points->data.fl[i * 2 + 1];
    cvCircle (img, ipt, 2, color_tab[cluster_idx], CV_FILLED, CV_AA, 0);
}

cvReleaseMat (&points);
cvReleaseMat (&clusters);

// (5)이미지를 표시, "Esc"키가 밀렸을 때에 종료
cvNamedWindow ("Clusters", CV_WINDOW_AUTOSIZE);
cvShowImage ("Clusters", img);
c = cvWaitKey (0);
if (c == 'Wx1b')
    break;

cvReleaseMat (&clusters);
cvReleaseMat (&points);
}

cvDestroyWindow ("Clusters");
cvReleaseImage (&img);

```

```
return 0;
```

```
}
```

// (1)복수의 가우시안으로부터 완성되는 랜덤 샘플을 생성한다

우선, 클러스터링 대상 데이터가 되는 데이터 집합을 작성한다. 각 데이터는 2 차원 좌표를 가지는 점이며, 그 전데이터는,  $1 \times \text{sample\_count}$  의 행렬( 각 요소는, CV\_32FC2)points 그리고 나타내진다. 함수 cvGetRows 에 의해 의사적으로(클러스터 몇분에) 분할된 부분 행렬을, 정규 분포 하는 데이터로 초기화한다.이 때의 평균은, 최초로 랜덤에 생성된 값, 분산은,  $0.1 \times \text{이미지의 크기}$ , 이다. 이 행렬의 초기화에는, 함수 cvRandArr()(을)를 이용한다.

// (2)랜덤 샘플을 상환 한다

또, 생성된 샘플은, 이 시점에서는 각 클러스터 마다 올바르게 정렬하고 있으므로, 이것을, 매크로 CV\_SWAP()에 의해 상환 한다. 매크로 CV\_SWAP(은)는,cxtypes.h 안에서, 다음과 같이 정의된다.

```
#define CV_SWAP(a,b,t) ((t) = (a), (a) = (b), (b) = (t))
```

// (3)K-menas 법에 따르는 클러스터링

함수 cvKMeans2()에 의해,K-menas 법에 따르는 클러스터링을 실시한다. 인수에는, 각각, 샘플 데이터, 클러스터수, 출력 클러스터, 종료 조건, 을 준다. 출력 클러스터에는, 각 샘플이 어느 클러스터에 속하는지를 나타내는 클러스터 인덱스가 들어간다. 또, 종료 조건은, 반복 계산의 최대수와 센터의 각 스텝에 있어서의 이동거리의 최소치가 주어져 이 어느 쪽인지가 만족되면 계산이 종료한다.

// (4)클러스터마다 색을 바꾸어 샘플을 그리기 한다

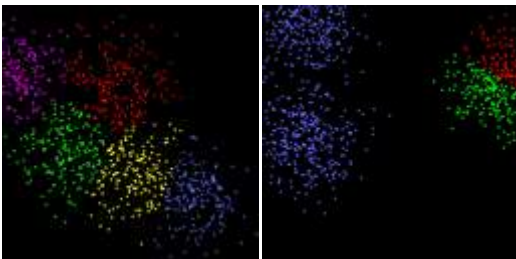
clusters->data.i[i]에,i 차례째의 샘플이 속하는 클러스터의 인덱스 (int 형태)가 보존되고 있으므로, 거기에 따라, 모든 샘플을 분류 해 엔으로 그리기 한다.

// (5)이미지를 표시,"Esc"키가 밀렸을 때에 종료

실제로 이미지를 표시해, 유저의 입력을 기다린다. "Esc"키가 밀렸을 경우는 종료해, 그 이외의 입력이 맞았을 경우는, 다시 랜덤 샘플을 생성한다.

### 실행 결과예

각 클러스터를 대표하는 최초의 센터는,OpenCV내부에서 랜덤에 초기화되기 위해, 반드시 적절한 클러스터링 결과가 되는 것은 아니다.



## 클러스터링에 의한 감소색처리 cvKMeans2

k-means 법에 따르는 클러스터링을 이용하고, 매우 단순한 감소색을 실시한다

```

#include <cv.h>
#include <highgui.h>

#define MAX_CLUSTERS (32)      /* 클러스터수 */

int
main (int argc, char **argv)
{
    int i, size;
    IplImage *src_img = 0, *dst_img = 0;
    CvMat *clusters;
    CvMat *points;
    CvMat *color = cvCreateMat (MAX_CLUSTERS, 1, CV_32FC3);
    CvMat *count = cvCreateMat (MAX_CLUSTERS, 1, CV_32SC1);

    // (1) 이미지를 읽어들인다
    if (argc != 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;

    size = src_img->width * src_img->height;
    dst_img = cvCloneImage (src_img);
    clusters = cvCreateMat (size, 1, CV_32SC1);
    points = cvCreateMat (size, 1, CV_32FC3);

    // (2) 픽셀의 값을 행렬에 대입
    for (i = 0; i < size; i++) {
        points->data.fl[i * 3 + 0] = (uchar) src_img->imageData[i * 3 + 0];
        points->data.fl[i * 3 + 1] = (uchar) src_img->imageData[i * 3 + 1];
        points->data.fl[i * 3 + 2] = (uchar) src_img->imageData[i * 3 + 2];
    }

    // (3) 클러스터링
    cvKMeans2 (points, MAX_CLUSTERS, clusters, cvTermCriteria (CV_TERMCRIT_EPS +
CV_TERMCRIT_ITER, 10, 1.0));

    // (4) 각 클러스터의 평균치를 계산
    cvSetZero (color);
    cvSetZero (count);
    for (i = 0; i < size; i++) {
        int idx = clusters->data.i[i];
        int j = ++count->data.i[idx];
        color->data.fl[idx * 3 + 0] = color->data.fl[idx * 3 + 0] * (j - 1) / j + points-

```

```

>data.fl[i * 3 + 0] / j;
    color->data.fl[idx * 3 + 1] = color->data.fl[idx * 3 + 1] * (j - 1) / j + points->data.fl[i * 3 + 1] / j;
>data.fl[i * 3 + 1] / j;
    color->data.fl[idx * 3 + 2] = color->data.fl[idx * 3 + 2] * (j - 1) / j + points->data.fl[i * 3 + 2] / j;
>data.fl[i * 3 + 2] / j;
}

// (5)클러스터마다 색을 그리기
for (i = 0; i < size; i++) {
    int idx = clusters->data.i[i];
    dst_img->imageData[i * 3 + 0] = (char) color->data.fl[idx * 3 + 0];
    dst_img->imageData[i * 3 + 1] = (char) color->data.fl[idx * 3 + 1];
    dst_img->imageData[i * 3 + 2] = (char) color->data.fl[idx * 3 + 2];
}

// (6)이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
cvShowImage ("src", src_img);
cvNamedWindow ("low-color", CV_WINDOW_AUTOSIZE);
cvShowImage ("low-color", dst_img);
cvWaitKey (0);

cvDestroyWindow ("src");
cvDestroyWindow ("low-color");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseMat (&clusters);
cvReleaseMat (&points);
cvReleaseMat (&color);
cvReleaseMat (&count);

return 0;
}

```

// (1)이미지를 읽어들인다

함수 cvLoadImage() 에 의해, 입력상을 칼라 이미지로서 읽어들인다.

// (2)픽셀의 값을 행렬에 대입

각 픽셀을 클러스터링 하기 위해서, 각 픽셀의 RGB 채널의 값을 CvMat 형태의 변수에 대입한다.

// (3)클러스터링

전술의 샘플과 같게, 함수 cvKMeans2() 에 의해,k-means 법을 이용한 클러스터링을 실시한다. 클러스터수는 미리 정의되고 있어 이것이, 감소색 후의 색 가지수가 된다.

// (4)각 클러스터의 평균치를 계산

클러스터링 된 각 픽셀군을, 그 클러스터의 센터에서 대표할 수 있으면 처리가 간단하지만, OpenCV 그림, 함수 cvKMeans2() 에 의해 클러스터링 된 각 클러스터의 센터에 액세스 할 수

없다. 거기서, 이 샘플에서는 클러스터링 후의 각 클러스터의(K 개의 대표치는 아니다) 완전한 평균치를 요구해 그것을 각 클러스터의 대표치로 한다.

// (5)클러스터마다 색을 그리기

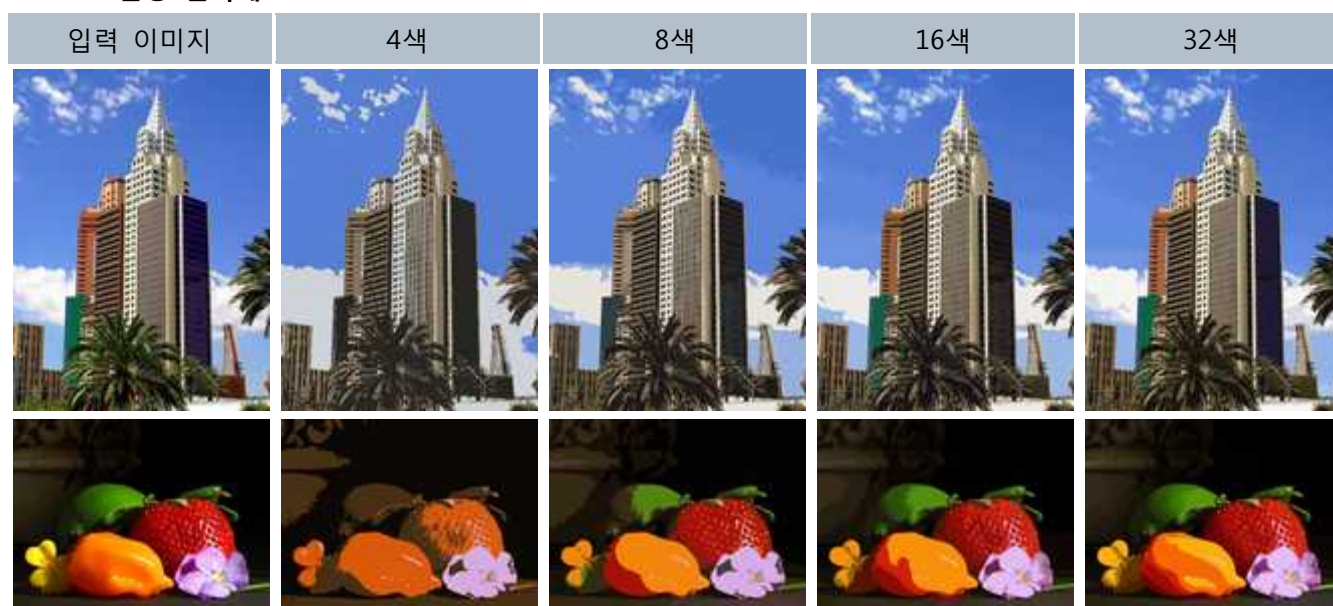
각 클러스터의 픽셀을, 구할 수 있던 평균치로 묻는다.

// (6)이미지를 표시, 키가 밀렸을 때에 종료

실제로, 입력 이미지와 클러스터링(감소색) 된 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

이와 같이 K-means 법을 이용해 감소색을 실시하는 일은, 색에 의한 이미지의 영역 분할과 동일하다. 그러한 처리에 대해서는, [이미지 분할, 영역 결합, 윤곽 검출](#) (을)를 참조의 일.

### 실행 결과예



엣지의 검출 cvSobel, cvLaplace, cvCanny

Sobel,Laplacian,Canny 에 의한 엣지 검출

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img, *dst_img1, *dst_img2, *dst_img3;
    IplImage *tmp_img;

    // (1)이미지의 읽기
```

```

if (argc != 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
    return -1;

tmp_img = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_16S, 1);
dst_img1 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
dst_img2 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
dst_img3 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);

// (2)Sobel 필터에 의한 미분 이미지의 작성
cvSobel (src_img, tmp_img, 1, 0);
cvConvertScaleAbs (tmp_img, dst_img1);

// (3)이미지의 Laplacian 의 작성
cvLaplace (src_img, tmp_img);
cvConvertScaleAbs (tmp_img, dst_img2);

// (4)Canny 에 의한 엣지 이미지의 작성
cvCanny (src_img, dst_img3, 50.0, 200.0);

// (5)이미지의 표시
cvNamedWindow ("Original", CV_WINDOW_AUTOSIZE);
cvShowImage ("Original", src_img);
cvNamedWindow ("Sobel", CV_WINDOW_AUTOSIZE);
cvShowImage ("Sobel", dst_img1);
cvNamedWindow ("Laplace", CV_WINDOW_AUTOSIZE);
cvShowImage ("Laplace", dst_img2);
cvNamedWindow ("Canny", CV_WINDOW_AUTOSIZE);
cvShowImage ("Canny", dst_img3);
cvWaitKey (0);

cvDestroyWindow ("Original");
cvDestroyWindow ("Sobel");
cvDestroyWindow ("Laplace");
cvDestroyWindow ("Canny");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img1);
cvReleaseImage (&dst_img2);
cvReleaseImage (&dst_img3);
cvReleaseImage (&tmp_img);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고



읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_GRAYSCALE (을)를 지정하는 것으로, 원이미지를 그레이 스케일 이미지로써 읽어들이는다.

// (2)Sobel 필터에 의한 미분 이미지의 작성  
x 방향으로 1 다음 미분하기 위한 ,3×3 사이즈의 Sobel 필터를 이용해 미분 이미지를 작성한다. 여기에서는,aperture\_size(을)를 지정하고 있지 않지만, 그 경우는 디폴트의 사이즈 3 하지만 이용된다. 또, 여기서 CV\_SCHARR(=-1)(을)를 지정하면,Sharr 필터를 이용할 수 있다. 이 함수에서는 슬캘링은 행해지지 않기 때문에, 출력 이미지의 각 픽셀치는, 대부분의 경우 입력 이미지의 그것보다 큰 값이 된다.오버플로우를 피 차기 위해서, 예를 들면 입력 이미지가 8 비트의 경우, 출력 이미지로서 16 비트 이미지를 지정할 필요가 있다.










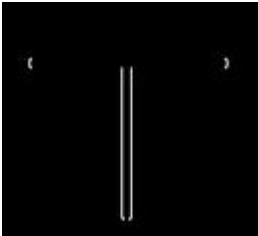
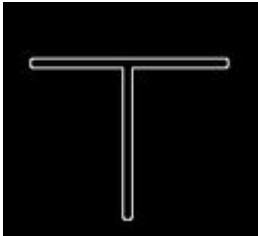
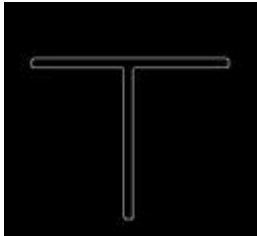
// (3)이미지의 Laplacian 의 작성  
이하와 같이,x-y 양방향의 2 차 미분의 화를 계산해, 이미지의 Laplacian(을)를 작성한다.

$dst(x,y) = d^2src/dx^2 + d^2src/dy^2$   
이 함수에서도,cvSobel의 경우와 같게 출력 이미지로서16비트 이미지를 지정한다.

// (4)Canny 에 의한 엣지 이미지의 작성  
함수 cvCanny()하지만 출력하는 결과는, 전술의 두 개의 함수와는 달라, 구배 이미지는 아니다. 여기서 출력되는 이미지는, 엣지인가, 그렇지 않은가의 2 치 이미지가 된다. cvCanny()(은)는, 2 종류의 반응을 일으키는 최소의 물리량을 인수(3 번째 ,4 번째 )에 있어, 작은 편이 엣지끼리를 접속하기 위해서 이용되어 큰 편이 강한 엣지 의 초기 검출에 이용된다. 엣지의 접속에서는, 주목 픽셀의 8 근방(4 방향)에 대한 구배가 요구되어 가장 강한 구배를 가질 방향으로(반응을 일으키는 최소의 물리량을 밀돌지 않는 한) 엣지가 접속된다.

// (5)이미지의 표시  
입력 이미지와 처리 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

입력 이미지	결과(Sobel)	결과(Laplacian)	결과(Canny)
			
			
			



## 코너의 검출 cvGoodFeaturesToTrack, cvFindCornerSubPix

이미지중의 코너(특징점) 검출

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, corner_count = 150;
    IplImage *dst_img1, *dst_img2, *src_img_gray;
    IplImage *eig_img, *temp_img;
    CvPoint2D32f *corners;

    if (argc != 2 || (dst_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYCOLOR |
CV_LOAD_IMAGE_ANYDEPTH)) == 0)
        return -1;

    dst_img2 = cvCloneImage (dst_img1);
    src_img_gray = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE);
    eig_img = cvCreateImage (cvGetSize (src_img_gray), IPL_DEPTH_32F, 1);
    temp_img = cvCreateImage (cvGetSize (src_img_gray), IPL_DEPTH_32F, 1);
    corners = (CvPoint2D32f *) cvAlloc (corner_count * sizeof (CvPoint2D32f));

    // (1)cvCornerMinEigenVal(을)를 이용한 코너 검출
    cvGoodFeaturesToTrack (src_img_gray, eig_img, temp_img, corners, &corner_count, 0.1,
15);
    cvFindCornerSubPix (src_img_gray, corners, corner_count,
                        cvSize (3, 3), cvSize (-1, -1), cvTermCriteria (CV_TERMCRIT_ITER |
CV_TERMCRIT_EPS, 20, 0.03));
    // (2)코너의 그리기
    for (i = 0; i < corner_count; i++)
        cvCircle (dst_img1, cvPointFrom32f (corners[i]), 3, CV_RGB (255, 0, 0), 2);

    // (3)cvCornerHarris(을)를 이용한 코너 검출
    corner_count = 150;
    cvGoodFeaturesToTrack (src_img_gray, eig_img, temp_img, corners, &corner_count, 0.1,
15, NULL, 3, 1, 0.01);
```

```

cvFindCornerSubPix (src_img_gray, corners, corner_count,
                    cvSize (3, 3), cvSize (-1, -1), cvTermCriteria (CV_TERMCRIT_ITER |
CV_TERMCRIT_EPS, 20, 0.03));
// (4)코너의 그리기
for (i = 0; i < corner_count; i++)
    cvCircle (dst_img2, cvPointFrom32f (corners[i]), 3, CV_RGB (0, 0, 255), 2);

// (5)이미지의 표시
cvNamedWindow ("EigenVal", CV_WINDOW_AUTOSIZE);
cvShowImage ("EigenVal", dst_img1);
cvNamedWindow ("Harris", CV_WINDOW_AUTOSIZE);
cvShowImage ("Harris", dst_img2);
cvWaitKey (0);

cvDestroyWindow ("EigenVal");
cvDestroyWindow ("Harris");
cvReleaseImage (&dst_img1);
cvReleaseImage (&dst_img2);
cvReleaseImage (&eig_img);
cvReleaseImage (&temp_img);
cvReleaseImage (&src_img_gray);

return 0;
}

```

// (1)cvCornerMinEigenVal(을)를 이용한 코너 검출

함수 cvGoodFeaturesToTrack() (은)는, 이미지중으로부터 큰 고유치를 가지는 코너를 검출한다. 이 함수는 최초로, 모든 입력 이미지의 픽셀에 대해서, cvCornerMinEigenVal (을)를 이용해 최소의 고유치를 계산해, 결과를(2 번째의 인수로 지정되었다) eig\_image 에 보존한다. 다음에, "non-maxima suppression"(을)를 실행한다(3×3 의 인접 영역내의 극대만이 남는다). 다음의 스텝에서는, (6 번째의 인수로 지정되었다)quality\_levelth (와)과 max(eig\_image(x,y)) 의 적보다 작은 최소 고유치를 가지는 코너를 삭제한다. 마지막으로, 이 함수는 코너점에 주목해(가장 강한 코너가 제일 최초로 대상이 된다), 새롭게 주목한 특징점으로 그 이전에 대상으로 한 특징점군과의 거리가 (7 번째의 인수로 지정되었다) min\_distance 보다 큰 것을 체크하는 것으로, 모든 검출된 코너 각각의 거리가 충분히 떨어져 있는 것을 보증한다. 그 때문에, 이 함수는 선명한 특징점과의 거리가 가까운 특징점을 삭제한다. 게다가 함수 cvFindCornerSubPix()(을)를 이용하고,4 번째의 인수로 지정한 사이즈의 배의 탐색 윈도우로, 코너(특징점)의 것보다 정확한 좌표를 탐색한다.

// (2)코너의 그리기

검출된 코너를 적색의 엔으로 그리기 한다.

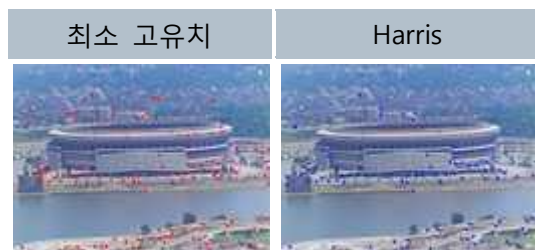
// (3)cvCornerHarris(을)를 이용한 코너 검출

10 번째의 인수가 0(이)가 아닌 경우는, 디폴트의 cvCornerMinEigenVal 대신에, Harris 오퍼레이터(cvCornerHarris)(을)를 이용한다.

// (4)코너의 그리기  
검출된 코너를 청색의 엔으로 그리기 한다.

// (5)이미지의 표시  
실제로 검출된 코너의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



#### 병진 이동을 위한 픽셀 산프링 cvGetRectSubPix

지정한 좌표가 이미지 중심이 되도록(듯이) 원이미지를 병진 시킨다.비정수치 좌표의 픽셀치는, 바이리니아 보간에 의해 얻을 수 있다, 또 이미지 경계의 외측에 존재하는 픽셀의 값을 얻기 위해서, 복제 경계 모드(이미지의 가장 외측의 픽셀치가 외측 무한원까지 성장하고 있으면 가정) 하지만 사용된다.

#### 샘플 코드

##### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img = 0;
    CvPoint2D32f center;

    // (1)이미지의 읽어들이어, 출력용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    // (2)dst_img 의 이미지 중심이 된다 src_img 안의 위치 center(을)를 지정한다
    center.x = src_img->width - 1;
    center.y = src_img->height - 1;

    // (3)center 하지만 이미지 중심이 되도록(듯이),GetRectSubPix(을)를 이용해 이미지 전체를
    시프트 시킨다
```

```
cvGetRectSubPix (src_img, dst_img, center);
```

```
// (4)결과를 표시한다
```

```
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
```

```
cvShowImage ("src", src_img);
```

```
cvShowImage ("dst", dst_img);
```

```
cvWaitKey (0);
```

```
cvDestroyWindow ("src");
```

```
cvDestroyWindow ("dst");
```

```
cvReleaseImage (&src_img);
```

```
cvReleaseImage (&dst_img);
```

```
return 1;
```

```
}
```

// (1)이미지의 읽어들이, 출력용 이미지 영역의 확보를 행한다

커멘트 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들이는다.또, 출력용 이미지의 영역을, 입력 이미지를 카피하는 것으로 확보한다.

// (2)dst\_img 의 이미지 중심이 된다 src\_img 안의 위치 center(을)를 지정한다

이 샘플에서는, 입력 이미지의 우하구석을 출력 이미지의 중심으로 하고 있다.

// (3)center 하지만 이미지 중심이 되도록(듯이),GetRectSubPix(을)를 이용해 이미지 전체를 시프트 시킨다

이미지를 시프트 시키는 것은, 출력 이미지(dst\_img)안의 각 픽셀치를 이하의 식을 이용해 입력 이미지중의 픽셀로부터 샘플링 하는 일로 결정물은 것이라도 있다.

$dst(x, y) = src(x + center.x - (width(dst)-1)*0.5, y + center.y - (height(dst)-1)*0.5)$

// (4)결과를 표시한다

윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

입력 이미지와 출력 이미지



[왼쪽에서] 입력 이미지, 출력 이미지

출력 이미지의 중심이 입력 이미지의 우하가 되도록(듯이) 화면이 좌상 방향으로 시프트 되고 있다. 나머지의 부분은 복제 경계 모드가 적용되고, 원이미지의 경계부의 픽셀치가 옆·세로 그리고 우하 영역에 뻗어 있다.

## 회전이동을 위한 픽셀 산프링 cvGetQuadrangleSubPix

지정한 회전 행렬을 이용해 원이미지를 회전시킨다.비정수치 좌표의 픽셀치는, 바이리니아 보간에 의해 얻을 수 있다, 또 이미지 경계의 외측에 존재하는 픽셀의 값을 얻기 위해서, 복제 경계 모드(이미지의 가장 외측의 픽셀치가 외측 무한원까지 성장하고 있으면 가정)가 사용된다.

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

int
main (int argc, char **argv)
{
    int angle = 45;
    float m[6];
    IplImage *src_img = 0, *dst_img = 0;
    CvMat M;

    // (1)이미지의 읽어들이어, 출력용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    // (2)회전을 위한 행렬(아핀 행렬) 요소를 설정해,CvMat 행렬 M(을)를 초기화한다
    m[0] = (float) (cos (angle * CV_PI / 180.));
    m[1] = (float) (-sin (angle * CV_PI / 180.));
    m[2] = src_img->width * 0.5;
    m[3] = -m[1];
    m[4] = m[0];
    m[5] = src_img->height * 0.5;
    cvInitMatHeader (&M, 2, 3, CV_32FC1, m, CV_AUTOSTEP);

    // (3)지정된 회전 행렬에 의해,GetQuadrangleSubPix(을)를 이용해 이미지 전체를 회전시킨다
    cvGetQuadrangleSubPix (src_img, dst_img, &M);

    // (4)결과를 표시한다
    cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
    cvShowImage ("src", src_img);
    cvShowImage ("dst", dst_img);
    cvWaitKey (0);
```

```

cvDestroyWindow ("src");
cvDestroyWindow ("dst");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 1;
}

```

// (1)이미지의 읽어들이어, 출력용 이미지 영역의 확보를 행한다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들이는다.또, 출력용 이미지의 영역을, 입력 이미지를 카피하는 것으로 확보한다.

// (2)회전을 위한 행렬(아핀 행렬) 요소를 설정해,CvMat 행렬 M(을)를 초기화한다  
여기서 지정하는 아핀 행렬은 다음과 같이 설정한다.

$$\text{map\_matrix} = \begin{vmatrix} A_{11} & A_{12} & b_1 \\ A_{21} & A_{22} & b_2 \end{vmatrix}$$

```

m[0] = A11 = cos(angle*π/180)
m[1] = A12 = -sin(angle*π/180)
m[2] = b1 = 회전 중심의 x 좌표
m[3] = A21 = sin(angle*π/180)
m[4] = A22 = cos(angle*π/180)
m[5] = b2 = 회전 중심의 y 좌표

```

angle 의 단위는 degree.

// (3)지정된 회전 행렬에 의해,GetQuadrangleSubPix(을)를 이용해 이미지 전체를 회전시킨다  
이미지를 회전시키는 것은, 출력 이미지(dst\_img)안의 각 픽셀치를 입력 이미지중의 픽셀로부터  
이하의 식에 따라서 샘플링 하는 일로, 결정하는 것이기도 하다.

```

dst(x, y) = src( A11x' + A12y' + b1, A21x' + A22y' + b2 )
x' = x - ( dst_img->width - 1 ) * 0.5
y' = y - ( dst_img->height - 1 ) * 0.5

```

// (4)결과를 표시한다

윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

## 실행 결과예

입력 이미지와 출력 이미지



[왼쪽에서] 입력 이미지, 출력 이미지

입력 이미지의 중심 좌표를 회전 중심으로서 45[deg]회전한 결과가 출력이 되어 있다. 출력 이미지의 픽셀로 대응점이 없는 경우(입력 이미지의 영역외)는, 복제 경계 모드가 적용되어 원이미지의 경계부의 픽셀치가 각 우방향으로 뻗어 있는 것이 확인할 수 있다.

## ■ 보간

여러가지 보간 방법을 이용해 이미지의 사이즈 변경을 실시한다.

샘플

---

## 이미지의 사이즈 변경 cvResize

지정한 출력 이미지 사이즈에 맞도록(듯이), 입력 이미지의 사이즈를 변경해 출력한다.

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img1 = 0, *dst_img2 = 0;

    // (1)이미지를 읽어들인다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;

    // (2)출력용 이미지 영역의 확보를 행한다
    dst_img1 = cvCreateImage (cvSize (src_img->width / 2, src_img->height / 2), src_img->depth, src_img->nChannels);
    dst_img2 = cvCreateImage (cvSize (src_img->width * 2, src_img->height * 2), src_img->depth, src_img->nChannels);
```

```

// (3)이미지의 사이즈 변경을 실시한다
cvResize (src_img, dst_img1, CV_INTER_NN);
cvResize (src_img, dst_img2, CV_INTER_NN);

// (4)결과를 표시한다
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("dst1", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("dst2", CV_WINDOW_AUTOSIZE);
cvShowImage ("src", src_img);
cvShowImage ("dst1", dst_img1);
cvShowImage ("dst2", dst_img2);
cvWaitKey (0);

cvDestroyWindow ("src");
cvDestroyWindow ("dst1");
cvDestroyWindow ("dst2");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img1);
cvReleaseImage (&dst_img2);

return 1;
}

```

// (1)이미지를 읽어들인다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.

// (2)출력용 이미지 영역의 확보를 행한다

입력 이미지 사이즈의 각각 1/2 배, 2 배의 사이즈를 지정한 출력 이미지 영역을 함수 cvCreateImage()그리고 확보한다.

// (3)이미지의 사이즈 변경을 실시한다

보간 방법을 지정하고, 사이즈 변경을 실시한다.

// (4)결과를 표시한다

윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

최근 인접 보간과 바이큐빅 보간에서는 이미지 축소 때에 무아레가 발생하고 있는 것을 확인할 수 있다.

보간 방법	사이즈1/2배	입력 이미지	사이즈2배
-------	---------	--------	-------



최근 인접



바이리니아



에리어



바이큐빅크



## ■ 기하 변환

아핀 변환·투시 투영 변환등의 이미지의 기하학적 변환을 실장하고 있다  
샘플

### 이미지의 아핀 변환(1) cvGetAffineTransform + cvWarpAffine

이미지상의 3 점 대응보다 아핀 변환 행렬을 계산해, 그 행렬을 이용해 이미지 전체의 아핀 변환을 실시한다.

```

#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img = 0;
    CvMat *map_matrix;
    CvPoint2D32f src_pnt[3], dst_pnt[3];

    // (1)이미지의 읽어들이, 출력용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    // (2)삼각형의 회전전과 회전 후의 대응하는 정점을 각각 세트 해
    //     cvGetAffineTransform(을)를 이용해 아핀 행렬을 요구한다
    src_pnt[0] = cvPoint2D32f (200.0, 200.0);
    src_pnt[1] = cvPoint2D32f (250.0, 200.0);
    src_pnt[2] = cvPoint2D32f (200.0, 100.0);

    dst_pnt[0] = cvPoint2D32f (300.0, 100.0);
    dst_pnt[1] = cvPoint2D32f (300.0, 50.0);
    dst_pnt[2] = cvPoint2D32f (200.0, 100.0);

    map_matrix = cvCreateMat (2, 3, CV_32FC1);
    cvGetAffineTransform (src_pnt, dst_pnt, map_matrix);

    // (3)지정된 아핀 행렬에 의해,cvWarpAffine(을)를 이용해 이미지를 회전시킨다
    cvWarpAffine (src_img, dst_img, map_matrix, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS,
cvScalarAll (0));

    // (4)결과를 표시한다
    cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
    cvShowImage ("src", src_img);
    cvShowImage ("dst", dst_img);
    cvWaitKey (0);
}

```

```

cvDestroyWindow ("src");
cvDestroyWindow ("dst");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseMat (&map_matrix);

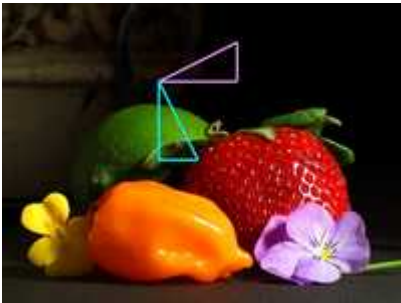
return 1;
}

```

// (1)이미지의 읽기와 출력용 이미지의 파티션

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.또, 출력용 이미지의 영역을, 입력 이미지를 카피하는 것으로 확보한다.

// (2)삼각형의 회전전과 회전 후의 대응하는 정점을 각각 세트 해 cvGetAffineTransform()(을)를 이용해 아핀 행렬을 요구한다



윗 이미지의 물색 삼각형이 핑크의 삼각형이 되도록(듯이) 회전했다고 한다.이 샘플 프로그램에서는, 삼각형의 각각의 대응점이 기존으로서 코드를 기술하고 있지만, 실제의 프로그램에서는 삼각형 닮아 대응하는 특징점의 트래킹등에 의해 이러한 점대응을 요구할 수가 있다.src\_pnt[]에 회전전의 정점 좌표를,dst\_pnt[]에 회전 후의 대응점을 세트 한다.cvGetAffineTransform()에 의해,2×3의 map\_matrix 에 요구된 아핀 행렬이 격납된다.

// (3)지정된 아핀 행렬에 의해,cvWarpAffine(을)를 이용해 이미지를 회전시킨다

변환 후 대응을 취할 수 없는 화소에 대해서는 fillval(으)로서 cvScalarAll(0) (을)를 지정해, 쿠로에서 묻는다.아핀 변환을 이용하면, 이미지의 확대·축소, 병진·회전을 행할 수가 있다.

// (4)결과를 표시한다

윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

입력 이미지와 출력 이미지



[왼쪽에서] 입력 이미지, 출력 이미지

## 이미지의 아핀 변환(2) cv2DRotationMatrix + cvWarpAffine

회전 각도, 스케일 계수, 회전 중심을 주어 변환 행렬을 계산해, 그 행렬을 이용해 이미지의 ROI 부분의 아핀 변환을 실시한다.cvWarpAffine()하 위에서 나타내 보였다 cvGetQuadrangleSubPix() 에 유사한 기능을 가지고 있지만, 입출력의 이미지 형식이 같을이라고 하는 제약을 가져 오버헤드가 크다.그러나,ROI(을)를 이용해 이미지의 일부를 변환할 수가 있다.

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    double angle = -45.0, scale = 1.0;
    IplImage *src_img = 0, *dst_img = 0;
    CvMat *map_matrix;
    CvPoint2D32f center;
    CvPoint pt1, pt2;
    CvRect rect;

    // (1)이미지의 읽어들이, 출력용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;

    rect.x = (int) (src_img->width * 0.25);
    rect.y = (int) (src_img->height * 0.25);
    rect.width = (int) (src_img->width * 0.5);
    rect.height = (int) (src_img->height * 0.5);
    cvSetImageROI (src_img, rect);

    dst_img = cvCloneImage (src_img);

    // (2)각도, 스케일 계수, 회전 중심을 지정해
    // cv2DRotationMatrix(을)를 이용해 아핀 행렬을 요구한다
    map_matrix = cvCreateMat (2, 3, CV_32FC1);
    center = cvPoint2D32f (src_img->width * 0.25, src_img->height * 0.25);
    cv2DRotationMatrix (center, angle, scale, map_matrix);

    // (3)지정된 아핀 행렬에 의해,cvWarpAffine(을)를 이용해 이미지의 ROI 부분을 회전시킨다
```

```

cvWarpAffine (src_img, dst_img, map_matrix, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS,
cvScalarAll (255));

// (4)결과를 표시한다
cvResetImageROI (src_img);
cvResetImageROI (dst_img);

pt1 = cvPoint (rect.x, rect.y);
pt2 = cvPoint (rect.x + rect.width, rect.y + rect.height);
cvRectangle (src_img, pt1, pt2, CV_RGB (255, 0, 255), 2, 8, 0);

cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
cvShowImage ("src", src_img);
cvShowImage ("dst", dst_img);
cvWaitKey (0);

cvDestroyWindow ("src");
cvDestroyWindow ("dst");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseMat (&map_matrix);

return 1;
}

```

// (1)이미지의 읽어들여, 출력용 이미지 영역의 확보를 행한다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들여, 이미지의 중앙에서, 폭·높이 모두 원이미지의 반의 영역을 지정해 ROI(을)를 세트한다.그 후, 출력용 이미지의 영역을 입력 이미지를 카피하는 것으로 확보한다.

// (2)각도, 스케일 계수, 회전 중심을 지정해 cv2DRotationMatrix(을)를 이용해 아핀 행렬을 요구한다

샘플 프로그램에서는, 시계회전에 45 번, 배율 1.0 배를 지정해,cv2DRotationMatrix()그리고 아핀 변환 행렬을 요구한다.

// (3)지정된 아핀 행렬에 의해,cvWarpaffine(을)를 이용해 이미지의 ROI 부분을 회전시킨다  
변환 후 대응을 취할 수 없는 화소에 대해서는 fillval(으)로서 cvScalarAll(255) (을)를 지정해, 흰색으로 묻는다.ROI 하지만 지정되어 있는 경우, 지정 영역만을 변환하고 있다.

// (4)결과를 표시한다

결과 표시 전에 ROI(을)를 해제하는 것으로, 이미지 전체를 표시 할 수 있도록 한다.그 후, 윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.



## 실행 결과예

입력 이미지와 출력 이미지



[왼쪽에서] 입력 이미지, 출력 이미지

ROI 그리고 지정한 영역만, 변환되어 그 외의 부분으로 변경이 없는 것을 확인할 수 있다.

## 이미지의 투시 투영 변환 cvGetPerspectiveTransform + cvWarpPerspective

이미지상의 4 점 대응보다 투시 투영 변환 행렬을 계산해, 그 행렬을 이용해 이미지 전체의 투시 투영 변환을 실시한다.

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img = 0;
    CvMat *map_matrix;
    CvPoint2D32f src_pnt[4], dst_pnt[4];

    // (1)이미지의 읽어들이, 출력용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    // (2)사각형의 변환전과 변환 후의 대응하는 정점을 각각 세트 해
    //     cvWarpPerspective(을)를 이용해 투시 투영 변환 행렬을 요구한다
    src_pnt[0] = cvPoint2D32f (150.0, 150.0);
    src_pnt[1] = cvPoint2D32f (150.0, 300.0);
```

```

src_pnt[2] = cvPoint2D32f (350.0, 300.0);
src_pnt[3] = cvPoint2D32f (350.0, 150.0);

dst_pnt[0] = cvPoint2D32f (200.0, 200.0);
dst_pnt[1] = cvPoint2D32f (150.0, 300.0);
dst_pnt[2] = cvPoint2D32f (350.0, 300.0);
dst_pnt[3] = cvPoint2D32f (300.0, 200.0);

map_matrix = cvCreateMat (3, 3, CV_32FC1);
cvGetPerspectiveTransform (src_pnt, dst_pnt, map_matrix);

// (3)지정된 투시 투영 변환 행렬에 의해,cvWarpPerspective(을)를 이용해 이미지를
변환시킨다
cvWarpPerspective (src_img, dst_img, map_matrix, CV_INTER_LINEAR +
CV_WARP_FILL_OUTLIERS, cvScalarAll (100));

// (4)결과를 표시한다
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
cvShowImage ("src", src_img);
cvShowImage ("dst", dst_img);
cvWaitKey (0);

cvDestroyWindow ("src");
cvDestroyWindow ("dst");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseMat (&map_matrix);

return 1;
}

```

// (1)이미지의 읽어들이, 출력용 이미지 영역의 확보를 행한다  
 커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고  
 읽어들이는다.또, 출력용 이미지의 영역을, 입력 이미지를 카피하는 것으로 확보한다.

// (2)사각형의 변환전과 변환 후의 대응하는 정점을 각각 세트  
 해 cvGetPerspectiveTransform()(을)를 이용해 투시 투영 변환 행렬을 요구한다





윗 이미지의 물색 사각형이 핑크의 사각형이 되도록(듯이)(샘플 코드에서는, 각각의 사각형의 저변은 같다고 하고 있다) 이미지를 변환시킨다.이 샘플 프로그램에서는, 사각형의 각각의 대응점이 기존으로서 코드를 기술하고 있지만, 실제의 프로그램에서는 특징점추출등에 의해 대응점을 요구해 행렬을 작성하거나 카메라 calibration matrices 등에서 직접 투시 투영 변환 행렬을 요구한다.

src\_pnt[]에 변환전의 좌표를,dst\_pnt[]에 대응하는 변환 후의 좌표를 세트 해,cvGetPerspectiveTransform() (을)를 부르는 일에 의해서,3×3 의 map\_matrix 에 투시 투영 변환 행렬이 격납되어 돌려주어진다.

// (3)지정된 투시 투영 변환 행렬에 의해,cvWarpPerspective(을)를 이용해 이미지를 변환시킨다  
변환 후 대응을 취할 수 없는 화소에 대해서는 fillval(으)로서 cvScalarAll(100) (을)를 지정해,  
그레이로 묻고 있다.

// (4)결과를 표시한다  
윈도우를 생성해, 입력 이미지, 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

입력 이미지와 출력 이미지



[왼쪽에서] 입력 이미지, 출력 이미지

---

## 전방위 이미지의 투시 투영 변환 cvGetPerspectiveTransform + cvWarpPerspective

전방위 이미지의 일부에 대해서 투시 투영 변환을 실시한다.

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <ctype.h>

int
main (int argc, char **argv)
{
    IplImage *src = NULL, *dst = NULL;
    CvPoint2D32f src_pnt[4], dst_pnt[4];
    CvMat *map_matrix;

    int i;
```

```

double t;
double a = 29.0, b = 40.0, c = sqrt (a * a + b * b); /* 밀러 파라미터 */
double f = 100;
int xc = 343, yc = 258; /* 이미지 중심 */
int x[4] = { -30, 30, 30, -30 };
int y[4] = { -50, -50, -50, -50 };
int z[4] = { 180, 180, 100, 100 };
int xx[4], yy[4];

// (1) 파일로부터 이미지를 읽어들인다
src = cvLoadImage ("src.jpg", CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
if (src == NULL)
    exit (-1);

dst = cvCloneImage (src);

// (2) 윈도우를 작성한다
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);

// (3) 대응점을 계산한다
for (i = 0; i < 4; i++) {
    t = -a * a * f / ((b * b + c * c) * z[i] - 2 * b * c * sqrt (x[i] * x[i] + y[i] *
y[i] + z[i] * z[i]));
    xx[i] = (int) (x[i] * t + xc + 0.5);
    yy[i] = (int) (y[i] * t + yc + 0.5);
    src_pnt[i] = cvPoint2D32f (xx[i], yy[i]);
}

// (4) 출력 이미지의 대응점을 지정
dst_pnt[0] = cvPoint2D32f (0.0, 0.0);
dst_pnt[1] = cvPoint2D32f (dst->width, 0.0);
dst_pnt[2] = cvPoint2D32f (dst->width, dst->height);
dst_pnt[3] = cvPoint2D32f (0.0, dst->height);

// (5) 투시 투영 변환을 실행한다
map_matrix = cvCreateMat (3, 3, CV_32FC1);
cvGetPerspectiveTransform (src_pnt, dst_pnt, map_matrix);
cvWarpPerspective (src, dst, map_matrix, CV_INTER_LINEAR + CV_WARP_FILL_OUTLIERS,
cvScalarAll (0));

// (6) 전개 영역을 그리기 한다
for (i = 0; i < 4; i++) {
    cvLine (src, cvPoint (xx[i], yy[i]), cvPoint (xx[(i + 1) % 4], yy[(i + 1) % 4]),
CV_RGB (255, 255, 0), 1, 0, 0);
}

```

```

// (7) 입출력 이미지를 표시한다
cvShowImage ("src", src);
cvShowImage ("dst", dst);

cvWaitKey (0);
cvDestroyWindow ("src");
cvDestroyWindow ("dst");
cvReleaseImage (&src);

return 0;
}

```

// (1) 파일로부터 이미지를 읽어들인다

입력 이미지(전방위 이미지)을 함수 cvLoadImage()그리고 읽어들인다. 또 출력 이미지의 영역을 함수 cvCloneImage ()그리고 확보한다.

// (2) 윈도우를 작성한다

입력 이미지와 출력 이미지를 표시하기 위한 윈도우를 작성한다.

// (3) 대응점을 계산한다

삼차원 좌표의 4 정점으로부터 이미지 좌표의 4 정점을 이하의 계산식에 의해 요구해 배열 src\_pnt 에 격납한다.

여기서  $(x_1, y_1, z_1)$  (은)는 삼차원 좌표의 점,  $(x_2, y_2, z_2)$  (은)는 이미지 좌표계의 점,

$(x_c, y_c, z_c)$  (은)는 이미지의 중심 좌표,  $(x_s, y_s, z_s)$  (은)는 쌍곡면 밀러의 파라미터로,

$(x_s, y_s, z_s)$  (을)를 채운다.  $(x_c, y_c, z_c)$  (은)는 촛점거리이지만 여기에서는 적당하게 지정해 있다.

// (4) 출력 이미지의 대응점을 지정

src\_pnt 에 대응하는 출력 이미지의 좌표를 배열 dst\_pnt 에 격납한다.

// (5) 투시 투영 변환을 실행한다

투시 투영 변환을 실행한다.

// (6) 전개 영역을 그리기 한다

입력 이미지에 전개 영역을 그리기 한다.

## 실행 결과예

입력 이미지와 출력 이미지



## 모르포로지 변환

구조 요소를 지정하고, 여러가지 모르포로지 연산을 행한다.

## 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

/* 메인 프로그램 */
int
main (int argc, char **argv)
{
    IplImage *src_img = 0;
    IplImage *dst_img_dilate, *dst_img_erode;
    IplImage *dst_img_opening, *dst_img_closing;
    IplImage *dst_img_gradient, *dst_img_tophat, *dst_img_blackhat;
    IplImage *tmp_img;
    IplConvKernel *element;

    //(1)이미지의 읽어들이, 연산 결과 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);
    dst_img_dilate = cvCloneImage (src_img);
    dst_img_erode = cvCloneImage (src_img);
    dst_img_opening = cvCloneImage (src_img);
    dst_img_closing = cvCloneImage (src_img);
    dst_img_gradient = cvCloneImage (src_img);
    dst_img_tophat = cvCloneImage (src_img);
    dst_img_blackhat = cvCloneImage (src_img);
    tmp_img = cvCloneImage (src_img);

    //(2)구조 요소를 생성한다
```

```
element = cvCreateStructuringElementEx (9, 9, 4, 4, CV_SHAPE_RECT, NULL);
```

```
//(3)각종의 모르포로지 연산을 실행한다
```

```
cvDilate (src_img, dst_img_dilate, element, 1);
```

```
cvErode (src_img, dst_img_erode, element, 1);
```

```
cvMorphologyEx (src_img, dst_img_opening, tmp_img, element, CV_MOP_OPEN, 1);
```

```
cvMorphologyEx (src_img, dst_img_closing, tmp_img, element, CV_MOP_CLOSE, 1);
```

```
cvMorphologyEx (src_img, dst_img_gradient, tmp_img, element, CV_MOP_GRADIENT, 1);
```

```
cvMorphologyEx (src_img, dst_img_tophat, tmp_img, element, CV_MOP_TOPHAT, 1);
```

```
cvMorphologyEx (src_img, dst_img_blackhat, tmp_img, element, CV_MOP_BLACKHAT, 1);
```

```
//(4)모르포로지 연산 결과를 표시한다
```

```
cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("dilate", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("erode", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("opening", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("closing", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("gradient", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("tophat", CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow ("blackhat", CV_WINDOW_AUTOSIZE);
```

```
cvShowImage ("src", src_img);
```

```
cvShowImage ("dilate", dst_img_dilate);
```

```
cvShowImage ("erode", dst_img_erode);
```

```
cvShowImage ("opening", dst_img_opening);
```

```
cvShowImage ("closing", dst_img_closing);
```

```
cvShowImage ("gradient", dst_img_gradient);
```

```
cvShowImage ("tophat", dst_img_tophat);
```

```
cvShowImage ("blackhat", dst_img_blackhat);
```

```
cvWaitKey (0);
```

```
cvDestroyWindow ("src");
```

```
cvDestroyWindow ("dilate");
```

```
cvDestroyWindow ("erode");
```

```
cvDestroyWindow ("opening");
```

```
cvDestroyWindow ("closing");
```

```
cvDestroyWindow ("gradient");
```

```
cvDestroyWindow ("tophat");
```

```
cvDestroyWindow ("blackhat");
```

```
cvReleaseImage (&src_img);
```

```
cvReleaseImage (&dst_img_dilate);
```

```
cvReleaseImage (&dst_img_erode);
```

```
cvReleaseImage (&dst_img_opening);
```

```
cvReleaseImage (&dst_img_closing);
```

```
cvReleaseImage (&dst_img_gradient);
```

```
cvReleaseImage (&dst_img_tophat);
```

```

cvReleaseImage (&dst_img_blackhat);
cvReleaseImage (&tmp_img);

return 1;
}

```

// (1)이미지의 읽어들이어, 연산 결과 이미지 영역의 확보를 행한다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들이는다.또, 출력용 이미지의 영역을, 입력 이미지를 카피하는 것으로 확보한다.고도의 모르포로지 연산 cvMorphologyEx()(을)를 행하는 경우는 텐포라리의 영역이 필요하다(모르포로지 구배때와 인프레이스모드로의 톱 하트 변환 (와)과 블랙 하트 변환의 경우)의로,tmp\_img(으)로서 준비해 있다.

// (2)구조 요소를 생성한다

모르포로지 연산에 사용하는 구조 요소를 생성한다.샘플 프로그램에서는,9×9 의 구형으로, 정확히 그 구형의 중심으로 앵커 포인트를 가지는 구조 요소를 지정해 있다.

// (3)각종의 모르포로지 연산을 실행한다

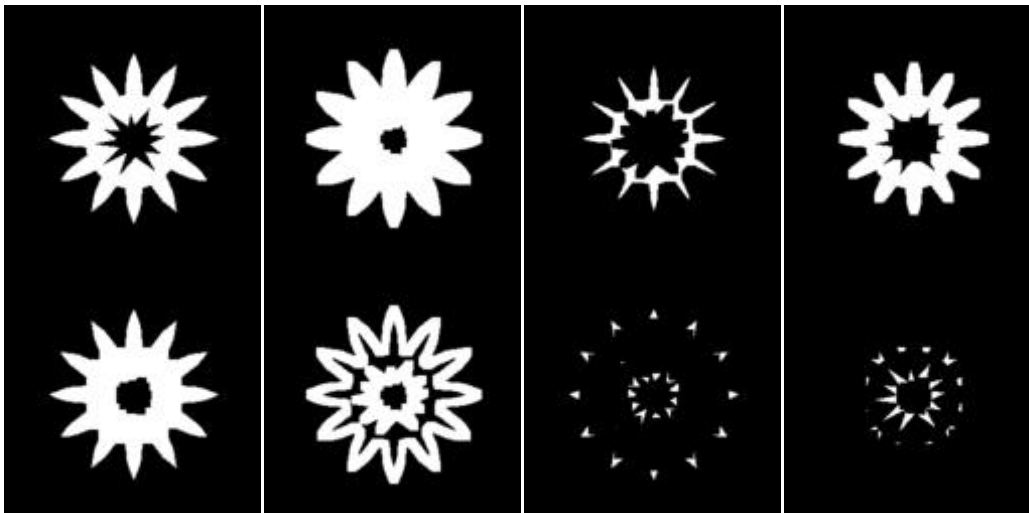
OpenCV 에 준비 떠날 수 있어 모든 모르포로지 연산의 실행을 행해, 결과를 각각의 출력 영역에 보존한다.

// (4)모르포로지 연산 결과를 표시한다

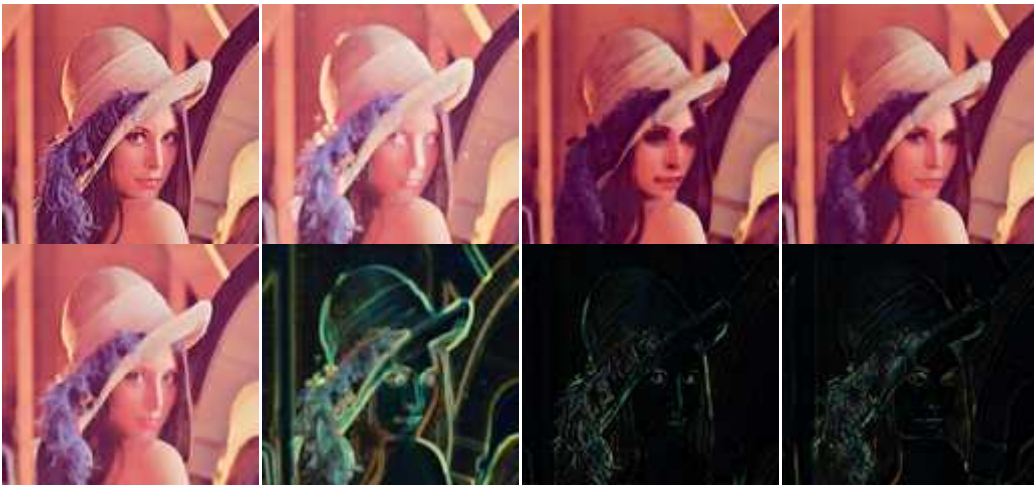
윈도우를 생성해, 각종 모르포로지 검출 결과를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

입력 이미지와 연산 결과



[상단왼쪽에서] 입력 이미지(2 값이미지),Dilation,Erosion,Opening  
[하단왼쪽에서]Closing, 경사, 톱 하트 변환, 블랙 하트 변환



[상단왼쪽에서] 입력 이미지(칼라 이미지),Dilation,Erosion,Opening  
[하단왼쪽에서]Closing, 경사, 톱 하트 변환, 블랙 하트 변환

## 평활화 cvSmooth

브라-, 가우시안, 미디언, 쌍방, 의 각 필터에 의한 평활화

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i;
    IplImage *src_img = 0, *dst_img[4];

    // (1)이미지를 읽어들인다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);

    for (i = 0; i < 4; i++)
        dst_img[i] = cvCloneImage (src_img);

    // (2)수법을 지정해 이미지를 평활화
    cvSmooth (src_img, dst_img[0], CV_BLUR, 5, 0, 0, 0);
    cvSmooth (src_img, dst_img[1], CV_GAUSSIAN, 11, 0, 0, 0);
    cvSmooth (src_img, dst_img[2], CV_MEDIAN, 5, 0, 0, 0);
    cvSmooth (src_img, dst_img[3], CV_BILATERAL, 80, 80, 0, 0);
```

```
// (3)처리된 이미지를 실제로 표시
cvNamedWindow ("Blur", CV_WINDOW_AUTOSIZE);
cvShowImage ("Blur", dst_img[0]);
cvNamedWindow ("Gaussian", CV_WINDOW_AUTOSIZE);
cvShowImage ("Gaussian", dst_img[1]);
cvNamedWindow ("Median", CV_WINDOW_AUTOSIZE);
cvShowImage ("Median", dst_img[2]);
cvNamedWindow ("Bilateral", CV_WINDOW_AUTOSIZE);
cvShowImage ("Bilateral", dst_img[3]);
cvWaitKey (0);

cvDestroyWindow ("Blur");
cvDestroyWindow ("Gaussian");
cvDestroyWindow ("Median");
cvDestroyWindow ("Bilateral");
cvReleaseImage (&src_img);
for (i = 0; i < 4; i++) {
    cvReleaseImage (&dst_img[i]);
}

return 0;
}
```

// (1)이미지를 읽어들인다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다.

// (2)수법을 지정해 이미지를 평활화

함수 cvSmooth()의 3 번째의 인수로,

CV\_BLUR(브라피르타),CV\_GAUSSIAN(가우시안피르타),CV\_MEDIAN(미디언

필터),CV\_BILATERAL(쌍방 필터) 의 각 수법을 지정하고 평활화를 실시한다. 4 번째 이후의 인수의 의미는, 수법마다 다르다.자세한 것은, 레퍼런스 메뉴얼을 참조하는 것.

// (3)처리된 이미지를 실제로 표시

각 수법으로 평활화된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

하단의 이미지는, 처리 이미지의 일부를 확대한 것이다. 다만, 수법마다 파라미터를 바꾸어 있으므로, 단순하게 비교 할 수 있는 이미지는 아니다.







## 유저 정의 필터 cvFilter2D

유저가 정의한 커널에 의한 필터링

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img;
    float data[] = { 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
};
CvMat kernel = cvMat (1, 21, CV_32F, data);

// (1)이미지의 읽기
if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
if (src_img == 0)
    exit (-1);

dst_img = cvCreateImage (cvGetSize (src_img), src_img->depth, src_img->nChannels);

// (2)커널의 정규화와 필터 처리
cvNormalize (&kernel, &kernel, 1.0, 0, CV_L1);
cvFilter2D (src_img, dst_img, &kernel, cvPoint (0, 0));

// (3)처리 이미지의 표시
cvNamedWindow ("Filter2D", CV_WINDOW_AUTOSIZE);
cvShowImage ("Filter2D", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Filter2D");
```

```

cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다.

// (2)커널의 정규화와 필터 처리

부동 소수점형 배열 data 의 값을 각 요소로 하는,  $1 \times 21$ 의 행렬을 커널로 하는 필터 처리를 실시한다. 함수 cvNormalize()에 의해서, 커널을 정규화한다. 즉, 여기에서는, 절대치의 법칙이 "1.0"(이)가 되도록(듯이), 배열의 값을 정규화하고 있다. 그리고, 그 커널을 이용(함수 cvFilter2D()의 3 번째의 인수로 지정) 하고, 필터 처리를 실시한다.

함수 cvFilter2D()의 4 번째의 인수는, 필터 대상 픽셀의 커널내에서의 상대 위치를 나타내고 있어 디폴트는 cvPoint(-1,-1)(이어)여 커널 중심을 가리키지만, 이번은, cvPoint(0,0)(을)를 지정하는 것으로, 커널의 좌단을 가리키고 있다.

// (3)처리 이미지의 표시

처리된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



#### ■ 경계선

이미지의 경계를 어떻게 취급할까는, 많은 이미지 처리에 대해 자주 문제가 된다. 예를 들면, 필터 처리에 대하고, 필터가 부분적으로 이미지의 밖에는 보기 시작하고 해 기다렸을 경우, 어떠한 화소치를 가정해 이용할까에 의해서, 처리 결과가 다르다. OpenCV그림, 이미지를 카피해, 그 주위에 경계선을 붙이는 처리를 실시하는 함수가 용 뜻 되고 있어 IPL\_BORDER\_CONSTANT, 및 IPL\_BORDER\_REPLICAT(을)를 서포트하고 있다. OpenCV그리고 이용되는 함수는, IPL\_BORDER\_REPLICAT, 즉 복제 경계 모드를 이용하는 것이 많이 있지만, 유저가 그러한 처리를 바라지 않은 경우에는, 미리 경계를 정수치로 묻고 나서 처리를 실시해, 처리 후의 이미지를 클리핑 하는 일로 경계의 취급을 컨트롤 할 수 있다.

샘플

#### 경계선의 작성 cvCopyMakeBorder

이미지의 카피와 경계의 작성

```
#include <cxcore.h>
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, offset = 30;
    IplImage *src_img = 0, *dst_img[2];

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);

    for (i = 0; i < 2; i++)
        dst_img[i] =
            cvCreateImage (cvSize (src_img->width + offset * 2, src_img->height + offset * 2),
src_img->depth,
                        src_img->nChannels);

    // (2)경계선의 작성
    cvCopyMakeBorder (src_img, dst_img[0], cvPoint (offset, offset), IPL_BORDER_REPLICATE);
    cvCopyMakeBorder (src_img, dst_img[1], cvPoint (offset, offset), IPL_BORDER_CONSTANT,
CV_RGB (255, 0, 0));

    // (3)이미지의 표시
    cvNamedWindow ("Border_replicate", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Border_replicate", dst_img[0]);
    cvNamedWindow ("Border_constant", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Border_constant", dst_img[1]);
    cvWaitKey (0);

    cvDestroyWindow ("Border_replicate");
    cvDestroyWindow ("Border_constant");
    cvReleaseImage (&src_img);
    for (i = 0; i < 2; i++) {
        cvReleaseImage (&dst_img[i]);
    }
}
```

```
return 0;
}
```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다. 또, 경계를 작성하기 위해서, 입력 이미지보다 큰 출력 영역을 확보한다.

// (2)경계선의 작성

함수 cvCopyMakeBorder()에 의해서, 입력 이미지를 출력 이미지에 카피해, 한층 더 경계를 작성한다. 3 번째의 인수는, 출력 이미지에 카피되는 입력 이미지의 좌상 좌표를 나타내고 있어 이미지가 카피되지 않는 부분(이번은, 상하 좌우의 구석으로부터 offset 픽셀분 )이 경계 영역이 된다. 4 번째의 인수는, 경계의 종류를 나타내고 있어 IPL\_BORDER\_REPLICATE 하지만 지정되면, 이미지위/아래의 구석과 왼쪽/오른쪽의 구석(이미지 영역의 제일 외측의 값)을 이용해 경계선을 생성한다. 또, 4 번째의 인수에, IPL\_BORDER\_CONSTANT 하지만 지정되면, 경계는 이 함수의 최후(5 번째 )의 파라미터로서 건네받은 정수(이번은 적색)로 묻힌다.

// (3)이미지의 표시

처리된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



#### ■ 반응을 일으키는 최소의 물리량 처리

이미지에 있는 반응을 일으키는 최소의 물리량을 마련하고, 화소치가 그 값보다 큰가 작은 것처럼 따르고 처리를 바꾸고 싶다고 하는 일은 자주 있다. 예를 들면, 단순한 배경 차분에서는, 현이미지와 배경 이미지와의 화소치의 차이의 절대치가 있는 반응을 일으키는 최소의 물리량 이하가 되는지 아닌지로, 배경인가 전경인지를 판단한다. OpenCV 그림, 이러한 반응을 일으키는 최소의 물리량 처리를 행하기 위한 함수, cvThreshold, 및 cvAdaptiveThreshold (이)가 준비되어 있다.

샘플

## 이미지의 2 치화 cvThreshold, cvAdaptiveThreshold

cvThreshold, cvAdaptiveThreshold(을)를 이용하고, 이미지의 2 치화를 실시한다

#### 샘플 코드

표시의 변환

```
#include <cv.h>
```

```

#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img;
    IplImage *src_img_gray = 0;
    IplImage *tmp_img1, *tmp_img2, *tmp_img3;

    // (1) 이미지를 읽어들인다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
    if (src_img == 0)
        return -1;

    tmp_img1 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    tmp_img2 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    tmp_img3 = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    src_img_gray = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    cvCvtColor (src_img, src_img_gray, CV_BGR2GRAY);
    dst_img = cvCloneImage (src_img);

    // (2) 가우시안 피르타로 평활화를 실시한다
    cvSmooth (src_img_gray, src_img_gray, CV_GAUSSIAN, 5);

    // (3) 2 치화: cvThreshold
    cvThreshold (src_img_gray, tmp_img1, 90, 255, CV_THRESH_BINARY);

    // (4) 2 치화: cvAdaptiveThreshold
    cvAdaptiveThreshold (src_img_gray, tmp_img2, 255, CV_ADAPTIVE_THRESH_MEAN_C,
        CV_THRESH_BINARY, 11, 10);

    // (5) 두 개의 2 치화 이미지의 논리적
    cvAnd (tmp_img1, tmp_img2, tmp_img3);
    cvCvtColor (tmp_img3, dst_img, CV_GRAY2BGR);

    // (6) 원 이미지와 2 치 이미지의 논리적
    cvSmooth (src_img, src_img, CV_GAUSSIAN, 11);
    cvAnd (dst_img, src_img, dst_img);

    // (7) 이미지를 표시한다
    cvNamedWindow ("Threshold", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Threshold", tmp_img1);
    cvNamedWindow ("AdaptiveThreshold", CV_WINDOW_AUTOSIZE);
    cvShowImage ("AdaptiveThreshold", tmp_img2);
    cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);

```

```

cvShowImage ("Image", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Threshold");
cvDestroyWindow ("AdaptiveThreshold");
cvDestroyWindow ("Image");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseImage (&src_img_gray);
cvReleaseImage (&tmp_img1);
cvReleaseImage (&tmp_img2);
cvReleaseImage (&tmp_img3);

return 0;
}

```

// (1)이미지를 읽어들인다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 또, 2 치화를 실시하기 위해서 입력 이미지를 그레이 스케일로 변환한다.

// (2)가우시안피르타로 평활화를 실시한다

2 치화를 실시하기 전에, 가우시안피르타로 평활화를 실시한다.평활화를 실시하는 것에 의해서, 이미지의 노이즈를 감소시켜, 안정된(노이즈가 적다) 2 치화 이미지를 얻을 수 있다.

// (3)2 치화:cvThreshold

함수 cvThreshold()에 의해서, 이미지의 2 치화 처리를 실시한다. 이 함수의 마지막 인수가, 반응을 일으키는 최소의 물리량 처리의 종류를 나타내고 있다. CV\_THRESH\_BINARY 하지만 지정되어 있는 경우는,

$src(x,y) > threshold$ 의 경우는,  $dst(x,y) = max\_value$   
그 이외는, 0

되도록(듯이) 처리를 실시한다.여기서,threshold(반응을 일으키는 최소의 물리량)(은)는3번째의 인수로, max\_value(최대치)(은)는4번째의 인수로, 각각 지정되어 있다. 그 외의 수법을 지정했을 경우의 처리에 대해서는, 레퍼런스 메뉴얼을 참조.

// (4)2 치화:cvAdaptiveThreshold

함수 cvAdaptiveThreshold()에 의해서, 이미지의 2 치화 처리를 실시한다. 함수 cvThreshold()그럼, 모든 픽셀에 대해 같은 반응을 일으키는 최소의 물리량을 가지고 처리를 실시했지만, 함수 cvAdaptiveThreshold()의 경우는, 픽셀마다 사용하는 반응을 일으키는 최소의 물리량이 다르다. 이 함수의 5 번째의 인수가, 함수 cvThreshold(와)과 같게 반응을 일으키는 최소의 물리량 처리의 종류를 나타내고 있다. CV\_ADAPTIVE\_THRESH\_BINARY 하지만 지정되어 있는 경우는,

$src(x,y) > T(x,y)$ 의 경우는,  $dst(x,y) = max\_value$   
그 이외는, 0

되도록(듯이) 처리를 실시한다.여기서,max\_value(최대치)(은)는3번째의 인수로 지정되어 있다. 또,T(x,y)(은)는, 각 픽셀마다 계산된 반응을 일으키는 최소의 물리량이며,4번째의 인수로, 의 반응을 일으키는 최소의 물리량의 산출 방법이 지정된다. CV\_ADAPTIVE\_THRESH\_MEAN\_C 하지만 지정되었을 경우는, 주목 픽셀의 block\_size×block\_size 인접 영역

의 평균으로부터,param1 (을)를 당긴 값이 되어, block\_size하6번째의 인수로,param1하7번째의 인수로, 각각 지정된다.

// (5)두 개의 2 치화 이미지의 논리적

두 개의 함수에 의해서 2 치화 된 이미지의 논리적을 계산한다. 즉, 2 종류의 2 치화 이미지를 합성해,3 채널 이미지로 변환한다.

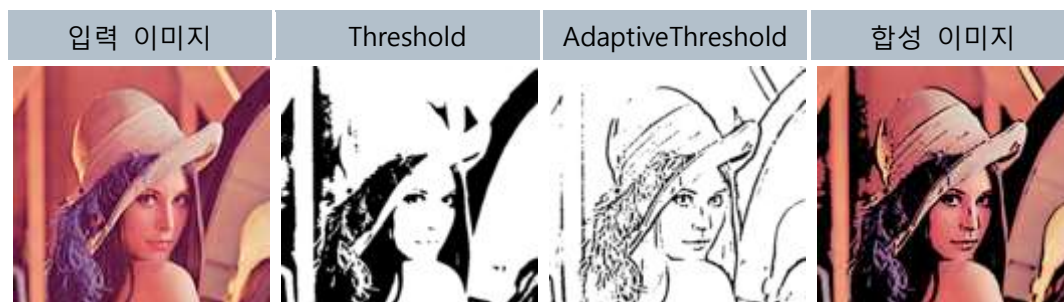
// (6)원이미지와 2 치 이미지의 논리적

(평활화한) 입력 이미지와 2 치화 이미지의 요소 마다의 논리적을 계산한다. 이것에 의해, 입력 이미지상에 2 치화 이미지가 합성된 이미지를 얻는다.

// (7)이미지를 표시한다

각각의 2 치화 이미지, 및 입력 이미지와 거듭해 맞춘 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예



## 이미지의 2 치화(오츠의 수법) cvThreshold

오츠의 수법을 이용하고 반응을 일으키는 최소의 물리량을 결정해, 이미지의 2 치화를 실시한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img;

    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE);
    if (src_img == 0)
        return -1;

    dst_img = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
```



```

cvSmooth (src_img, src_img, CV_GAUSSIAN, 5);

// (1)2 치화(�츠의 수법을 이용)
cvThreshold (src_img, dst_img, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);

cvNamedWindow ("Threshold", CV_WINDOW_AUTOSIZE);
cvShowImage ("Threshold", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Threshold");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 0;
}

```

// (1)2 치화(�츠의 수법을 이용)

전술의 2 치화와 다른 것은, 마지막 인수(threshold type)에, CV\_THRESH\_BINARY | CV\_THRESH\_OTSU 하지만 더해지고 있는 점이다. OpenCV-1.0.0 시점에서의 레퍼런스 메뉴얼에는 기술이 없다(CVS 판에는 있다)가, 함수 cvThreshold() (은)는, �츠의 수법으로 불리는 반응을 일으키는 최소의 물리량 결정 수법을 실장하고 있다. 여기서 이용되는 �츠의 수법이란, 어느 값의 집합을 두 개 클래스로 분류하는 경우의, 적절한 반응을 일으키는 최소의 물리량을 결정하는 수법이다. 두 개의 클래스내의 분산과 클래스간의 분산을 생각해 이러한 비가 최소에(즉, 클래스 비밀산은 가능한 한 작고, 클래스간분산은 가능한 한 크고) 되는 반응을 일으키는 최소의 물리량을 요구한다. 따라서, 3 번째의 인수(threshold)에게 주는 값은 이용되지 않기 때문에, 적당한 값을 지정해 좋다.

자세한 것은, 이하의 문헌을 참고로 하는 것.

- �츠, "판별 및 최소 2 승기준에 근거하는 자동 해 귀의치 선정법", 전자 통신 학회 논문잡지, Vol. J63-D, No.4, pp.349-356, 1980.
- N. Otsu, "A threshold selection method from gray level histograms", IEEE Trans. Systems, Man and Cybernetics, 1979, Vol.9, pp.62-66

#### 실행 결과예



#### 이미지 피라미드의 작성 cvPyrDown, cvPyrUp

입력 이미지로부터, 피라미드의 상하의 층 이미지를 작성한다



```

#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img, *dst_img1, *dst_img2;

    if (argc != 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH |
CV_LOAD_IMAGE_ANYCOLOR)) == 0)
        return -1;

    dst_img1 = cvCreateImage (cvSize (src_img->width / 2, src_img->height / 2), src_img-
>depth, src_img->nChannels);
    dst_img2 = cvCreateImage (cvSize (src_img->width * 2, src_img->height * 2), src_img-
>depth, src_img->nChannels);

    // (1)입력 이미지에 대한 이미지 피라미드를 구성
    cvPyrDown (src_img, dst_img1, CV_GAUSSIAN_5x5);
    cvPyrUp (src_img, dst_img2, CV_GAUSSIAN_5x5);

    cvNamedWindow ("Original", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Original", src_img);
    cvNamedWindow ("PyrDown", CV_WINDOW_AUTOSIZE);
    cvShowImage ("PyrDown", dst_img1);
    cvNamedWindow ("PyrUp", CV_WINDOW_AUTOSIZE);
    cvShowImage ("PyrUp", dst_img2);
    cvWaitKey (0);

    cvDestroyWindow ("Original");
    cvDestroyWindow ("PyrDown");
    cvDestroyWindow ("PyrUp");
    cvReleaseImage (&src_img);
    cvReleaseImage (&dst_img1);
    cvReleaseImage (&dst_img2);

    return 0;
}

```

// (1)입력 이미지에 대한 이미지 피라미드를 구성

함수 cvPyrDown()에 의해서, 입력 이미지의 저해상(1/4) 번이미지를, 함수 cvPryUp()에 의해서, 고해상도 이미지(2)(을)를 구성한다. 마지막 인수, CV\_GAUSSIAN\_5x5(현재 서포트되고 있는 값은, 이것만)(을)를 지정하는 것에 의해서, 가우시안피르타에 근거한 리산프링을 한다.

## 실행 결과예



## 이미지 피라미드를 이용한 이미지의 영역 분할 cvPyrSegmentation

레벨을 지정해 이미지 피라미드를 작성해, 그 정보를 이용해 이미지의 부분화를 행한다.

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int level = 4;
    double threshold1, threshold2;
    IplImage *src_img = 0, *dst_img;
    CvMemStorage *storage = 0;
    CvSeq *comp = 0;
    CvRect roi;

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);

    // (2)영역 분할을 위해서 ROI(을)를 세트 한다
    roi.x = roi.y = 0;
    roi.width = src_img->width & ~(1 << level);
    roi.height = src_img->height & ~(1 << level);
    cvSetImageROI (src_img, roi);
```

```

// (3)분할 결과 이미지 출력용의 이미지 영역을 확보해, 영역 분할을 실행
dst_img = cvCloneImage (src_img);

storage = cvCreateMemStorage (0);
threshold1 = 255.0;
threshold2 = 50.0;

cvPyrSegmentation (src_img, dst_img, storage, &comp, level, threshold1, threshold2);

// (4)입력 이미지와 분할 결과 이미지의 표시
cvNamedWindow ("Source", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("Segmentation", CV_WINDOW_AUTOSIZE);
cvShowImage ("Source", src_img);
cvShowImage ("Segmentation", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Source");
cvDestroyWindow ("Segmentation");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.

// (2)영역 분할을 위해서 ROI(을)를 세트 한다

cvPyrSegmentation(은)는,  $2^{(\text{피라미드의 레벨})}$  그리고 나뉘어 떨어지는 이미지 사이즈 밖에 처리할 수 없다(에러가 발생한다).그 때문에, 지정한 레벨에 대응한다 ROI 사이즈를 입력 이미지에 세트 할 필요가 있다.샘플 프로그램에서는, 지정한 레벨수만큼 1 을 왼쪽으로 시프트 해( $2^{(\text{지정한 레벨})}$ (을)를 계산), 그 후 2 의 보수를 취하는 일로 비트 마스크를 작성해, 원이미지의 width, height(와)과의 AND(을)를 취해, 나뉘어 떨어지는 이미지 사이즈 (을)를 계산한다.

// (3)분할 결과 이미지 출력용의 이미지 영역을 확보해, 영역 분할을 실행

입력 이미지와 같은 사이즈, 채널수, 데프스의 출력용

이미지 dst\_img(을)를,cvCloneImage()그리고 확보해, 2 개의 반응을 일으키는 최소의 물리량을 세트 하고, 영역 분할을 실행한다.

// (4)입력 이미지와 분할 결과 이미지의 표시

윈도우를 생성해, 입력 이미지, 분할 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

## 실행 결과예

입력 이미지와 영역 분할 결과



[왼쪽에서] 입력 이미지, 영역 분할 결과(threshold2=25), 영역 분할 결과(threshold2=50), 영역 분할 결과(threshold2=75), 영역 분할 결과(threshold2=100)

## 평균치 시프트법에 따르는 이미지의 부분화 cvPyrMeanShiftFiltering

평균치 시프트법에 따르는 이미지의 부분화를 실시한다.

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int level = 2;
    IplImage *src_img = 0, *dst_img;
    CvRect roi;

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);
    if (src_img->nChannels != 3)
        exit (-1);
    if (src_img->depth != IPL_DEPTH_8U)
        exit (-1);

    // (2)영역 분할을 위해서 ROI(을)를 세트 한다
    roi.x = roi.y = 0;
    roi.width = src_img->width & ~(1 << level);
```

```

roi.height = src_img->height & -(1 << level);
cvSetImageROI (src_img, roi);

// (3)분할 결과 이미지 출력용의 이미지 영역을 확보해, 영역 분할을 실행
dst_img = cvCloneImage (src_img);

cvPyrMeanShiftFiltering (src_img, dst_img, 30.0, 30.0, level,
                        cvTermCriteria (CV_TERMCRIT_ITER + CV_TERMCRIT_EPS, 5, 1));

// (4)입력 이미지와 분할 결과 이미지의 표시
cvNamedWindow ("Source", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("MeanShift", CV_WINDOW_AUTOSIZE);
cvShowImage ("Source", src_img);
cvShowImage ("MeanShift", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Source");
cvDestroyWindow ("MeanShift");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.

// (2)영역 분할을 위해서 ROI(을)를 세트 한다

cvPyrMeanShiftFiltering 도 위에서 나타내 보였다 cvPyrSegmentation(와)과 같이, 2<sup>(피라미드의 레벨)</sup>그리고 나뉘어 떨어지는 이미지 사이즈 밖에 처리할 수 없다(에러가 발생한다). 그 때문에, 지정한 레벨에 대응한다 ROI 사이즈를 입력 이미지에 세트 할 필요가 있다. 샘플 프로그램에서는, 지정한 레벨수만큼 1 을 왼쪽으로 시프트 해(2<sup>(지정한 레벨)</sup>(을)를 계산), 그 후 2 의 보수를 취하는 일로 비트 마스크를 작성해, 원이미지의 width, height(와)과의 AND(을)를 취해, 나뉘어 떨어지는 이미지 사이즈를 계산한다.

// (3)분할 결과 이미지 출력용의 이미지 영역을 확보해, 영역 분할을 실행

입력 이미지와 같은 사이즈, 채널수, 데프스의 출력용

이미지 dst\_img(을)를,cvCloneImage()그리고 확보해, 2 개의 반응을 일으키는 최소의 물리량을 세트 하고, 영역 분할을 실행한다.

// (4)입력 이미지와 분할 결과 이미지의 표시

윈도우를 생성해, 입력 이미지, 분할 결과 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

## 실행 결과예

입력 이미지와 부분화 결과



[왼쪽에서] 입력 이미지, 부분화 결과

## Watershed 알고리즘에 의한 이미지의 영역 분할 cvWatershed

마우스로 원형의 마커(배정 영역)의 중심을 지정해, 복수의 마커를 설정한다. 이 마커를 이미지의 gradient에 따라서 넓혀 가, gradient의 높은 부분에 할 수 있는 경계를 바탕으로 영역을 분할한다. 영역은, 최초로 지정한 마커의 수로 분할된다.

이 샘플 코드는, 마우스 이벤트를 처리하기 위한 함수(on\_mouse)(을)를 포함하고 있다.

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

IplImage *markers = 0, *dsp_img = 0;

/* 마우스 이벤트용 콜백 함수 */
void
on_mouse (int event, int x, int y, int flags, void *param)
{
    int seed_rad = 20;
    static int seed_num = 0;
    CvPoint pt;

    // (1)클릭에 의해 중심을 지정해, 원형의 배정 영역을 설정한다
    if (event == CV_EVENT_LBUTTONDOWN) {
        seed_num++;
        pt = cvPoint (x, y);
        cvCircle (markers, pt, seed_rad, cvScalarAll (seed_num), CV_FILLED, 8, 0);
        cvCircle (dsp_img, pt, seed_rad, cvScalarAll (255), 3, 8, 0);
        cvShowImage ("image", dsp_img);
    }
}

/* 메인 프로그램 */
```

```

int
main (int argc, char **argv)
{
    int *idx, i, j;
    IplImage *src_img = 0, *dst_img = 0;

    // (2)이미지의 읽어들이, 마커 이미지의 초기화, 결과 표시용 이미지 영역의 확보를 행한다
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);
    dsp_img = cvCloneImage (src_img);
    dst_img = cvCloneImage (src_img);

    markers = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_32S, 1);
    cvZero (markers);

    // (3)입력 이미지를 표시해 배경 컴퍼넌트 지정을 위한 마우스 이벤트를 등록한다
    cvNamedWindow ("image", CV_WINDOW_AUTOSIZE);
    cvShowImage ("image", src_img);
    cvSetMouseCallback ("image", on_mouse, 0);
    cvWaitKey (0);

    // (4)watershed 분할을 실행한다
    cvWatershed (src_img, markers);

    // (5)실행 결과의 이미지중의 watershed 경계(픽셀치=-1)(을)를 결과 표시용 이미지상에
    표시한다
    for (i = 0; i < markers->height; i++) {
        for (j = 0; j < markers->width; j++) {
            idx = (int *) cvPtr2D (markers, i, j, NULL);
            if (*idx == -1)
                cvSet2D (dst_img, i, j, cvScalarAll (255));
        }
    }

    cvNamedWindow ("watershed transform", CV_WINDOW_AUTOSIZE);
    cvShowImage ("watershed transform", dst_img);
    cvWaitKey (0);

    cvDestroyWindow ("watershed transform");
    cvReleaseImage (&markers);
    cvReleaseImage (&dsp_img);
    cvReleaseImage (&src_img);
    cvReleaseImage (&dst_img);

```

```
return 1;
}
```

// (1)클릭에 의해 중심을 지정해, 원형의 배경 영역을 설정한다

입력 이미지를 표시한 윈도우상에서, 마우스 클릭의 이벤트가 발생하면, 배경의 수(seed\_num)(을)를 하나 늘려, 마커 이미지(markers)위에, 지정 반경에서 색을 seed\_num 인 전부 칠해 엔을 그리기 한다. 이,marker 하지만 watershed 분할의 배경이 된다. 무엇인가 키 입력을 행할 때까지, 배경을 추가할 수가 있다.

// (2)이미지의 읽어들이, 마커 이미지의 초기화, 결과 표시용 이미지 영역의 확보를 행한다  
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들이다.동시에 표시용 이미지로서 입력 이미지를 카피한다.

마커 이미지로서 32 비트 싱글 채널의 이미지 영역(markers)(을)를 확보한다. 이 마커 이미지에서는,watershed 경계가-1, 관계가 미지의 영역은 0, 그 이외에서는 속한다 영역 번호가 들어가 있다.

// (3)입력 이미지를 표시해 배경 컴퍼넌트 지정을 위한 마우스 이벤트를 등록한다

입력 이미지를 표시해, 이 윈도우상에서의 마우스 이벤트에 대한 콜백 함수(on\_mouse)(을)를 지정한다.

// (4)watershed 분할을 실행한다

마커 이미지 설정 완료의 키 입력을 기다려,cvWatershed()(을)를 실행한다.

// (5)실행 결과의 이미지중의 watershed 경계(픽셀치=-1)(을)를 결과 표시용 이미지상에 표시한다

분할 결과를 보관 유지하고 있는 마커 이미지를 스캔(cvPtr2D 그리고 지정한 배열 요소에의 포인터가 되돌아 온다) 해, 값이-1 의 픽셀에 대응하는 결과 표시용 이미지(dst\_img)의 픽셀치를 255(흰색)에 cvSet2D()(을)를 이용해 설정해, 결과를 표시한다.

### 실행 결과예

입력 이미지와 영역 분할 결과



[왼쪽에서] 입력 이미지, 입력 이미지상에 표시한 마커 영역, 영역 분할 결과

### ■ 이미지의 윤곽 검출

OpenCV그림, 윤곽은 다양한 데이터 구조에 의해 관리된다. 즉, 모든 윤곽이 단순한 리스트로 표현되기도 하면, 부모와 자식 관계를 보관 유지한 트리 구조로 표현되기도 한다. 윤곽을 이용한 처리(포함 구형, 근사, 비교)의 해설 붙어 다른 장에 양보해, 여기에서는, 윤곽의 검출과 주사에 대해 간단하게 설명한다.

샘플



## 윤곽의 검출과 그리기 cvFindContours

이미지중으로부터 윤곽을 검출해, -1~+1 까지의 레벨에 있는 윤곽을 그리기 한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char *argv[])
{
    int i;
    IplImage *src_img = 0, **dst_img;
    IplImage *src_img_gray = 0;
    IplImage *tmp_img;
    CvMemStorage *storage = cvCreateMemStorage (0);
    CvSeq *contours = 0;
    int levels = 0;

    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
    if (src_img == 0)
        return -1;

    src_img_gray = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    cvCvtColor (src_img, src_img_gray, CV_BGR2GRAY);
    tmp_img = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
    dst_img = (IplImage **) cvAlloc (sizeof (IplImage *) * 3);
    for (i = 0; i < 3; i++) {
        dst_img[i] = cvCloneImage (src_img);
    }

    // (1)이미지의 2 치화
    cvThreshold (src_img_gray, tmp_img, 120, 255, CV_THRESH_BINARY);

    // (2)윤곽의 검출
    cvFindContours (tmp_img, storage, &contours, sizeof (CvContour), CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);

    // (3)윤곽의 그리기
    cvDrawContours (dst_img[0], contours, CV_RGB (255, 0, 0), CV_RGB (0, 255, 0), levels -
1, 2, CV_AA, cvPoint (0, 0));
```

```

cvDrawContours (dst_img[1], contours, CV_RGB (255, 0, 0), CV_RGB (0, 255, 0), levels,
2, CV_AA, cvPoint (0, 0));
cvDrawContours (dst_img[2], contours, CV_RGB (255, 0, 0), CV_RGB (0, 255, 0), levels +
1, 2, CV_AA, cvPoint (0, 0));

// (4)이미지의 표시
cvNamedWindow ("Level:-1", CV_WINDOW_AUTOSIZE);
cvShowImage ("Level:-1", dst_img[0]);
cvNamedWindow ("Level:0", CV_WINDOW_AUTOSIZE);
cvShowImage ("Level:0", dst_img[1]);
cvNamedWindow ("Level:1", CV_WINDOW_AUTOSIZE);
cvShowImage ("Level:1", dst_img[2]);
cvWaitKey (0);

cvDestroyWindow ("Level:-1");
cvDestroyWindow ("Level:0");
cvDestroyWindow ("Level:1");
cvReleaseImage (&src_img);
cvReleaseImage (&src_img_gray);
cvReleaseImage (&tmp_img);
for (i = 0; i < 3; i++) {
    cvReleaseImage (&dst_img[i]);
}
cvFree (dst_img);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지의 2 치화

읽어들인 이미지를 2 치화한다.함수 cvFindContours()(은)는, 처리 대상의 이미지를 2 값이미지로서 취급한다(값이 0 이외의 픽셀은"1",0의 픽셀은"0"(으)로 한다) 유익이다.

// (2)윤곽의 검출

2 치화 된 이미지로부터, 윤곽을 검출한다.5 번째의 인수에서는, 윤곽의 추출 모드를 지정한다. 여기에서는,CV\_RETR\_TREE (을)를 지정해, 모든 윤곽을 추출해, 분기한 윤곽을 완전하게 표현하는 계층 구조를 구성한다. 또,6 번째의 인수는, 윤곽의 근사 수법을 나타내고 있어 CV\_CHAIN\_APPROX\_SIMPLE 하지만 지정되었을 경우는, 수평·수직·기울기의 선분이 압축되어 각각의 단 점만이 점렬로서 남는다. 그 외의 모드, 근사 수법에 대해서는, 레퍼런스 메뉴얼을 참조하는 것.

// (3)윤곽의 그리기

함수 cvDrawContours()(을)를 이용하고, 윤곽을 그리기 한다.이번은, 이하와 같이-1,0,1의 세 개의 레벨을 지정해 윤곽을 그리기 하고 있다.

-1 : 2 번째의 인수로 지정한 윤곽과 그 아이의 레벨 0 까지의 윤곽(즉, 하나하의 계층의 윤곽)이 그리기 된다

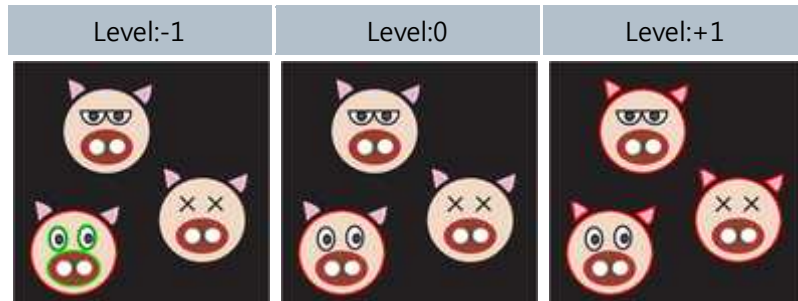
0 : 2 번째의 인수로 지정한 윤곽만이 그리기 된다

+1 : 2 번째의 인수로 지정한 윤곽과 동레벨의 윤곽(즉, 같은 계층에 있는 윤곽)이 그리기 된다

// (4)이미지의 표시

윈도우를 생성해,3 종류의 윤곽 추출 이미지를 표시하고, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예



### ■ 이미지와 형상의 모멘트

이미지나 형상의 특징을 정량적으로 표현하는 방법의 하나에 모멘트가 있다.이미지의(x\_order+y\_order)다음의 공간 모멘트는 다음과 같이 정의된다.

$$M_{x\_order,y\_order} = \sum_{x,y} (I(x,y) \cdot x^{x\_order} \cdot y^{y\_order})$$

여기서,I(x,y)(은)는 좌표(x, y)의 휘도치이며,x\_order, y\_order(은)는 각각x방향,y방향의 차수를 나타내고 있다.

2값이미지의 경우,0다음 모멘트M<sub>0,0</sub>하지만 면적을 나타내,( M<sub>1,0</sub>/M<sub>0,0</sub>, M<sub>0,1</sub>/M<sub>0,0</sub> )하지만 중심 좌표를 나타내는 일이 된다.

샘플

## 이미지의 모멘트

이미지의 각종 모멘트치를 계산한다

### 샘플 코드

표시의 변환

```
#include <stdio.h>
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    char text[10][30];
    int i;
    double spatial_moment, central_moment, norm_c_moment;
```

```

IplImage *src_img = 0;
CvFont font;
CvSize text_size;
CvMoments moments;
CvHuMoments hu_moments;

// (1)이미지를 읽어들인다.3 채널 이미지의 경우는 C01 하지만 세트 되어 있지 않으면 안 된다
if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
if (src_img == 0)
    return -1;

if (src_img->nChannels == 3 && cvGetImageC0I (src_img) == 0)
    cvSetImageC0I (src_img, 1);

// (2)입력 이미지의 3 차까지의 이미지 모멘트를 계산한다
cvMoments (src_img, &moments, 0);
cvSetImageC0I (src_img, 0);

// (3)모멘트나 Hu 모멘트 불변량을, 얻을 수 있었다 CvMoments 구조체의 값을 사용해
계산한다.
spatial_moment = cvGetSpatialMoment (&moments, 0, 0);
central_moment = cvGetCentralMoment (&moments, 0, 0);
norm_c_moment = cvGetNormalizedCentralMoment (&moments, 0, 0);
cvGetHuMoments (&moments, &hu_moments);

// (4)얻을 수 있던 모멘트나 Hu 모멘트 불변량을 문자로서 이미지에 그리기
cvInitFont (&font, CV_FONT_HERSHEY_SIMPLEX, 1.0, 1.0, 0, 2, 8);
snprintf (text[0], 30, "spatial=%.3f", spatial_moment);
snprintf (text[1], 30, "central=%.3f", central_moment);
snprintf (text[2], 30, "norm=%.3f", spatial_moment);
snprintf (text[3], 30, "hu1=%f", hu_moments.hu1);
snprintf (text[4], 30, "hu2=%f", hu_moments.hu2);
snprintf (text[5], 30, "hu3=%f", hu_moments.hu3);
snprintf (text[6], 30, "hu4=%f", hu_moments.hu4);
snprintf (text[7], 30, "hu5=%f", hu_moments.hu5);
snprintf (text[8], 30, "hu6=%f", hu_moments.hu6);
snprintf (text[9], 30, "hu7=%f", hu_moments.hu7);
cvGetTextSize (text[0], &font, &text_size, 0);
for (i = 0; i < 10; i++)
    cvPutText (src_img, text[i], cvPoint (10, (text_size.height + 3) * (i + 1)), &font,
cvScalarAll (0));

// (5)입력 이미지와 모멘트 계산 결과를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
cvShowImage ("Image", src_img);

```

```

cvWaitKey (0);

cvDestroyWindow ("Image");
cvReleaseImage (&src_img);

return 0;
}

```

// (1)이미지를 읽어들인다.3 채널 이미지의 경우는 COI 하지만 세트 되어 있지 않으면 안 된다 커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.만약, 지정된 이미지가 3 채널 이미지이면,COI 하지만 세트 되어 있지 않으면, 모멘트의 계산을 할 수 없기 때문에 체크한 후, 제일 최초의 채널을 지정해 둔다.

// (2)입력 이미지의 3 차까지의 이미지 모멘트를 계산한다  
 이미지의 3 다음까지의 공간 모멘트와 중심 모멘트를 함수 cvMoments()그리고 계산한다.  
 CvMoments 구조체는, 이하와 같이 정의되고 있어

```

typedef struct CvMoments
{
    double  m00, m10, m01, m20, m11, m02, m30, m21, m12, m03; /* spatial moments */
    double  mu20, mu11, mu02, mu30, mu21, mu12, mu03; /* central moments */
    double  inv_sqrt_m00; /* m00 != 0 ? 1/sqrt(m00) : 0 */
}
CvMoments;

```

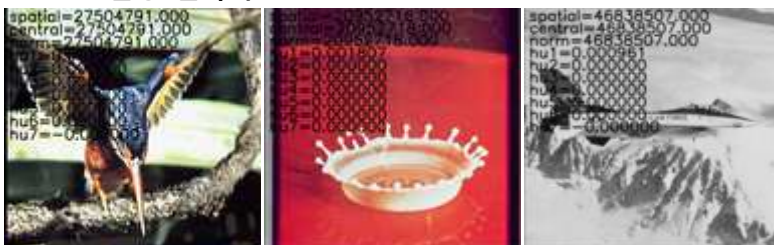
샘플 예의 경우이면, 예를 들면 직접,moments.m00 (으)로서0다음 모멘트의 계산 결과를 취득하는 일도 할 수 있다. 결과 표시 때문에,COI의 설정을 해제해 둔다.

// (3)모멘트나 Hu 모멘트 불변량을 얻을 수 있었다 CvMoments 구조체의 값을 사용해 계산한다. 위와 같게 직접 구조체의 내용에 액세스 해도 계산 결과는 취득할 수 있지만,OpenCV 에는 함수 cvGetSpatialMoment(), cvGetCentralMoment(), cvGetNormalizedCentralMoment()등에서, 필요한 공간 모멘트나 중심 모멘트, 정규화된 중심 모멘트등을 취득할 수가 있다.샘플예에서는 모든 0 차 모멘트치의 취득을 지정했다.

또 Hu 모멘트 불변량도 계산한다.7 의 Hu 모멘트 불변량의 각각의 의미는, 레퍼런스를 참조의 일.

// (4)얻을 수 있던 각종 모멘트치나 Hu 모멘트 불변량을 문자로서 이미지에 그리기  
 윈도우를 생성해, 요구한 모멘트치를 표시하기 위해서, 함수 cvPutText()(을)를 이용하고, 값을 텍스트로서 이미지에 덧쓰기해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예



## ■ 하후 변환

OpenCV그림, 표준적 하후 변환, 확률적 하후 변환, 멀티 스케일형의 고전적 하후 변환의 3 종류의 수법을 실장하고 있다.표준적 하후 변환과 고전적 하후 변환에서는, 검출된 선은 점(0, 0)(으)로부터 선까지의 거리 $\rho$ 라고 선의 법선이 x축과 이루는 모퉁이 $\theta$ 의2개의 값으로 나타내진다.한편 확률적 하후 변환에서는, 단 점을 가지는 선분으로서 선을 검출한다.그 때문에, 검출된 선분은, 시점과 종점에서 나타내진다.

또, 엔은 중심과 반경을 나타내는 3개의 파라미터로 표현할 수 있기 위해, 직선 검출로 투표에 이용하는 하후 공간을, 삼차원에 확장하는 것으로 엔을 검출하는 것이 가능하게 된다.

샘플

---

## 하후 변환에 의한 직선 검출 cvHoughLines2

표준적 하후 변환과 확률적 하후 변환을 지정해 선(선분)의 검출을 행한다.샘플 코드내의 각 파라미터치는 처리 예의 이미지에 대해서 튜닝 되고 있다.

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

int
main (int argc, char **argv)
{
    int i;
    float *line, rho, theta;
    double a, b, x0, y0;
    IplImage *src_img_std = 0, *src_img_prob = 0, *src_img_gray = 0;
    CvMemStorage *storage;
    CvSeq *lines = 0;
    CvPoint *point, pt1, pt2;

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img_gray = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE);
    if (src_img_gray == 0)
        return -1;
    src_img_std = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
    src_img_prob = cvCloneImage (src_img_std);

    // (2)하후 변환을 위한 사전 처리
    cvCanny (src_img_gray, src_img_gray, 50, 200, 3);
    storage = cvCreateMemStorage (0);
```

```

// (3)표준적 하후 변환에 의한 선의 검출과 검출한 선의 그리기
lines = cvHoughLines2 (src_img_gray, storage, CV_HOUGH_STANDARD, 1, CV_PI / 180, 50, 0, 0);
for (i = 0; i < MIN (lines->total, 100); i++) {
    line = (float *) cvGetSeqElem (lines, i);
    rho = line[0];
    theta = line[1];
    a = cos (theta);
    b = sin (theta);
    x0 = a * rho;
    y0 = b * rho;
    pt1.x = cvRound (x0 + 1000 * (-b));
    pt1.y = cvRound (y0 + 1000 * (a));
    pt2.x = cvRound (x0 - 1000 * (-b));
    pt2.y = cvRound (y0 - 1000 * (a));
    cvLine (src_img_std, pt1, pt2, CV_RGB (255, 0, 0), 3, 8, 0);
}

// (4)확률적 하후 변환에 의한 선분의 검출과 검출한 선분의 그리기
lines = 0;
lines = cvHoughLines2 (src_img_gray, storage, CV_HOUGH_PROBABILISTIC, 1, CV_PI / 180, 50, 50, 10);
for (i = 0; i < lines->total; i++) {
    point = (CvPoint *) cvGetSeqElem (lines, i);
    cvLine (src_img_prob, point[0], point[1], CV_RGB (255, 0, 0), 3, 8, 0);
}

// (5)검출 결과 표시용의 윈도우를 확보해 표시한다
cvNamedWindow ("Hough_line_standard", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("Hough_line_probalistic", CV_WINDOW_AUTOSIZE);
cvShowImage ("Hough_line_standard", src_img_std);
cvShowImage ("Hough_line_probalistic", src_img_prob);
cvWaitKey (0);

cvDestroyWindow ("Hough_line_standard");
cvDestroyWindow ("Hough_line_probalistic");
cvReleaseImage (&src_img_std);
cvReleaseImage (&src_img_prob);
cvReleaseImage (&src_img_gray);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고

읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_GRAYSCALE (을)를 지정하는 것으로, 처리용으로서 1 채널, 곳 의자 케일로 변환한다. 또, 표시용(검출한 선을 빨강으로 나타내 보인다)을 위해서, 2 번째의 인수에 CV\_LOAD\_IMAGE\_COLOR (을)를 지정하고, 칼라 이미지이라고 해도 읽어들이 든다.

// (2)하후 변환을 위한 사전 처리

cvHoughLines2()(은)는, 입력으로서 8 비트, 싱글 채널, 2 치화 이미지를 취한다. 그 때문에 사전 처리로서 cvCanny()(을)를 이용하고, 입력 처리용의 그레이 스케일 이미지를 엣지 이미지로 변환한다. cvCanny()(은)는, 샘플과 같이 인플레이스 처리가 가능하다.

또, cvHoughLines2() 그리고 사용하는 검출 결과의 선을 보존하는 메모리스트레이지를 생성해 둔다.

// (3)표준적 하후 변환에 의한 선의 검출과 검출한 선의 그리기

표준적 하후 변환 CV\_HOUGH\_STANDARD(을)를 지정하고, 엣지 이미지로부터 선을 검출한다. 검출 결과는 순서 lines 의 선두 포인터로서 돌려주어진다.

하나 하나의 선은, 순서 lines(으)로부터, cvGetSeqElem()(을)를 사용하고, 인덱스의 순서에 꺼낸다. 순서 lines 의 하나의 요소 line(은)는, 32 비트 부동 소수점형 2 채널 배열(배열 요소가 2 개) CV\_32FC2<sup>(주 1)</sup>이기 위해, line[0]에  $\rho$  하지만, line[1]에  $\theta$  하지만 들어간다.

최초로,  $\rho$ (와)과  $\theta$ (을)를 이용해 선상의 1 점( $x_0, y_0$ )(을)를,  $x_0 = \rho \cdot \cos(\theta)$ ,  $y_0 = \rho \cdot \sin(\theta)$ (으)로서 계산한다.

다음에( $x_0, y_0$ )(을)를 중심으로서 각각 1000pixel 떨어진 선상의 2 점 pt1, pt2(을)를 계산한다. 여기서  $\theta$  하지만 검출된 선의 법선 방향을 나타내고 있기 때문에, 검출된 선과 x 축이 이루는 모퉁이는  $\theta + 90[\text{deg}]$  된다.  $\sin(\theta + 90) = \cos(\theta)$ ,  $\cos(\theta + 90) = -\sin(\theta)$  이기 위해,  $pt1.x = x_0 + 1000 \cdot (-\sin(\theta))$ ,  $pt1.y = y_0 + 1000 \cdot \cos(\theta)$  그리고 얻을 수 있다.

선상의 2 점을 이용하고, 결과 표시용의 이미지 데이터 src\_img\_std 위에 적색, 선폭 3 그리고 직선을 표시한다. 이미지 영역외의 선은 클리핑 된다.

검출 직선수가 많아져 결과가 보기 나빠지므로, 표시를 100 line 까지 누르고 있다.

// (4)확률적 하후 변환에 의한 선분의 검출과 검출한 선분의 그리기

확률적 하후 변환 CV\_HOUGH\_PROBABILISTIC(을)를 지정하고, 엣지 이미지로부터 선분을 검출한다. 검출 결과는 순서 lines 의 선두 포인터로서 돌려주어진다.

하나 하나의 선분은, 순서 lines(으)로부터, cvGetSeqElem()(을)를 사용하고, 인덱스의 순서에 꺼낸다. 순서 lines 의 하나의 요소 point(은)는, 32 비트 정수형 4 채널 배열(배열 요소가 4 개) CV\_32SC4<sup>(주 1)</sup>이기 위해, point[0]에 시점 좌표가, point[1]에 종점 좌표가 들어간다.

시점, 종점을 이용하고, 결과 표시용의 이미지 데이터 src\_img\_prob 위에 적색, 선폭 3 그리고 선분을 표시한다.

// (5)검출 결과 표시용의 윈도우를 확보해 표시한다

윈도우를 생성해, 직선 검출 결과를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

(주 1) CV\_<비트수(데프스)>(S|U|F)C<채널수>



## 실행 결과예

입력 이미지와 직선 검출 결과



[좌] 입력 이미지

[중] 표준적 하후 변환에 의해, 검출된 직선(100[line]마서 표시)

[우] 확률적 하후 변환에 의해, 검출된 선분

## 하후 변환에 의한 엔검출 cvHoughCircles

그레이 스케일 이미지중의 엔을 하후 변환을 이용해 검출한다.샘플 코드내의 각 파라미터치는 처리 예의 이미지에 대해서 튜닝 되고 있다.

### 샘플 코드

표시의 변환

```
#include <stdio.h>
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i;
    float *p;
    IplImage *src_img = 0, *src_img_gray = 0;
    CvMemStorage *storage;
    CvSeq *circles = 0;

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img_gray = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE);
    if (src_img_gray == 0)
        exit (-1);
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);

    // (2)하후 변환을 위한 사전 처리(이미지의 평활화를 행하지 않으면 오류 검출이 발생하기
    // 쉽다)
    cvSmooth (src_img_gray, src_img_gray, CV_GAUSSIAN, 11, 11, 0, 0);
    storage = cvCreateMemStorage (0);
```

```

// (3)하후 변환에 의한 엔의 검출과 검출한 엔의 그리기
circles = cvHoughCircles (src_img_gray, storage, CV_HOUGH_GRADIENT,
                          1, 100, 20, 50, 10, MAX (src_img_gray->width, src_img_gray->height));
for (i = 0; i < circles->total; i++) {
    p = (float *) cvGetSeqElem (circles, i);
    cvCircle (src_img, cvPoint (cvRound (p[0]), cvRound (p[1])), 3, CV_RGB (0, 255, 0), -1, 8, 0);
    cvCircle (src_img, cvPoint (cvRound (p[0]), cvRound (p[1])), cvRound (p[2]), CV_RGB (255, 0, 0), 3, 8, 0);
}

// (4)검출 결과 표시용의 윈도우를 확보해 표시한다
cvNamedWindow ("circles", 1);
cvShowImage ("circles", src_img);
cvWaitKey (0);

cvDestroyWindow ("circles");
cvReleaseImage (&src_img);
cvReleaseImage (&src_img_gray);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 src\_img\_gray 에 읽어들인다.2 번째의 인수에 CV\_LOAD\_IMAGE\_GRAYSCALE (을)를 지정하는 것으로, 처리용으로서 1 채널, 그레이 스케일로 변환한다. 또, 표시용(검출한 선을 빨강으로 나타내 보인다)을 위해서,2 번째의 인수에 CV\_LOAD\_IMAGE\_COLOR (을)를 지정하고, 칼라 이미지 src\_img(이)라고 해도 읽어들이든다.

// (2)하후 변환을 위한 사전 처리

cvHoughCircles()(은)는, 입력으로서 8 비트, 싱글 채널, 그레이 스케일 이미지를 이용한다. 오류 검출을 줄이기 위해서, 사전 처리로서 cvSmooth()(을)를 이용한 이미지의 평활화를 베푼다. cvSmooth()(은)는, 샘플과 같이 인플레이스 처리가 가능하다. 또,cvHoughCircles()그리고 사용하는 검출 결과의 엔을 보존하는 메모리스트레이지를 생성해 둔다.

// (3)하후 변환에 의한 엔의 검출과 검출한 엔의 그리기

기본적인 2 단계 하후 변환 CV\_HOUGH\_GRADIENT(을)를 지정하고(현재는 이것 밖에 실장되어 있지 않다), 그레이 스케일 이미지로부터 엔을 검출한다. 검출 결과는 순서 circles 의 선두 포인터로서 돌려주어진다.

하나 하나의 엔은, 순서 circles(으)로부터,cvGetSeqElem()(을)를 사용하고, 인덱스의 순서에 꺼낸다. 순서 circles 의 하나의 요소 p(은)는,32 비트 부동 소수점형 3 채널 배열(배열 요소가 3 개)

CV\_32FC3<sup>(주 1)</sup>이기 위해, p[0]에 엔중심의 x 좌표치가, p[1]에 엔중심의 y 좌표가, p[2]에 엔의 반경이 각각 들어간다.

검출된 엔을 결과 표시용의 이미지 데이터 src\_img 위에 적색, 선폭 3 그리고 표시해, 엔의 중심을 초록으로 표시하는,

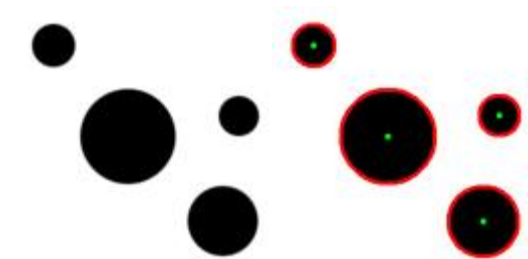
// (4)검출 결과 표시용의 윈도우를 확보해 표시한다

윈도우를 생성해, 엔검출 결과를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

(주 1) CV\_<비트수(데프스)>(S|U|F)C<채널수>

### 실행 결과예

입력 이미지와 엔검출 결과



[좌] 입력 이미지

[우] 하후 변환에 의해 검출된 엔

### ■ 거리 변환

2값이미지의 각 화소에 대해서, 거기로부터 값이0인 화소에의 최단 거리를 주는 변환을 거리 변환이라고 한다.Open CV그림, 마스크를 지정하지 않고 근사 없음의 정확한 거리계산을 행하는 일도, 처리의 고속화를 위해서, 지정한 마스크의 시프트를 이용한 근사 계산을 행하는 일도 가능하다.

샘플

### 거리 변환과 그 가시화 cvDistTransform

입력 이미지에 대해서 거리 변환을 행해, 결과를 0-255 에 정규화해 가시화한다

### 샘플 코드

표시의 변환

```

#include <stdio.h>
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *dst_img, *dst_img_norm;

    // (1)이미지를 읽어들이
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);
    if (src_img->nChannels != 1)
        exit (-1);
    if (src_img->depth != IPL_DEPTH_8U)
        exit (-1);

    // (2)처리 결과의 거리 이미지 출력용의 이미지 영역과 표시 윈도우를 확보
    dst_img = cvCreateImage (cvSize (src_img->width, src_img->height), IPL_DEPTH_32F, 1);
    dst_img_norm = cvCreateImage (cvSize (src_img->width, src_img->height), IPL_DEPTH_8U,
1);

    // (3)거리 이미지를 계산해, 표시용으로 결과를 0-255 에 정규화한다
    cvDistTransform (src_img, dst_img, CV_DIST_L2, 3, NULL, NULL);
    cvNormalize (dst_img, dst_img_norm, 0.0, 255.0, CV_MINMAX, NULL);

    // (4)거리 이미지를 표시, 키가 밀렸을 때에 종료
    cvNamedWindow ("Source", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Distance Image", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Source", src_img);
    cvShowImage ("Distance Image", dst_img_norm);
    cvWaitKey (0);

    cvDestroyWindow ("Source");
    cvDestroyWindow ("Distance Image");
    cvReleaseImage (&src_img);
    cvReleaseImage (&dst_img);
    cvReleaseImage (&dst_img_norm);

    return 0;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다.

cvDistTransform()(은)는, 입력으로서 8 비트 싱글 채널의 2 치 이미지 밖에 잡히지 않는다 의로, 여기서 입력 이미지의 채널수와 데프스의 체크를 행한다.

// (2)처리 결과의 거리 이미지 출력용의 이미지 영역과 표시 윈도우를 확보

cvDistTransform()의 출력은,32 비트 부동 소수점형 싱글 채널이기 위해, 채널, 데프스를 지정해 출력용 이미지 dst\_img(을)를 준비한다. 또, 결과를 보기 쉽게 표시하기 위한 결과 표시용의 8 비트 부호 없음 싱글 채널의 이미지 영역 dst\_img\_norm(을)를 준비해, 이 이미지를 표시하기 위한 윈도우를 준비한다.

// (3)거리 이미지를 계산해, 표시용으로 결과를 0-255 에 정규화한다

각 픽셀로부터 값 0 까지의 최근 거리 거리를 계산한다.cvDistTransform()의 제 1, 제 2 인수의 형태는 CvArr\* 이지만, 샘플로 나타내 보이도록(듯이), 직접 IpImage\* 형태의 변수를 지정할 수 있다. 또 이 샘플에서는, 고속으로 근사 거리를 요구하기 위해서 CV\_DIST\_L2(Euclid 거리), 3×3 마스크를 지정해 있다.

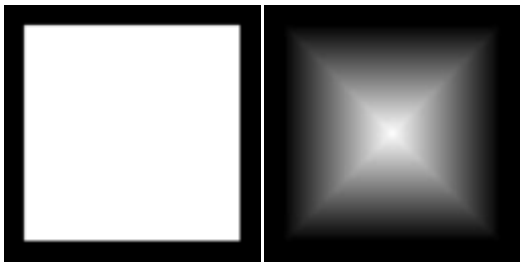
계산 후의 이미지의 화소치의 범위를,cvNormalized()(을)를 이용해 0-255 에 정규화한다 (8 비트 부호 없음 싱글 채널 이미지로 한다). 정규화의 타입으로서 CV\_MINMAX(배열의 값이 지정의 범위에 들어가도록(듯이) 스케일링과 시프트를 실시한다)(을)를 지정해, 제 3, 제 4 인수로 그 범위를 지정해 두면 간단하게 정규화를 할 수 있다.

// (4)거리 이미지를 표시, 키가 밀렸을 때에 종료

입력 이미지를 표시하기 위한 윈도우를 cvNamedWindow()그리고 확보해,cvShowImage()에 입력 이미지의 포인터를 건네주는 것으로, 실제의 표시를 행한다. 거리 이미지도 마찬가지. 정규화된 거리 이미지 dst\_img\_norm(을)를 윈도우를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다. 마지막으로, 종료할 경우에는 확보한 구조체의 메모리를 해방한다.

### 실행 결과예

입력 이미지(8 비트,1 채널,2 값이미지)과 0-255 에 정규화한 거리 이미지



[좌] 256×256 의 이미지의 주위에 값 0 의 폭 20 의 보더를 가지는 입력 이미지.

[우] 입력 이미지에 대해서 거리 변환한 결과, 이미지의 중심이 가장 거리가 멀고, 휘도가 높아지고 있다.

---

### ■ 이미지 수복

OpenCV1.0그리고 새롭게 추가된 기능으로, 지정한 영역 경계 근방의 화소를 이용해 지정한 영역내 의 화소치를 재 구성한다. PhotoShop(Ver.CS2이후)의 스포트 수복 브러쉬 기능과 같이, 이미지 노이즈를 제거하거나 이미지로부터 불필요한 오브젝트를 제거하기 위해서 이용하는 것이 가능하다.

## 불요 오브젝트의 제거 cvInpaint

이미지의 불필요한 문자열 부분에 대한 마스크 이미지를 지정해 문자열을 제거한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src_img = 0, *mask_img = 0, *dst_img;

    // (1)이미지의 읽기
    if (argc >= 2)
        src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (src_img == 0)
        exit (-1);
    if (src_img->depth != IPL_DEPTH_8U)
        exit (-1);

    // (2)마스크 이미지의 읽기
    if (argc >= 3)
        mask_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
    if (mask_img == 0)
        exit (-1);
    if (mask_img->nChannels != 1)
        exit (-1);
    if (mask_img->depth != IPL_DEPTH_8U)
        exit (-1);

    // (3)수복 이미지 출력용의 이미지 영역을 확보해, 이미지 수복을 실행
    dst_img = cvCloneImage (src_img);
    cvInpaint (src_img, mask_img, dst_img, CV_INPAINT_NS, 10);

    // (4)입력, 마스크, 수복 결과 이미지의 표시
    cvNamedWindow ("Source", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Mask", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Inpaint", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Source", src_img);
    cvShowImage ("Mask", mask_img);
```

```

cvShowImage ("Inpaint", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Source");
cvDestroyWindow ("Mask");
cvDestroyWindow ("Inpaint");
cvReleaseImage (&src_img);
cvReleaseImage (&mask_img);
cvReleaseImage (&dst_img);

return 1;
}

```

// (1)이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다.

cvInpaint()(은)는, 입력으로서 8 비트의 이미지 밖에 잡히지 않기 때문에, 여기서 입력 이미지의 데프스의 체크를 행한다.

// (2)마스크 이미지의 읽기

커멘드 인수로 지정된 파일명의 이미지(마스크 이미지)을 읽어들인다.

마스크로서는 8 비트, 싱글 채널의 이미지 밖에 잡히지 않는다 의로, 여기서 마스크 이미지의 채널수와 데프스의 체크를 행한다.

마스크 이미지의 비 0 부분이, 수복 대상 영역이 된다.

// (3)수복 이미지 출력용의 이미지 영역을 확보해, 이미지 수복을 실행

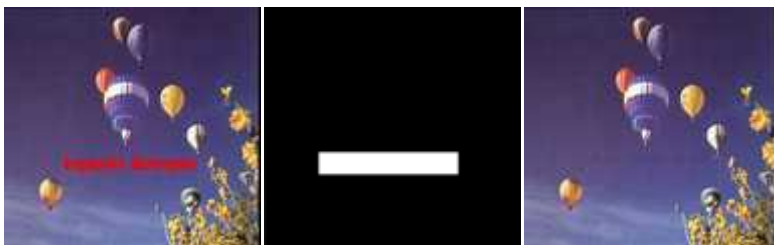
입력 이미지와 같은 사이즈, 채널수, 데프스의 출력용

이미지 dst\_img(을)를,cvCloneImage()그리고 확보해, 나비에·stokes 베이스의 수법 CV\_INPAINT\_NS (을)를 지정하고, 이미지 수복을 실행한다.

// (4)입력, 마스크, 수복 결과 이미지의 표시

윈도우를 생성해, 입력, 마스크, 결과를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예



[좌] 「Inpaint Sample」 라고 하는 불필요한 문자가 삽입된 이미지

[중] 문자 영역을 화소치 255 의 흰색 픽셀로 나타내 보인 마스크 이미지

[우] 이미지 수복 결과, 예쁘게 수복하려면 마스크 형상을 세세하게 지정하는 편이 좋다.

## 히스토그램의 그리기 cvCalcHist

입력 이미지의 히스토그램을 계산해, 결과의 그래프를 그리기 한다

### 샘플 코드

표시의 변환

Python판으로 변경

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, j, bin_w;
    int hist_size = 256;
    int sch = 0, ch_width = 260;
    float max_value = 0;
    float range_0[] = { 0, 256 };
    float *ranges[] = { range_0 };
    IplImage *src_img = 0, *dst_img[4] = { 0, 0, 0, 0 }, *hist_img;
    CvHistogram *hist;

    // (1)이미지를 읽어들인다
    if (argc < 2 ||
        //(src_img = cvLoadImage (argv[1],
        CV_LOAD_IMAGE_ANYDEPTH|CV_LOAD_IMAGE_ANYCOLOR))==0)
        (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYCOLOR)) == 0)
        return -1;

    // (2)입력 이미지의 채널 몇분의 이미지 영역을 확보
    sch = src_img->nChannels;
    for (i = 0; i < sch; i++) {
        dst_img[i] = cvCreateImage (cvSize (src_img->width, src_img->height), src_img->depth,
1);
    }

    // (3)히스토그램 구조체, 히스토그램 이미지 영역을 확보
    hist = cvCreateHist (1, &hist_size, CV_HIST_ARRAY, ranges, 1);
    hist_img = cvCreateImage (cvSize (ch_width * sch, 200), 8, 1);

    // (4)입력 이미지가 멀티 채널의 경우, 이미지를 채널마다 분할
    if (sch == 1)
        cvCopy (src_img, dst_img[0], NULL);
    else
        cvSplit (src_img, dst_img[0], dst_img[1], dst_img[2], dst_img[3]);

    // (5)히스토그램 이미지를 클리어
```



```

cvSet (hist_img, cvScalarAll (255), 0);
for (i = 0; i < sch; i++) {
    // (6)히스토그램을 계산하고, 슬캘링
    cvCalcHist (&dst_img[i], hist, 0, NULL);
    cvGetMinMaxHistValue (hist, 0, &max_value, 0, 0);
    cvScale (hist->bins, hist->bins, ((double) hist_img->height) / max_value, 0);
    // (7)히스토그램의 그리기
    bin_w = cvRound ((double) ch_width / hist_size);
    for (j = 0; j < hist_size; j++)
        cvRectangle (hist_img,
                     cvPoint (j * bin_w + (i * ch_width), hist_img->height),
                     cvPoint ((j + 1) * bin_w + (i * ch_width),
                               hist_img->height - cvRound (cvGetReal1D (hist->bins, j))),
                     cvScalarAll (0), -1, 8, 0);
}

// (8)히스토그램 이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
cvShowImage ("Image", src_img);
cvNamedWindow ("Histogram", CV_WINDOW_AUTOSIZE);
cvShowImage ("Histogram", hist_img);
cvWaitKey (0);

cvDestroyWindow ("Histogram");
cvReleaseImage (&src_img);
cvReleaseImage (&hist_img);
for (i = 0; i < sch; i++) {
    cvReleaseImage (&dst_img[i]);
}
cvReleaseHist (&hist);

return 0;
}

```

// (1)이미지를 읽어들인다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변경하지 않고 읽어들인다.

// (2)입력 이미지의 채널 몇분의 이미지 영역을 확보

입력 이미지의 채널수와 같은 수의 이미지 영역을 확보해, 그 포인터를 배열 dst\_img[]에 납입한다.또, 이 배열은 0 그리고 초기화되고 있으므로, 장 네루수이상의 인덱스를 가지는 요소의 값은 0(이)가 된다.

// (3)히스토그램 구조체, 히스토그램 이미지 영역을 확보  
 히스토그램 구조체 CvHistogram, 히스토그램 이미지를 그리기 한다 IpImage 의 영역을 확보한다. 이번 계산되는 것은, 휘도치만의 히스토그램이므로, 히스토그램 구조체 하 1 차원, 빈의 수는 256(=hist\_size), 빈의 반응을 일으키는 최소의 물리량은 0~255(=ranges)이므로, 입력치중,0~255 의 범위에 있는 값(하나의 채널에 대해 8bit)만이, 256 개의 빈에 누적되어 가게 된다.또, 마지막 인수를 1(비 0 의 값)으로 하는 것으로, 등간격인 빈을 가지는 히스토그램이 된다. 또,ranges[i][0]≤v<ranges[i][1]의 범위의 값이 입력으로서 놓치는 것에 주의한다.

// (4)입력 이미지가 멀티 채널의 경우, 이미지를 채널마다 분할  
 입력 이미지가 멀티 채널의 경우에는, 이미지를 채널마다 분할하고, 의 히스토그램을 요구한다.여기서, 분할 후의 이미지 배열 dst\_img[] (을)를 최초로 0 그리고 초기화하고 있으므로, 그대로 함수 cvSplit()에 건네주어 도 좋은 것에 주의한다.

// (5)히스토그램 이미지를 클리어  
 다음에, 우선, 히스토그램을 그리기 한다 IpImage(을)를, 휘도치 255(백색)(으)로 쿠 리어 한다.즉, 히스토그램은, 백색 배경의 이미지가 된다.

// (6)히스토그램을 계산하고, 슬캘링  
 각 채널마다 히스토그램을 계산한다.계산 후의 히스토그램의 최대치를 꺼내, 그 값을 이용해 히스토그램의 높이가 이미지에 들어가도록(듯이) 슬캘링을 실시한다. 여기에서는, 히스토그램의 피크(최대치)가, 이미지 hist\_img 의 높이로 동일하고 되도록(듯이) 조정하고 있다.

// (7)히스토그램의 그리기  
 결과적으로 얻을 수 있던 히스토그램을, 차례차례, 이미지에 그리기 한다.

// (8)히스토그램 이미지를 표시, 키가 밀렸을 때에 종료  
 전채널의 히스토그램이 그리기 된 이미지 포함한 윈도우를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

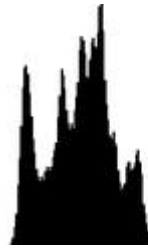
입력 이미지(24 비트,3 채널)과 각 채널에 대한 히스토그램



입력 이미지(24 비트,1 채널)과 그에 대한 히스토그램



[좌] 높은 명도를 가지도록(듯이) 변경한 이미지와 그 히스토그램.그래프가 높은 휘도치 측에 모이고 있다. [우] 낮은 명도를 가지도록(듯이) 변경한 이미지와 그 히스토그램.그래프가 낮은 휘도치 측에 모이고 있다.



[좌] 높은 콘트라스트를 가지도록(듯이) 변경한 이미지와 그 히스토그램. 최저 휘도가 0, 최고 휘도가 255 되는 넓은 분포를 가지지만, 콘트라스트를 너무 강조해서, 특정 의 범위(빈)의 정보가 빠져 있다. [우] 낮은 콘트라스트를 가지도록(듯이) 변경한 이미지와 그 히스토그램. 히스토그램이 중앙 부분에 치우쳐, 최저 휘도와 최고 명도와의 차이가 작아지고 있다.



## 히스토그램간의 거리 cvCompareHist

두 개의 입력 이미지의 히스토그램의 거리를 계산해, 그 값을 표시한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <math.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    char text[16];
    int i, hist_size = 256, sch = 0;
    float range_0[] = { 0, 256 };
    float *ranges[] = { range_0 };
    double tmp, dist = 0;
    IplImage *src_img1 = 0, *src_img2 = 0, *dst_img1[4] = { 0, 0, 0, 0 }, *dst_img2[4] = {
    0, 0, 0, 0 };
    CvHistogram *hist1, *hist2;
    CvFont font;
    CvSize text_size;

    // (1)2 매의 이미지를 읽어들인다. 채널수가 동일하지 않은 경우는, 종료
    if (argc >= 3) {
```

```

src_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
src_img2 = cvLoadImage (argv[2], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
}
if (src_img1 == 0 || src_img2 == 0)
    return -1;
if (src_img1->nChannels != src_img2->nChannels)
    return -1;

// (2)입력 이미지의 채널 몇분의 이미지 영역을 확보
sch = src_img1->nChannels;
for (i = 0; i < sch; i++) {
    dst_img1[i] = cvCreateImage (cvSize (src_img1->width, src_img1->height), src_img1-
>depth, 1);
    dst_img2[i] = cvCreateImage (cvSize (src_img2->width, src_img2->height), src_img2-
>depth, 1);
}

// (3)히스토그램 구조체를 확보
hist1 = cvCreateHist (1, &hist_size, CV_HIST_ARRAY, ranges, 1);
hist2 = cvCreateHist (1, &hist_size, CV_HIST_ARRAY, ranges, 1);

// (4)입력 이미지가 멀티 채널의 경우, 이미지를 채널마다 분할
if (sch == 1) {
    cvCopy (src_img1, dst_img1[0], NULL);
    cvCopy (src_img2, dst_img2[0], NULL);
}
else {
    cvSplit (src_img1, dst_img1[0], dst_img1[1], dst_img1[2], dst_img1[3]);
    cvSplit (src_img2, dst_img2[0], dst_img2[1], dst_img2[2], dst_img2[3]);
}

// (5)히스토그램을 계산, 정규화하고, 거리를 요구한다
for (i = 0; i < sch; i++) {
    cvCalcHist (&dst_img1[i], hist1, 0, NULL);
    cvCalcHist (&dst_img2[i], hist2, 0, NULL);
    cvNormalizeHist (hist1, 10000);
    cvNormalizeHist (hist2, 10000);
    tmp = cvCompareHist (hist1, hist2, CV_COMP_BHATTACHARYYA);
    dist += tmp * tmp;
}
dist = sqrt (dist);

// (6)요구한 거리를 문자로서 이미지에 그리기
snprintf (text, 16, "Distance=%.3f", dist);
cvInitFont (&font, CV_FONT_HERSHEY_SIMPLEX, 2.0, 2.0, 0, 4, 8);
cvPutText (src_img2, text, cvPoint (10, src_img2->height - 10), &font, cvScalarAll

```

```

(0));
cvGetTextSize ("Input1", &font, &text_size, 0);
cvPutText (src_img1, "Input1", cvPoint (10, text_size.height), &font, cvScalarAll (0));
cvPutText (src_img2, "Input2", cvPoint (10, text_size.height), &font, cvScalarAll (0));

// (7)두 개의 이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Image1", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("Image2", CV_WINDOW_AUTOSIZE);
cvShowImage ("Image1", src_img1);
cvShowImage ("Image2", src_img2);
cvWaitKey (0);

cvDestroyWindow ("Image1");
cvDestroyWindow ("Image2");
for (i = 0; i < src_img1->nChannels; i++) {
    cvReleaseImage (&dst_img1[i]);
    cvReleaseImage (&dst_img2[i]);
}
cvReleaseImage (&src_img1);
cvReleaseImage (&src_img2);
cvReleaseHist (&hist1);
cvReleaseHist (&hist2);

return 0;
}

```

// (1)2 매의 이미지를 읽어들인다

「히스토그램 이미지의 그리기」의 경우와 같게, 커멘드 인수로 지정된 이미지 (을)를, 함수 cvLoadImage()에 의해서 읽어들인다.이번은, 히스토그램간의 거리 (을)를 계산하므로, 2 매의 이미지를 읽어들인다.

// (2)입력 이미지의 채널 몇분의 이미지 영역을 확보

채널 마다의 히스토그램간 거리를 계산하기 위해서, 「히스토그램 이미지의 그리기」의 경우와 같게, 채널 몇분의 이미지 구조체를 확보한다.

// (3)히스토그램 구조체를 확보

히스토그램 구조체를 확보.

// (4)입력 이미지가 멀티 채널의 경우, 이미지를 채널마다 분할

이것도, 「히스토그램 이미지의 그리기」의 경우와 같게, 각 채널을 분할한다.

// (5)히스토그램을 계산, 정규화하고, 거리를 요구한다

함수 cvCalcHist()에 의해서, 각 이미지의 히스토그램을 계산한 후에, 거리를 비교하기 위해서 정규화를 실시한다.어느 쪽의 히스토그램도, 전빈의 값의 합계 하지만

일정(여기에서는,10000)(이)가 되도록(듯이) 조정한다. 실제로 거리를 계산하는

함수 cvCompareHist()(은)는, 이와 같이 정규화된 히스토그램에 대해서만 올바르게 실행되는

것에 주의. 이번은, 히스토그램 비교의 수법으로서 CV\_COMP\_BHATTACHARYYA(을)를 지정했다 하지만, 그 밖에도 CV\_COMP\_CORREL, CV\_COMP\_CHISQR, CV\_COMP\_INTERSECT 등이 손가락 정가능하다. 자세한 것은 레퍼런스를 참조하는 것. 또, 이 프로그램에서는, 멀티 채널 이미지의 경우의 최종적인 거리와 하고, 채널간의 거리의 자승을의 화의 평방근을 요구하도록(듯이) 하고 있다. 다만, 이것이 유일한 거리의 정의라고 하는 것으로 않는다.

// (6)요구한 거리를 문자로서 이미지에 그리기

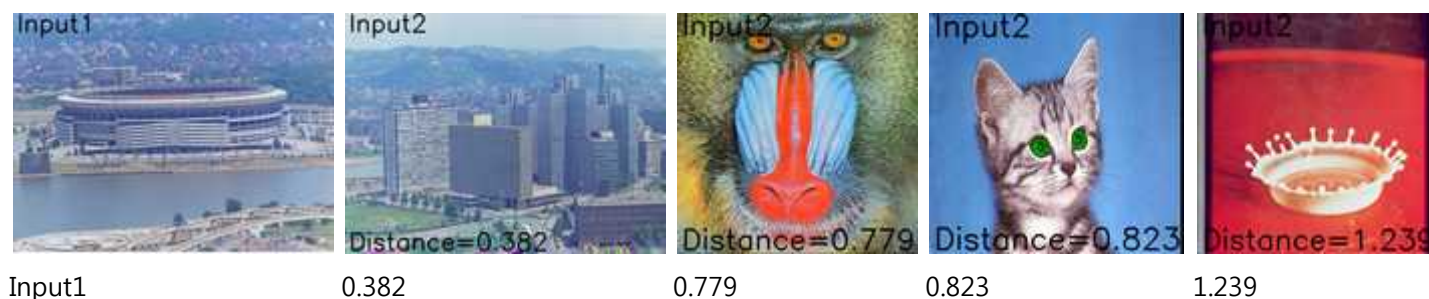
요구한 거리를 표시하기 위해서, 함수 cvPutText()(을)를 이용하고, 값을 테 키스트로서 2 매목의 이미지에 덧쓰기한다.또,1 매목,2 매목의 이미지에 ,"Input1","Input2"의 문자를 그린다.

// (7)두 개의 이미지를 표시, 키가 밀렸을 때에 종료

2 매의 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.종료시에는, 메모리를 해방한다.

### 실행 결과예

입력 이미지(Input1, Input2)의 히스토그램간의 거리



## 이차원의 히스토그램 cvCreateHist

입력 이미지의 색공간(color space)를 HSV(으)로 변환해,H-S 의 히스토그램을 그리기 한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    IplImage *src, *hsv;
    IplImage *h_plane, *s_plane, *v_plane;
    IplImage *planes[2];
    int h_bins = 30, s_bins = 32;
    int hist_size[] = { h_bins, s_bins };
    float h_ranges[] = { 0, 181 }; /* 색상은 0(0 도, 빨강)으로부터 180(360 도, 빨강)까지
변화한다 */
```

```

float s_ranges[] = { 0, 256 };          /* 채도는 0(흑-그레이-흰색)으로부터 255(순수한
스펙트럼 칼라)까지 변화한다 */
float *ranges[] = { h_ranges, s_ranges };
int scale = 10;
IplImage *hist_img = cvCreateImage (cvSize (h_bins * scale, s_bins * scale), 8, 3);
CvHistogram *hist;
float max_value = 0;
int h, s;

if (argc != 2 || (src = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
    return -1;
hsv = cvCreateImage (cvGetSize (src), 8, 3);
planes[0] = h_plane = cvCreateImage (cvGetSize (src), 8, 1);
planes[1] = s_plane = cvCreateImage (cvGetSize (src), 8, 1);
v_plane = cvCreateImage (cvGetSize (src), 8, 1);

// (1)입력 이미지의 색공간(color space)를 RGB(으)로부터 HSV(으)로 변환한다
cvCvtColor (src, hsv, CV_BGR2HSV);
cvCvtPixToPlane (hsv, h_plane, s_plane, v_plane, 0);
// (2)히스토그램을 계산
hist = cvCreateHist (2, hist_size, CV_HIST_ARRAY, ranges, 1);
cvCalcHist (planes, hist, 0, 0);
// (3)각 bin에 있어서의 값을 요구해 그것을 바탕으로 휘도치를 산출해 그리기
cvGetMinMaxHistValue (hist, 0, &max_value, 0, 0);
cvZero (hist_img);
for (h = 0; h < h_bins; h++) {
    for (s = 0; s < s_bins; s++) {
        float bin_val = cvQueryHistValue_2D (hist, h, s);
        int intensity = cvRound (bin_val * 255 / max_value);
        cvRectangle (hist_img, cvPoint (h * scale, s * scale),
                     cvPoint ((h + 1) * scale - 1, (s + 1) * scale - 1),
                     CV_RGB (intensity, intensity, intensity), CV_FILLED);
    }
}

// (4)입력 이미지와 이차원(H-S)히스토그램을 표시하고, 무엇인가 키가 밀릴 때까지
기다린다
cvNamedWindow ("Source", 1);
cvShowImage ("Source", src);
cvNamedWindow ("H-S Histogram", 1);
cvShowImage ("H-S Histogram", hist_img);
cvWaitKey (0);

cvDestroyWindow ("Source");
cvDestroyWindow ("H-S Histogram");

```



```

cvReleaseImage (&src);
cvReleaseImage (&hsv);
cvReleaseImage (&h_plane);
cvReleaseImage (&s_plane);
cvReleaseImage (&v_plane);
cvReleaseImage (&hist_img);
cvReleaseHist (&hist);

return 0;
}

```

// (1)입력 이미지의 색공간(color space)를 RGB(으)로부터 HSV(으)로 변환한다  
 함수 cvCvtColor()(을)를 이용하고, 입력된 칼라 이미지의 색공간(color space)를 RGB 공간으로부터 HSV 공간으로 변환한다. 실제로는,cvLoadImage()그리고 읽힌 이미지는,BGR 의 차례로 줄지어 있으므로, CV\_BGR2HSV(을)를 지정해 변환한다. 다음에,cvCvtPixToPlane()(을)를 이용하고,H,S,V 의 각 채널(색평면) 로 분해한다. 이,cvCvtPixToPlane()(은)는,cvcompat.h 그리고

```
#define cvCvtPixToPlane cvSplit
```

(이)라고 정의되고 있어cvSplit()(와)과 등가이다.

// (2)히스토그램을 계산

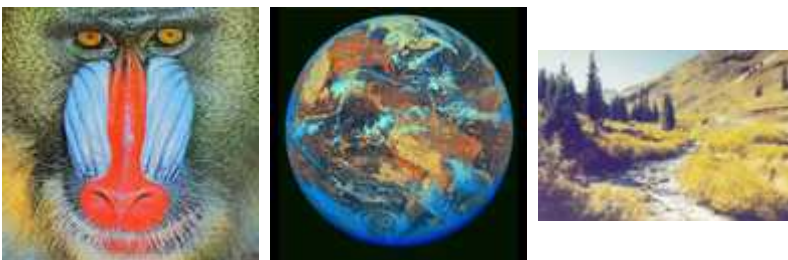
이번은,2 차원의 히스토그램 구조체를 작성해, 함수 cvCalcHist()그럼, 둘 의 색평면(H,S)에 있어서의 히스토그램을 계산한다.

// (3)각 빈에 있어서의 값을 요구해 그것을 바탕으로 휘도치를 산출해 그리기

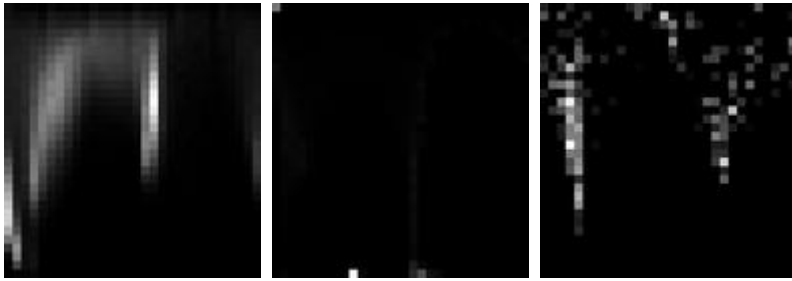
함수 cvQueryHistValue\_2D()에 의해서, 지정했다 H-S 에 있어서의 빈의 값을 요구해 그 값을 이용하고, 최대치가 255, 최소치가 0 되어에 휘도치를 산출한다. 그리고, 그 휘도치로 전부 칠해진 구형을 순서대로, 이미지 hist\_img 에 쓴다.

// (4)입력 이미지와 이차원(H-S)히스토그램을 표시하고, 무엇인가 키가 밀릴 때까지 기다린다  
 마지막으로, 입력 이미지와 그리기 되었다 2 차원(H-S) 히스토그램 이미지를 표시한다.횡축이 H(색상), 세로축이 S(채도)(을)를 나타낸다.

### 실행 결과예







## 백 프로젝션 패치 cvCalcBackProjectPatch

히스토그램간 거리에 의한, 이미지내의 템플릿 탐색

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, hist_size = 90;
    float h_ranges[] = { 0, 181 };
    float *ranges[] = { h_ranges };
    double min_val, max_val;
    CvSize dst_size;
    CvPoint min_loc, max_loc;
    IplImage *src_img, *tmp_img, *dst_img;
    IplImage *src_hsv, *tmp_hsv;
    IplImage **src_planes, *tmp_planes[3];
    CvHistogram *hist = 0;

    if (argc != 3 ||
        (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0 ||
        (tmp_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;

    src_planes = (IplImage **) cvAlloc (sizeof (IplImage *) * 3);
    for (i = 0; i < 3; i++) {
        src_planes[i] = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
        tmp_planes[i] = cvCreateImage (cvGetSize (tmp_img), IPL_DEPTH_8U, 1);
    }

    // (1)2개입력 이미지(탐색 이미지, 템플릿 이미지)의 색공간(color
    space)를,RGB(으)로부터 HSV 에 변환
```

```

src_hsv = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 3);
tmp_hsv = cvCreateImage (cvGetSize (tmp_img), IPL_DEPTH_8U, 3);
cvCvtColor (src_img, src_hsv, CV_BGR2HSV);
cvCvtColor (tmp_img, tmp_hsv, CV_BGR2HSV);
cvCvtPixToPlane (src_hsv, src_planes[0], src_planes[1], src_planes[2], 0);
cvCvtPixToPlane (tmp_hsv, tmp_planes[0], tmp_planes[1], tmp_planes[2], 0);

// (2)템플릿 이미지의 히스토그램을 계산
hist = cvCreateHist (1, &hist_size, CV_HIST_ARRAY, ranges, 1);
cvCalcHist (&tmp_planes[0], hist, 0, 0);
// (3)탐색 이미지 전체에 대해서, 템플릿의 히스토그램과의 거리(수법으로 의존)를 계산
dst_size = cvSize (src_img->width - tmp_img->width + 1, src_img->height - tmp_img-
>height + 1);
dst_img = cvCreateImage (dst_size, IPL_DEPTH_32F, 1);
cvCalcBackProjectPatch (src_planes, dst_img, cvGetSize (tmp_img), hist, CV_COMP_CORREL,
1.0);
cvMinMaxLoc (dst_img, &min_val, &max_val, &min_loc, &max_loc, NULL);
// (4)템플릿에 대응하는 위치에 구형을 그리기
cvRectangle (src_img, max_loc,
             cvPoint (max_loc.x + tmp_img->width, max_loc.y + tmp_img->height), CV_RGB
(255, 0, 0), 3);

cvNamedWindow ("Image", 1);
cvShowImage ("Image", src_img);
cvWaitKey (0);

cvDestroyWindow ("Image");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseImage (&tmp_img);
cvReleaseImage (&src_hsv);
cvReleaseImage (&tmp_hsv);
for (i = 0; i < 3; i++) {
    cvReleaseImage (&src_planes[i]);
    cvReleaseImage (&tmp_planes[i]);
}
cvFree (&src_planes);

return 0;
}

```

// (1)2 개입력 이미지(탐색 이미지, 템플릿 이미지)의 색공간(color space)를,RGB(으)로부터 HSV 에 변환  
 이번은,HSV 공간에 있어서의 H(색상)의 히스토그램을 비교하므로, 입력 이미지 의 색공간(color space)를 RGB(으)로부터 HSV(으)로 변환한다. 비교하는 히스토그램은, 하나의 채널(색평면)에 한정하지 않고와도 좋다.

// (2)템플릿 이미지의 히스토그램을 계산

최초로, 템플릿이 상의 히스토그램을 미리 계산해 둔다. 함수 cvCalcHist()에 대해서는, 전술의 샘플 코드를 참조하는 것.

// (3)탐색 이미지 전체에 대해서, 템플릿의 히스토그램과의 거리(수법으로 의존)를 계산  
함수 cvCalcBackProjectPatch()에 의해서 출력되는 이미지는,32bit 부동 소수점, 1 채널의 이미지며, 그 사이즈는,

탐색 이미지(의 폭, 높이)- 템플릿 이미지(의 폭높이)+ 1

된다.이 이미지 영역 dst\_img(을)를 확보하고, 함수 cvCalcBackProjectPatch()(을)를 실행한다. 이번은, 히스토그램 비교의 수법으로서 CV\_COMP\_CORREL(상관)(을)를 지정해 있다. 결과 이미지중에 있고 최대치를 가지는 위치, 즉 템플릿의 히스토그램과 가장 가까운 히스토그램을 가지는 위치, 를 템플릿에 대응하는 위치로 한다.

이미지중의 값의 최대, 최소치, 및 그 위치는, 함수 cvMinMaxLoc()에 의해서 꺼낸다.

// (4)템플릿에 대응하는 위치에 구형을 그리기

전술의 처리에 의해서 꺼내진 위치(좌표)는, 템플릿의 좌상의 좌표 (을)를 나타내므로, 거기에 템프레이트사이즈의 구형을 그리기 한다. 마지막으로, 그 이미지를 표시하고, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과에

이하의 예에서는, 탐색 이미지의 일부를 자른 것을 템플릿 이미지로서 급 (이)라고 있다(탐색 이미지에 직접 ROI(을)를 설정해도 같은 일이 가능하다). 따라서, 매칭 결과의 피크치는 1.0(이)가 되지만, 실제로는 이러한 완전하게 템플릿과 영역이 입력 이미지내에 존재하는 것은 드물다. 매칭 위치의 피크는 1.0(을)를 밀돌아, 또, 피크 자체가 복수 존재하는 일도 생각할 수 있다.



---

## 히스토그램의 균일화 cvEqualizeHist

그레이 스케일 이미지의 히스토그램을 균일화한다

### 샘플 코드

표시의 변환

```

#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, j, bin_w, ch_width = 260;
    int hist_size = 256;
    float max_value = 0;
    float range_0[] = { 0, 256 };
    float *ranges[] = { range_0 };
    IplImage *src_img[2], *hist_img[2];
    CvHistogram *hist;

    if (argc != 2 || (src_img[0] = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
        return -1;

    hist = cvCreateHist (1, &hist_size, CV_HIST_ARRAY, ranges, 1);
    hist_img[0] = cvCreateImage (cvSize (ch_width, 200), 8, 1);
    hist_img[1] = cvCreateImage (cvSize (ch_width, 200), 8, 1);
    src_img[1] = cvCreateImage (cvGetSize (src_img[0]), 8, 1);

    // (1)히스토그램을 균일화한다
    cvEqualizeHist (src_img[0], src_img[1]);

    for (i = 0; i < 2; i++) {
        cvSet (hist_img[i], cvScalarAll (255), 0);
        // (2)히스토그램을 계산해, 스케링
        cvCalcHist (&src_img[i], hist, 0, NULL);
        cvGetMinMaxHistValue (hist, 0, &max_value, 0, 0);
        cvScale (hist->bins, hist->bins, ((double) hist_img[i]->height) / max_value, 0);
        bin_w = cvRound ((double) ch_width / hist_size);
        // (3)히스토그램을 그리기
        for (j = 0; j < hist_size; j++)
            cvRectangle (hist_img[i],
                        cvPoint (j * bin_w, hist_img[i]->height),
                        cvPoint ((j + 1) * bin_w,
                                hist_img[i]->height - cvRound (cvGetReal1D (hist->bins, j))),
                        cvScalarAll (0), -1, 8, 0);
    }

    // (4)균일화 전후의 이미지와 그에 대한 히스토그램을 표시
    cvNamedWindow ("Image1", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Image1", src_img[0]);
    cvNamedWindow ("Histogram1", CV_WINDOW_AUTOSIZE);

```

```

cvShowImage ("Histogram1", hist_img[0]);
cvNamedWindow ("Image2", CV_WINDOW_AUTOSIZE);
cvShowImage ("Image2", src_img[1]);
cvNamedWindow ("Histogram2", CV_WINDOW_AUTOSIZE);
cvShowImage ("Histogram2", hist_img[1]);
cvWaitKey (0);

cvDestroyWindow ("Image1");
cvDestroyWindow ("Image2");
cvDestroyWindow ("Histogram1");
cvDestroyWindow ("Histogram2");
cvReleaseImage (&src_img[0]);
cvReleaseImage (&src_img[1]);
cvReleaseImage (&hist_img[0]);
cvReleaseImage (&hist_img[1]);
cvReleaseHist (&hist);

return 0;
}

```

// (1)히스토그램을 균일화한다

함수 cvEqualizeHist()에 의해, 입력 이미지를, 그 히스토그램이 균일하게 분포하는 이미지로 변환한다. 이 변환에 의해, 이미지의 콘트라스트가 오른다.

그러나, 변환 후의 이미지의 휘도치는, 변환전의 이미지의 히스토그램을 정규화한 값의 부분적인 총화이므로, 변환 후는 히스토그램에 누락이 생긴다.

// (2)히스토그램을 계산해, 슬캘링

재차, 히스토그램 균일화 전후의 이미지에 대해 히스토그램을 계산해, 스 케이링을 베풀다.

// (3)히스토그램을 그리기

계산한 히스토그램을 그리기 한다.

// (4)균일화 전후의 이미지와 그에 대한 히스토그램을 표시

실제로, 균일화 전후의 이미지와 그에 대한 히스토그램을 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예



## 템플릿 매칭 cvMatchTemplate

탐색 이미지로부터 템플릿의 위치를 찾는다

```

#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    double min_val, max_val;
    CvPoint min_loc, max_loc;
    CvSize dst_size;
    IplImage *src_img, *tmp_img, *dst_img;

    if (argc != 3 ||
        (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0 ||
        (tmp_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;

    // (1)탐색 이미지 전체에 대해서, 템플릿의 매칭치(지정한 수법으로 의존)를 계산
    dst_size = cvSize (src_img->width - tmp_img->width + 1, src_img->height - tmp_img->
>height + 1);
    dst_img = cvCreateImage (dst_size, IPL_DEPTH_32F, 1);
    cvMatchTemplate (src_img, tmp_img, dst_img, CV_TM_CCOEFF_NORMED);
    cvMinMaxLoc (dst_img, &min_val, &max_val, &min_loc, &max_loc, NULL);

    // (2)템플릿에 대응하는 위치에 구형을 그리기
    cvRectangle (src_img, max_loc,
                 cvPoint (max_loc.x + tmp_img->width, max_loc.y + tmp_img->height), CV_RGB
(255, 0, 0), 3);
    cvNamedWindow ("Image", 1);
    cvShowImage ("Image", src_img);
    cvWaitKey (0);

    cvDestroyWindow ("Image");
    cvReleaseImage (&src_img);
    cvReleaseImage (&tmp_img);
    cvReleaseImage (&dst_img);

    return 0;
}

```

// (1)탐색 이미지 전체에 대해서, 템플릿의 매칭치(지정한 수법으로 의존)를 계산  
 읽힌 탐색 이미지와 템플릿 이미지를 이용하고, 템플릿 매칭을 실시한다. 매칭에 이용하는 상관  
 함수에는, CV\_TM\_SQDIFF, CV\_TM\_SQDIFF\_NORMED,  
 CV\_TM\_CCORR, CV\_TM\_CCORR\_NORMED, CV\_TM\_CCOEFF, CV\_TM\_CCOEFF\_NORMED, 하지만

지정 가능하다.이번은,CV\_TM\_CCOEFF\_NORMED(을)를 지정해 있다.이것은, 각 비교 영역의 휘도치의 평균이 동일하면 가정했을 경우의 상관 연산식의 근사이다. 자세한 것은, 레퍼런스를 참조하는 것. 매칭은, 함수 cvCalcBackProjectionPatch()(와)과 닮아 있지만, 미리 히스토그램을 계산하는 사전 처리는 필요하지 않다.

이미지중의 값의 최대, 최소치, 및 그 위치는, 함수 cvMinMaxLoc()에 의해서 꺼낸다.

CV\_TM\_CCOEFF\_NORMED(을)를 지정했을 경우에는, 최대치의 좌표가 매칭 위치가 된다.

또,CV\_TM\_SQDIFF(SSD(이)라고도 불린다),CV\_TM\_SQDIFF\_NORMED(을)를 지정한 장소 합에는, 최소치의 좌표가 매칭 위치가 된다.

// (2)템플릿에 대응하는 위치에 구형을 그리기

탐색 이미지중이 구할 수 있던 좌표 위치에 구형을 그리기 한다.그 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

이하의 예에서는, 탐색 이미지의 일부를 자른 것을 템플릿 이미지로서 급 (이)라고 있다(탐색 이미지에 직접 ROI(을)를 설정해도 같은 일이 가능하다)의로, 완전하게 올바른 결과가 얻어지고 있다.

따라서, 탐색 영역과 템플릿 이미지로 이미지가 변화하는 경우는, 항상 올바른 위치에 매칭한다고는 할 수 없다.



### ■ 형상 매칭

두 개의 형상(윤곽 데이터, 혹은 그레이 스케일 이미지)을 입력으로 한다. OpenCV의 함수cvMatchShapes()그럼, 형상의 비교에Hu모멘트를 이용하기 위해, 회전, 슬캘링, 반전에 대해서 불변이다.

샘플

## 형상의 매칭 cvMatchShapes

두 개의 입력 이미지에 대해서,3 종류의 수법으로 형상 비교를 실시한다

### 샘플 코드

표시의 변환

```

#include <cv.h>
#include <highgui.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    char text[16];
    int i;
    double result[3];
    CvFont font;
    IplImage *src_img1, *src_img2;
    IplImage *dst_img[3];

    if (argc != 3 ||
        (src_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ||
        (src_img2 = cvLoadImage (argv[2], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
        return -1;

    for (i = 0; i < 3; i++) {
        dst_img[i] = (IplImage *) cvClone (src_img2);
    }

    // (1)3 종류의 수법으로 형상을 비교
    result[0] = cvMatchShapes (src_img1, src_img2, CV_CONTOURS_MATCH_I1, 0);
    result[1] = cvMatchShapes (src_img1, src_img2, CV_CONTOURS_MATCH_I2, 0);
    result[2] = cvMatchShapes (src_img1, src_img2, CV_CONTOURS_MATCH_I3, 0);

    // (2)형상 매칭의 결과를 이미지에 그리기
    for (i = 0; i < 3; i++) {
        snprintf (text, 16, "%.5f", result[i]);
        cvInitFont (&font, CV_FONT_HERSHEY_SIMPLEX, 1.0, 1.0, 0, 4, 8);
        cvPutText (dst_img[i], text, cvPoint (10, dst_img[i]->height - 10), &font,
cvScalarAll (0));
    }

    // (3)입력 이미지 1(와)과 3 종류의 값이 써진 입력 이미지 2(을)를 표시해, 무엇인가 키가
    밀릴 때까지 기다린다
    cvNamedWindow ("Image1", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Image2a", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Image2b", CV_WINDOW_AUTOSIZE);
    cvNamedWindow ("Image2c", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Image1", src_img1);
    cvShowImage ("Image2a", dst_img[0]);
    cvShowImage ("Image2b", dst_img[1]);

```



```

cvShowImage ("Image2c", dst_img[2]);
cvWaitKey (0);

cvDestroyWindow ("Image1");
cvDestroyWindow ("Image2a");
cvDestroyWindow ("Image2b");
cvDestroyWindow ("Image2c");
cvReleaseImage (&src_img1);
cvReleaseImage (&src_img2);
cvReleaseImage (&dst_img[0]);
cvReleaseImage (&dst_img[1]);
cvReleaseImage (&dst_img[2]);

return 0;
}

```

// (1)3 종류의 수법으로 형상을 비교

함수 cvMatchShapes()에 의해서, 두 개의 입력 이미지에 대해서 형상 매칭을 실시한다.여기서,3 번째의 인수가 비교 수법을 나타낸다. 또, 4 번째의 인수는, 현재로서는 이용되어 있지 않다.

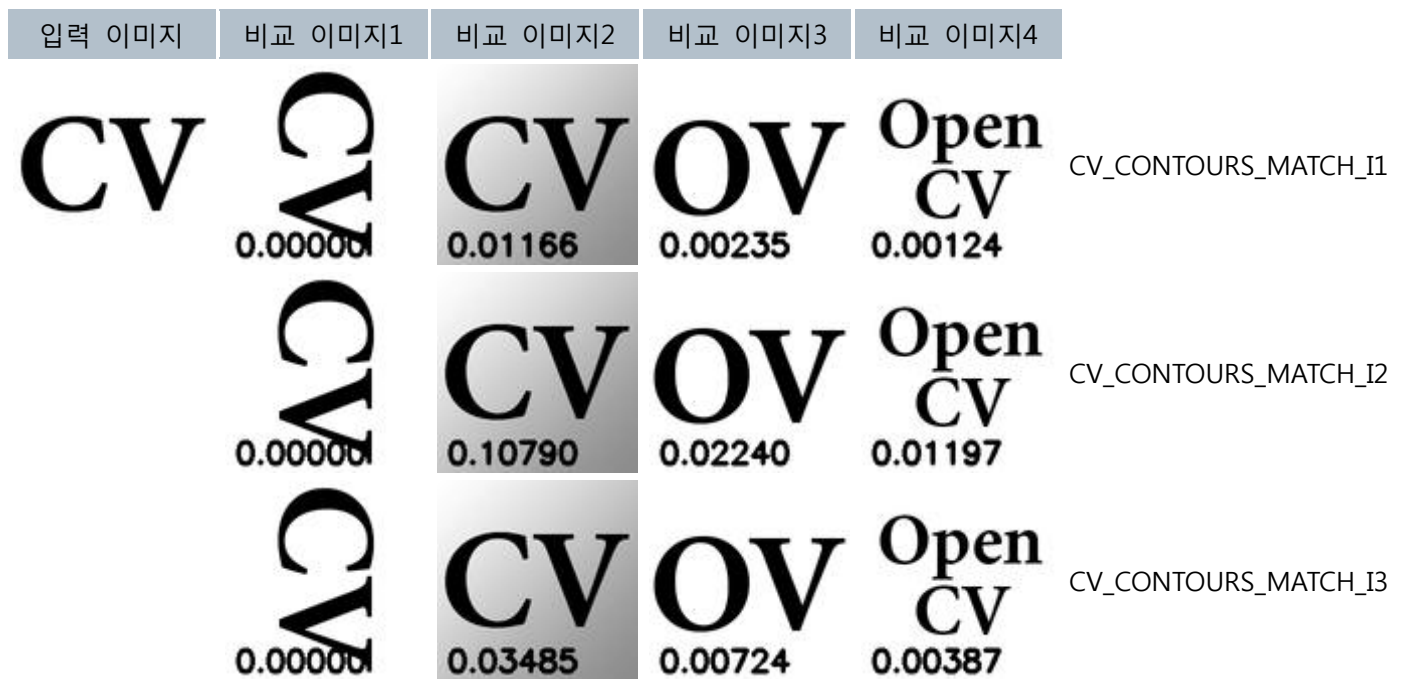
// (2)형상 매칭의 결과를 이미지에 그리기

비교 결과를, 이미지에 문자열로서 그리기 한다.작은 값인 만큼 두 개의 이미지 의 형상이 가깝다(0 의 경우는 같은 형상)이라고 하는 것이 된다.

// (3)입력 이미지 1(와)과 3 종류의 값이 써진 입력 이미지 2(을)를 표시  
결과의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예

입력 이미지	비교 이미지1	비교 이미지2	비교 이미지3	비교 이미지4	
					CV_CONTOURS_MATCH_I1
					CV_CONTOURS_MATCH_I2
					CV_CONTOURS_MATCH_I3



점렬을 포함 하는 구형 cvBoundingRect

점렬을 포함 하는 구형을 요구한다

#### 샘플 코드

표시의 변환

Python판으로 변경

```
#include <cv.h>
#include <highgui.h>
#include <time.h>

int
main (int argc, char **argv)
{
    int i;
    IplImage *img = 0;
    CvMemStorage *storage = cvCreateMemStorage (0);
    CvSeq *points;
    CvRNG rng = cvRNG (time (NULL));
    CvPoint pt;
    CvRect rect;

    // (1)이미지를 확보해 초기화한다
    img = cvCreateImage (cvSize (640, 480), IPL_DEPTH_8U, 3);
    cvZero (img);

    // (2)점렬을 생성한다
    points = cvCreateSeq (CV_SEQ_ELTYPE_POINT, sizeof (CvSeq), sizeof (CvPoint), storage);
    for (i = 0; i < 20; i++) {
```

```

pt.x = cvRandInt (&rng) % (img->width / 2) + img->width / 4;
pt.y = cvRandInt (&rng) % (img->height / 2) + img->height / 4;
cvSeqPush (points, &pt);
cvCircle (img, pt, 3, CV_RGB (0, 255, 0), CV_FILLED);
}

// (3)점렬을 포함 하는 구형을 요구해 그리기 한다
rect = cvBoundingRect (points, 0);
cvRectangle (img, cvPoint (rect.x, rect.y),
             cvPoint (rect.x + rect.width, rect.y + rect.height), CV_RGB (255, 0, 0),
2);

// (4)이미지의 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("BoundingRect", CV_WINDOW_AUTOSIZE);
cvShowImage ("BoundingRect", img);
cvWaitKey (0);

cvDestroyWindow ("BoundingRect");
cvReleaseImage (&img);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지를 확보해 초기화한다

이 샘플에서는, 외부 이미지를 읽어들이지 않고, 프로그램 내부에서 이미지를 준비한다. 또, 작성된 칼라 이미지를, 흑(0)(으)로 전부 칠해 초기화한다.

// (2)점렬을 생성한다

함수 cvRandInt()(을)를 이용하고, 이미지내가 있는 범위에 들어가는 랜덤인 점렬을 생성한다. 또, 개개의 점을 중심으로 하는 작은 엔을 이미지에 그리기 해, 점의 위치를 나타낸다. 이 샘플에서는, 랜덤인 점렬을 이용하고 있지만, 물론 사전에 검출했다 윤곽(를 구성하는 점렬)에 대해서도, 처리를 실시할 수 있다.

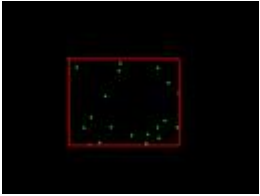
// (3)점렬을 포함 하는 구형을 요구해 그리기 한다

함수 cvBoundingRect()(을)를 이용하고, 점렬을 포함 하는 최소의 구형(다만 기울지 않았다)을 요구한다. 함수 cvMinAreaRect2()(와)과의 차이는, 이 기울기를 고려하는지 아닌지이다. cvBoundingRect()하 cvRect(을)를 돌려주지만, cvMinAreaRect2()하 cvBox2D(을)를 돌려준다. 구할 수 있었다 cvRect(을)를 이미지에 그리기 한다.

// (4)이미지의 표시, 키가 밀렸을 때에 종료

결과의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

## 실행 결과예



## 윤곽 영역의 면적과 윤곽의 길이 cvContourArea, cvArcLength

윤곽에 의해서 단락지어진 영역의 면적과 윤곽의 길이를 요구한다

### 샘플 코드

표시의 변환

Python판으로 변경

```
#include <cv.h>
#include <highgui.h>
#include <time.h>
#include <math.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    int i, size = 500;
    char text[2][64];
    double scale, area, length;
    IplImage *img = 0;
    CvMemStorage *storage = cvCreateMemStorage (0);
    CvSeq *points;
    CvRNG rng = cvRNG (time (NULL));
    CvPoint pt0, pt1;
    CvRect rect;
    CvFont font;

    // (1)이미지를 확보해 초기화한다
    img = cvCreateImage (cvSize (size, size), IPL_DEPTH_8U, 3);
    cvZero (img);

    // (2)점렬을 생성한다
    points = cvCreateSeq (CV_SEQ_POLYLINE, sizeof (CvSeq), sizeof (CvPoint), storage);
    scale = cvRandReal (&rng) + 0.5;
    pt0.x = int (cos (0) * size / 4 * scale + size / 2);
    pt0.y = int (sin (0) * size / 4 * scale + size / 2);
    cvCircle (img, pt0, 2, CV_RGB (0, 255, 0));
    cvSeqPush (points, &pt0);
```

```

for (i = 1; i < 20; i++) {
    scale = cvRandReal (&rng) + 0.5;
    pt1.x = int (cos (i * 2 * CV_PI / 20) * size / 4 * scale + size / 2);
    pt1.y = int (sin (i * 2 * CV_PI / 20) * size / 4 * scale + size / 2);
    cvLine (img, pt0, pt1, CV_RGB (0, 255, 0), 2);
    pt0.x = pt1.x;
    pt0.y = pt1.y;
    cvCircle (img, pt0, 3, CV_RGB (0, 255, 0), CV_FILLED);
    cvSeqPush (points, &pt0);
}
cvLine (img, pt0, *CV_GET_SEQ_ELEM (CvPoint, points, 0), CV_RGB (0, 255, 0), 2);

// (3)포함 구형, 면적, 길이를 요구한다
rect = cvBoundingRect (points, 0);
area = cvContourArea (points);
length = cvArcLength (points, CV_WHOLE_SEQ, 1);

// (4)결과를 이미지에 쓴다
cvRectangle (img, cvPoint (rect.x, rect.y),
             cvPoint (rect.x + rect.width, rect.y + rect.height), CV_RGB (255, 0, 0),
2);
snprintf (text[0], 64, "Area:   wrect=%d, contour=%d", rect.width * rect.height, int
(area));
snprintf (text[1], 64, "Length: rect=%d, contour=%d", 2 * (rect.width + rect.height),
int (length));
cvInitFont (&font, CV_FONT_HERSHEY_SIMPLEX, 0.7, 0.7, 0, 1, CV_AA);
cvPutText (img, text[0], cvPoint (10, img->height - 30), &font, cvScalarAll (255));
cvPutText (img, text[1], cvPoint (10, img->height - 10), &font, cvScalarAll (255));

// (5)이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("BoundingRect", CV_WINDOW_AUTOSIZE);
cvShowImage ("BoundingRect", img);
cvWaitKey (0);

cvDestroyWindow ("BoundingRect");
cvReleaseImage (&img);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지를 확보해 초기화한다

이 샘플에서는, 외부 이미지를 읽어들이지 않고, 프로그램 내부에서 이미지를 준비한다. 또, 작성된 칼라 이미지를, 흑(0)(으)로 전부 칠해 초기화한다.

// (2)점렬을 생성한다

함수 cvRandInt()(을)를 이용하고, 이미지내가 있는 범위에 들어가는 랜덤인 점렬을 생성한다. 다만, 면적을 요구할 때에 알기 쉽게 자기 교차하지 않게 한다. 또, 개개의 점을 중심으로 하는 작은 원을 이미지에 그리기 해, 점의 위치를 나타낸다. 게다가 달은 영역이 되듯이 더욱 점렬끼리를 묶는 선분을 그리기 한다. 이 샘플에서는, 랜덤인 점렬을 이용하고 있지만, 물론 사전에 검출했다 윤곽(를 구성하는 점렬) 등에 대해서도, 처리를 실시할 수 있다.

// (3)포함 구형, 면적, 길이를 요구한다

전술의 샘플과 같게, 함수 cvBoundingRect()(을)를 이용해 점렬을 포함 하는 구형 영역을 요구한다. 게다가 함수 cvContourArea()에 의해서 점렬이 구성하는 윤곽의 면적을 요구해 함수 cvArcLength()에는 그 길이(주위장)를 요구한다. cvContourArea()의 2 번째의 인수는, 윤곽의 어느 부분의 면적을 계산하는지를 나타내고 있어 이 인수가 지정되지 않는 경우에는 디폴트 인수이다 CV\_WHOLE\_SEQ 하지만 이용된다. 이 경우는, 윤곽이 나타내는 영역 모든 면적이 요구된다. cvArcLength()의 2 번째의 인수의 의미도 마찬가지여,3 번째의 인수에 0 보다 큰 값을 지정하는 것으로, 윤곽을 폐곡선으로서 취급한다. 그 외의 값에 대해서는, 레퍼런스를 참조하는 것.

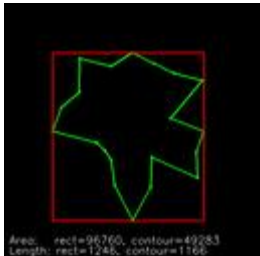
// (4)결과를 이미지에 쓴다

포함 구형, 포함 구형의 면적과 길이, 윤곽의 면적과 길이, 를 이미지에 쓴다.

// (5)이미지를 표시, 키가 밀렸을 때에 종료

결과의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



#### 배경과 주목 물체의 분리

OpenCV에는, 배경 차분을 계산할 때에 편리한 배경 통계량의 누적에 관한 함수가 실장되고 있다.

이미지중으로부터, 변화가 없는 배경 영역과 그 이외의 영역을 분리하는 것은, 컴퓨터 비전 시스템에 대해 많이 사용되는 기술이며, 여러가지 수법이 제안되고 있다. OpenCV냄새나도,cvaux그리고2종류의 동적 배경 차분 수법이 실장되고 있다. 여기에서는, 참고 문헌<sup>[1]</sup>그리고 이용되고 있는 배경 이미지의 시간적인 변화를 고려한 동적 배경 갱신에 의한 로바스트인 주목 물체의 검출 수법을,OpenCV의 함수를 이용해 실장했다.

이 수법에서는, 배경 영역 화소의 휘도 (을)를 이하와 같이 모델화하고 있다.

$$\sin$$

(은)는 휘도치의 시간 평균, (은)는 휘도의 진폭, (은)는 휘도의 주파수, (은)는 시간,

(은)는 계수, (은)는 카메라에만 의존하는 노이즈의 최대치를 각각 나타낸다.

이 모델에 대하고, 하지만,

의 경우에, 그 화

소는 배경 영역에 존재하는 화소이라고 판단한다.

배경으로 판정된 영역에서는 휘도 평균치 (와)과 진폭 (을)를 이하의 식을 이용해 갱신한다.

여기서, (은)는 갱신 속도 파라미터이다.

한편, 물체 영역으로 판정된 영역에서는, 휘도 평균치는 원래의 값을 보관 유지해, 진폭 만을 이하의 식을 이용해 갱신한다.

여기서, (은)는 물체 영역 갱신 속도 파라미터이다.

[1] 모리타 신지, 산택 일성, 테라사와 마사히코, 요코야 고평 화: "전방위 이미지 센서를 이용한 네트워크 대응형 원격 감시 시스템", 전자 정보 통신 학회 논문집지(D-II), Vol. J88-D-II, No. 5, pp. 864-875, (2005.5).

샘플

---

## 동적 배경 갱신에 의한 물체 검출 cvAcc, cvRunningAvg

배경 이미지의 시간적인 명도 변화를 고려한 물체 검출.프로그램 개시보다, 윈도우가 표시될 때까지 는 배경 통계량을 초기화하고 있기 때문에 카메라를 작동시키지 않게 주의한다.샘플내의 초기화 프레임수나 갱신 파라미터는 테스트로 사용 카메라에 관해서 튜닝 되고 있다.또,OpenCV 함수의 사용예제시를 우선했기 때문에, 처리 효율은 약간 낮다.

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ctype.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
```

```

int i, c, counter;
int INIT_TIME = 100;
int w = 0, h = 0;
double B_PARAM = 1.0 / 50.0;
double T_PARAM = 1.0 / 200.0;
double Zeta = 10.0;
CvCapture *capture = 0;
IplImage *frame = 0;
IplImage *av_img, *sgm_img;
IplImage *lower_img, *upper_img, *tmp_img;
IplImage *dst_img, *msk_img;

CvFont font;
char str[64];

// (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다
if (argc == 1 || (argc == 2 && strlen (argv[1]) == 1 && isdigit (argv[1][0])))
    capture = cvCreateCameraCapture (argc == 2 ? argv[1][0] - '0' : 0);

// (2)1 프레임 캡처 해, 캡처사이즈를 취득한다.
frame = cvQueryFrame (capture);
w = frame->width;
h = frame->height;

// (3)작업용의 영역을 생성한다
av_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_32F, 3);
sgm_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_32F, 3);
tmp_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_32F, 3);
lower_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_32F, 3);
upper_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_32F, 3);

dst_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_8U, 3);
msk_img = cvCreateImage (cvSize (w, h), IPL_DEPTH_8U, 1);

// (4)배경의 휘도 평균의 초기치를 계산한다
printf ("Background statistics initialization start\n");

cvSetZero (av_img);
for (i = 0; i < INIT_TIME; i++) {
    frame = cvQueryFrame (capture);
    cvAcc (frame, av_img);
}
cvConvertScale (av_img, av_img, 1.0 / INIT_TIME);

// (5)배경의 휘도 진폭의 초기치를 계산한다
cvSetZero (sgm_img);

```



```

for (i = 0; i < INIT_TIME; i++) {
    frame = cvQueryFrame (capture);
    cvConvert (frame, tmp_img);
    cvSub (tmp_img, av_img, tmp_img);
    cvPow (tmp_img, tmp_img, 2.0);
    cvConvertScale (tmp_img, tmp_img, 2.0);
    cvPow (tmp_img, tmp_img, 0.5);
    cvAcc (tmp_img, sgm_img);
}
cvConvertScale (sgm_img, sgm_img, 1.0 / INIT_TIME);

printf ("Background statistics initialization finish\n");

// (6)표시용 윈도우를 생성한다
cvInitFont (&font, CV_FONT_HERSHEY_COMPLEX, 0.7, 0.7);
cvNamedWindow ("Input", CV_WINDOW_AUTOSIZE);
cvNamedWindow ("Subtraction", CV_WINDOW_AUTOSIZE);

// (7)취득 이미지로부터 배경을 분리하는 루프
counter = 0;
while (1) {
    frame = cvQueryFrame (capture);
    cvConvert (frame, tmp_img);

    // (8)배경이 될 수 있는 화소의 휘도치의 범위를 체크한다
    cvSub (av_img, sgm_img, lower_img);
    cvSubS (lower_img, cvScalarAll (Zeta), lower_img);
    cvAdd (av_img, sgm_img, upper_img);
    cvAddS (upper_img, cvScalarAll (Zeta), upper_img);
    cvInRange (tmp_img, lower_img, upper_img, msk_img);

    // (9)휘도 진폭을 재계산한다
    cvSub (tmp_img, av_img, tmp_img);
    cvPow (tmp_img, tmp_img, 2.0);
    cvConvertScale (tmp_img, tmp_img, 2.0);
    cvPow (tmp_img, tmp_img, 0.5);

    // (10)배경이라고 판단된 영역의 배경의 휘도 평균과 휘도 진폭을 갱신한다
    cvRunningAvg (frame, av_img, B_PARAM, msk_img);
    cvRunningAvg (tmp_img, sgm_img, B_PARAM, msk_img);

    // (11)물체 영역이라고 판단된 영역에서는 휘도 진폭만을(배경 영역보다 늦은 속도로)
    갱신한다
    cvNot (msk_img, msk_img);
    cvRunningAvg (tmp_img, sgm_img, T_PARAM, msk_img);
}

```

```

// (12)물체 영역만을 출력 이미지에 카피한다(배경 영역은 흑)
cvSetZero (dst_img);
cvCopy (frame, dst_img, msk_img);

// (13)처리 결과를 표시한다
snprintf (str, 64, "%03d[frame]", counter);
cvPutText (dst_img, str, cvPoint (10, 20), &font, CV_RGB (0, 255, 100));
cvShowImage ("Input", frame);
cvShowImage ("Subtraction", dst_img);

counter++;

c = cvWaitKey (10);
if (c == 'Wx1b')
    break;
}

cvDestroyWindow ("Input");
cvDestroyWindow ("Subtraction");
cvReleaseImage (&frame);
cvReleaseImage (&dst_img);
cvReleaseImage (&av_img);
cvReleaseImage (&sgm_img);
cvReleaseImage (&lower_img);
cvReleaseImage (&upper_img);
cvReleaseImage (&tmp_img);
cvReleaseImage (&msk_img);

return 0;
}

```

// (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다  
함수 cvCreateCameraCapture()(을)를 이용하고, 커멘드 인수로서 주어진 번호의 카메라에 대한  
캡처 구조체 CvCapture(을)를 작성한다. 인수가 지정되지 않는 경우는, 디폴트의  
값이다"0"하지만 이용된다.

// (2)1 프레임 캡처 해, 캐프체사이즈를 취득한다.  
함수 cvQueryFrame() (을)를 호출해, 실제로 1 프레임 캡처를 실시해, 취득한 이미지로부터,  
이미지의 폭 w(와)과 높이 h(을)를 취득한다.

// (3) 작업용의 영역을 생성한다  
휘도 평균이나 휘도 진폭 계산을 위한 작업 영역을 확보한다.누적 계산이나 누승 계산을 행하기  
위해, 데프스는 32 비트를 지정해 둔다.마스크 이미지는 8 비트, 싱글 채널을 지정한다.

// (4) 배경의 휘도 평균의 초기치를 계산한다  
배경의 모델화를 위해, 지정된 프레임수(INIT\_TIME)사이의 각 화소치의 평균을 요구한다.  
cvAcc()함수를 이용하고, 이미지의 누적치를 계산한다.

// (5) 배경의 휘도 진폭의 초기치를 계산한다

(4)그리고 계산된 배경 이미지의 평균치를 이용하고, 휘도 진폭의 초기치를 계산한다. 휘도 진폭 하 (으)로서 계산한다. 휘도 진폭에 대해서도 지정된 프레임수(INIT\_TIME)사이의 평균을 요구한다.

// (6)표시용 윈도우를 생성한다

입력 이미지, 출력 이미지를 표시하는 윈도우를 생성한다.

// (7)취득 이미지로부터 배경을 분리하는 루프

배경 이미지 모델의 초기치를 계산한 후, 취득 이미지로부터 배경 분리를 행하는 루프에 들어간다.

// (8)배경이 될 수 있는 화소의 휘도치의 범위를 체크한다

이미지의 각 화소에 대해 (을)를 계산해,cvInRange()그리고 값이 범위내인지 어떤지를 비교해,msk\_img 에 결과를 보존해 돌려준다. 범위내의 화소에 관해서는 msk\_img 의 대응하는 화소치가 0xFF(으)로서 세트 된다.

// (9)휘도 진폭을 재계산한다

(5)(와)과 같게 해 각 화소의 휘도 진폭 (을)를 재계산한다.

// (10)배경이라고 판단된 영역의 배경의 휘도 평균과 휘도 진폭을 갱신한다

(8)그리고 돌려주어지는 마스크 이미지를 이용하고, 배경 영역만 cvRunningAvg()함수를 이용해 휘도 평균과 휘도 진폭을 갱신한다.

// (11)물체 영역이라고 판단된 영역에서는 휘도 진폭만을(배경 영역보다 늦은 속도로) 갱신한다  
물체 영역의 휘도 진폭을 갱신하기 위해서, 마스크 이미지를 반전시킨다.그 후(10)(와)과 같게 해 휘도 진폭을 갱신한다. 여기서 사용하는 갱신 속도 파라미터는 배경 영역 갱신에 사용하는 파라미터보다 크지 않으면 안 된다.

// (12)물체 영역만을 출력 이미지에 카피한다(배경 영역은 흑)

처리 결과를 나타내기 위해서, 이미지 dst\_img 에 대해서,frame(으)로부터 마스크에 따라서 화소치를 카피한다.

// (13)처리 결과를 표시한다

입력 이미지와 처리 결과를 표시한다.캡처 안에"Esc"키가 밀렸을 경우는, 종료한다.

### 실행 결과예



CLOSE(왼쪽) 입력 이미지, ( 오른쪽) 물체 검출 결과

## 동적 윤곽 추적

동적 윤곽 추적이란, 변형하는 윤곽(물체 형상 그 자체의 윤곽이 변형하지 않는 경우에서도, 이미지에 투영 된 윤곽이 변형하는 경우가 있다) (을)를 순서대로 계산하는(많은 경우는 반복 계산) 수법이다. 추적 대상이 되는 물체는, 그러한 수법에 의해, 삼차원 형상 물체, 변형 물체, 관절 구조체 등 여러가지이다.

Opencv그리고 실장되고 있다snake(은)는, 동적 윤곽 추적 수법의 가장 기본적인 방법의 하나이다. snake그럼, 내부 에너지와 외부 에너지의 화가 최소가 되는 윤곽을 요구한다. 적절한 윤곽을 얻기 위해서는 각 에너지의 중량감이 되는 파라미터가 중요한 역할을 완수하지만, 이러한 파라미터를 결정하기 위해서는, 추적 물체 형상이나 배경 이미지의 특징이 어느 정도 기준일 필요가 있다. 또, 기본적인snake그럼, 탐색 공간이 이차원 이미지 그 자체로 나타내지는 공간이므로, 윤곽의 자유도가 높은 분, 이미지자등의 외란의 영향을 받기 쉽다.

샘플

---

## snake 에 의한 윤곽 추적(정지화면) cvSnakelImage

snake 에 의해 동적 윤곽 추적을 실시한다(이번은 대상이 정지화면이므로, 어느 쪽인가 하면 윤곽 검출)

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <math.h>
#include <stdio.h>

typedef struct parameter Parameter;
struct parameter {
    float alpha;
    float beta;
    float gamma;
};

int
main (int argc, char **argv)
{
    int i, j = 0, c;
    IplImage *src_img, *dst_img;
    CvPoint *contour;
    CvPoint center;
    int length = 60;                /* 동적 윤곽을 구성하는 점수 */
    Parameter param = { 0.45, 0.35, 0.2 };    /* cvSnakelImage 의 파라미터 */
    CvFont font;
    char iter[8];

    // (1)이미지를 읽어들인다
```

```

if (argc < 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
    return -1;
dst_img = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 3);

cvInitFont (&font, CV_FONT_HERSHEY_DUPLEX, 0.7, 0.7);
center.x = src_img->width / 2;
center.y = src_img->height / 2;
// (2)동적 윤곽의 초기화
contour = (CvPoint *) cvAlloc (sizeof (CvPoint) * length);
for (i = 0; i < length; i++) {
    contour[i].x = (int) (center.x * cos (2 * CV_PI * i / length) + center.x);
    contour[i].y = (int) (center.y * sin (2 * CV_PI * i / length) + center.y);
}

// (3)초기 윤곽의 그리기
cvCvtColor (src_img, dst_img, CV_GRAY2RGB);
for (i = 0; i < length - 1; i++) {
    cvLine (dst_img, contour[i], contour[i + 1], CV_RGB (255, 0, 0), 2, 8, 0);
}
cvLine (dst_img, contour[length - 1], contour[0], CV_RGB (255, 0, 0), 2, 8, 0);
cvNamedWindow ("Snakes", CV_WINDOW_AUTOSIZE);
cvShowImage ("Snakes", dst_img);
cvWaitKey (0);

/* 동적 윤곽의 수습 계산(과정을 표시한다) */
while (1) {

    // (4)동적 윤곽의 윤곽 계산
    cvSnakeImage (src_img, contour, length, ¶m.alpha, ¶m.beta, ¶m.gamma,
                  CV_VALUE, cvSize (15, 15), cvTermCriteria (CV_TERMCRIT_ITER, 1, 0.0),
1);

    // (5)계산된 동적 윤곽의 그리기
    cvCvtColor (src_img, dst_img, CV_GRAY2RGB);
    for (i = 0; i < length - 1; i++) {
        cvLine (dst_img, contour[i], contour[i + 1], CV_RGB (255, 0, 0), 2);
    }
    cvLine (dst_img, contour[length - 1], contour[0], CV_RGB (255, 0, 0), 2);
    sprintf (iter, 8, "%03d", ++j);
    cvPutText (dst_img, iter, cvPoint (15, 30), &font, CV_RGB (0, 0, 255));

    // (6)결과의 표시
    cvShowImage ("Snakes", dst_img);
    c = cvWaitKey (0);
    if (c == 'Wx1b')
        break;
}

```

```

}

cvDestroyWindow ("Snakes");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 0;
}

```

// (1)이미지를 읽어들인다

커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV\_LOAD\_IMAGE\_GRAYSCALE (을)를 지정하는 것으로, 원이미지를 그레이 스케일 이미지로서 읽어들인다. 또, 결과를 표시하기 위해서, 입력 이미지와 동일 사이즈의 칼라 이미지를 작성한다.

// (2)동적 윤곽의 초기화

초기 윤곽의 위치를 결정한다.본래는, 추적되는 물체의 형상에 응한 적절한 초기 윤곽이 설정 되는 것이 당연하지만, 이번은 단지, 주어진 이미지 구형에 내접 하는 타원을 초기 윤곽으로서 이용하고 있다.

// (3)초기 윤곽의 그리기

초기화된 윤곽을 그리기 하고, 실제로 표시한다.무엇인가 키가 압하될 때까지 기다린다.

// (4)동적 윤곽의 윤곽 계산

동적 윤곽의 윤곽을 계산한다.이 샘플에서는, 입력 이미지에 대한 사전 처리 등 (은)는 아무것도 가서 않지만, 실제로는, 예를 들면 노이즈의 제거나 콘트라스트의 강조, 혹은 엣지의 검출등이 사전 처리로서 행해지는 경우가 있다.

여기에서는, 윤곽이 수습하는 과정을 표시하기 위해서, 함수 cvSnakeImage()하지만 1 회의 반복 계산 밖에 실시하지 않게 파라미터를 설정해 있다(CvTermCriteria). 예를 들면, 이하와 같이 하면, 「100 회반복 계산을 하든가, 혹은 각 반복 계산에 대해 이동하는 점의 수 하지만 0 이하(모든 점이 이동하지 않게 된다)까지」, 함수 cvSnakeImage()안에서 반복 계산을 하고 나서 돌아간다. 즉, 통상은, 반복 계산을 위한 루프는 필요없다.

```

cvSnakeImage( src_img, contour, length, ¶m.alpha, ¶m.beta, ¶m.gamma,
              CV_VALUE, cvSize(15, 15), cvTermCriteria( CV_TERMCRIT_ITER, 100, 0.0), 1);

```

또, 구배 플래그(함수cvSnakeImage()의 마지막 인수)가1그래서, 이미지 구배가 에너지장으로서 이용된다. 그 외, 각 인수의 상세한 것에 대하여는, 참조 설명서를 참조.

// (5)계산된 동적 윤곽의 그리기

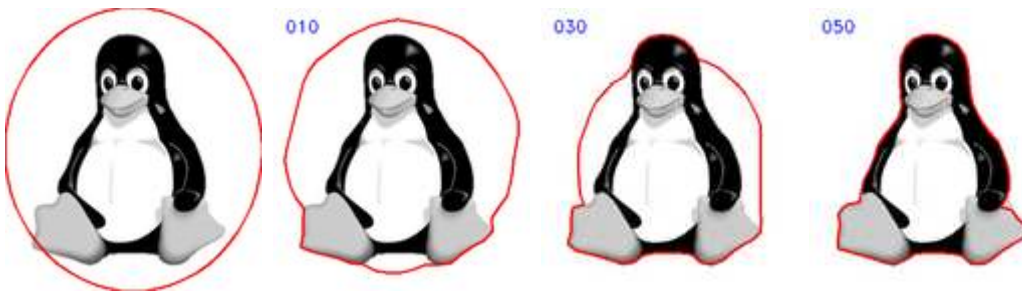
계산되어 갱신된 윤곽(여기에서는,1 스텝분 ), 및, 현재의 반복수를 이미지상에 그리기 한다.

// (6)결과의 표시

처리 결과를 표시한다. "Esc"키가 밀렸을 경우는 종료해, 그 이외가 밀렸을 경우는, 다음의 반복 계산에 들어간다.

## 실행 결과예

초기 위치	iter=10	iter=30	iter=50
-------	---------	---------	---------



## 옵티컬 플로우

옵티컬 플로우는, 시간 연속인 이미지열을 이용하고, 이미지의 속도장(물체 의 속도 + 카메라의 속도)를 요구해 그것을 벡터 집합으로 표현한 것이다. 크게 나누고, 구배법, 블록 매칭법이 존재한다.

구배법에서는, 「옵티컬 플로우 구속 방정식」이라고 불리는, 휘도의 시간/공간적 미분(휘도 구배)의 구속 방정식을 이용하고, 이것에 제약 조건 (을)를 추가하는 것으로 플로우를 요구한다. 비교적 고속으로 전화소에 대한 속도장을 계산할 수 있지만, 전제 조건에 맞지 않는 개 곳(급격한 휘도 변화, 노이즈)에서는, 현저한 오차가 발생하는 것이 있다. 블록 매칭법에서는, 이미지중이 있는 블록을 템플릿으로서 다음 시간의 이미지중으로부터 매치하는 곳을 탐색하는 것으로 플로우를 요구한다. 적절한 특징을 가지는 이미지에 대해서는, 로바스트에 플로우를 계산할 수 있지만, 비교적 계산 시간이 걸린다.또, 회전, 슬캘링에 대한 로바스트성은, 템플릿이 되는 블록의 사이즈나 이미지의 특징에 의존한다.

## 샘플

## 옵티컬 플로우 1 cvCalcOpticalFlowHS, cvCalcOpticalFlowLK

구배법에 따르는 옵티컬 플로우의 계산

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, j, dx, dy, rows, cols;
    IplImage *src_img1, *src_img2, *dst_img1, *dst_img2;
    CvMat *velx, *vely;
    CvTermCriteria criteria;

    if (argc != 3 ||
        (src_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ||
        (src_img2 = cvLoadImage (argv[2], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
        return -1;

    dst_img1 = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR);
    dst_img2 = (IplImage *) cvClone (dst_img1);
```

```

// (1)속도벡터를 격납하는 구조체의 확보, 등
cols = src_img1->width;
rows = src_img1->height;
velx = cvCreateMat (rows, cols, CV_32FC1);
vely = cvCreateMat (rows, cols, CV_32FC1);
cvSetZero (velx);
cvSetZero (vely);
criteria = cvTermCriteria (CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 64, 0.01);

// (2)옵티컬 플로우를 계산(HS)
cvCalcOpticalFlowHS (src_img1, src_img2, 0, velx, vely, 100.0, criteria);
// (3)옵티컬 플로우를 그리기(HS)
for (i = 0; i < cols; i += 5) {
    for (j = 0; j < rows; j += 5) {
        dx = (int) cvGetReal2D (velx, j, i);
        dy = (int) cvGetReal2D (vely, j, i);
        cvLine (dst_img1, cvPoint (i, j), cvPoint (i + dx, j + dy), CV_RGB (255, 0, 0), 1,
CV_AA, 0);
    }
}

// (4)옵티컬 플로우를 계산(LK)
cvCalcOpticalFlowLK (src_img1, src_img2, cvSize (15, 15), velx, vely);
// (5)계산된 플로우를 그리기(LK)
for (i = 0; i < cols; i += 5) {
    for (j = 0; j < rows; j += 5) {
        dx = (int) cvGetReal2D (velx, j, i);
        dy = (int) cvGetReal2D (vely, j, i);
        cvLine (dst_img2, cvPoint (i, j), cvPoint (i + dx, j + dy), CV_RGB (255, 0, 0), 1,
CV_AA, 0);
    }
}

// (6)옵티컬 플로우의 표시
cvNamedWindow ("ImageHS", 1);
cvShowImage ("ImageHS", dst_img1);
cvNamedWindow ("ImageLK", 1);
cvShowImage ("ImageLK", dst_img2);
cvWaitKey (0);

cvDestroyWindow ("ImageHS");
cvDestroyWindow ("ImageLK");
cvReleaseImage (&src_img1);
cvReleaseImage (&src_img2);
cvReleaseImage (&dst_img1);

```



```

cvReleaseImage (&dst_img2);
cvReleaseMat (&velx);
cvReleaseMat (&vely);

return 0;
}

```

// (1)속도벡터를 격납하는 구조체의 확보, 등  
입력 이미지와 같은 사이즈의 CvArr(을)를 확보한다.이번은,cvMat(을)를 이용하고 있지만,  
IplImage 등에서도 상관없다. 또, 함수 cvCalcOpticalFlowHS()그리고 이용되는 종료 조건  
구조체도 확보한다.

// (2)옵티컬 플로우를 계산(HS)  
Horn & Schunck 알고리즘을 이용해 옵티컬 플로우를 계산한다.

// (3)옵티컬 플로우를 그리기(HS)  
X 방향,y 방향 모두 5[pixel]마다, 함수 cvGetReal2D()(을)를 이용하고, 인덱스(좌표)를 지정하고  
속도벡터를 꺼내, 그것을 입력 이미지상에 그리기 한다.

// (4)옵티컬 플로우를 계산(LK)  
Lucas & Kanade 알고리즘을 이용해 옵티컬 플로우를 계산한다.

// (5)계산된 플로우를 그리기(LK)  
X 방향,y 방향 모두 5[pixel]마다, 함수 cvGetReal2D()(을)를 이용하고, 인덱스(좌표)를 지정하고  
속도벡터를 꺼내, 그것을 입력 이미지상에 그리기 한다.

// (6)옵티컬 플로우의 표시  
상기의 두 개의 수법에 의해 계산된 옵티컬 플로우를, 입력 이미지상에 그리기 한 것을 실제로  
표시한다. 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



## 옵티컬 플로우 2 cvCalcOpticalFlowBM

블록 매칭에 의한 옵티컬 플로우의 계산

#### 샘플 코드

표시의 변환

```

#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i, j, dx, dy, rows, cols;
    int block_size = 10;
    int shift_size = 1;
    CvMat *velx, *vely;
    CvSize block = cvSize (block_size, block_size);
    CvSize shift = cvSize (shift_size, shift_size);
    CvSize max_range = cvSize (50, 50);
    IplImage *src_img1, *src_img2, *dst_img;

    if (argc != 3 ||
        (src_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ||
        (src_img2 = cvLoadImage (argv[2], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
        return -1;
    dst_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR);

    // (1)속도벡터를 격납하는 구조체의 확보
    rows = int (ceil (double (src_img1->height) / block_size));
    cols = int (ceil (double (src_img1->width) / block_size));
    velx = cvCreateMat (rows, cols, CV_32FC1);
    vely = cvCreateMat (rows, cols, CV_32FC1);
    cvSetZero (velx);
    cvSetZero (vely);

    // (2)옵티컬 플로우의 계산
    cvCalcOpticalFlowBM (src_img1, src_img2, block, shift, max_range, 0, velx, vely);

    // (3)계산된 플로우를 그리기
    for (i = 0; i < velx->width; i++){
        for (j = 0; j < vely->height; j++) {
            dx = (int) cvGetReal2D (velx, j, i);
            dy = (int) cvGetReal2D (vely, j, i);
            cvLine (dst_img, cvPoint (i * block_size, j * block_size),
                    cvPoint (i * block_size + dx, j * block_size + dy), CV_RGB (255, 0, 0), 1,
                    CV_AA, 0);
        }
    }

    cvNamedWindow ("Image", 1);
    cvShowImage ("Image", dst_img);
}

```

```

cvWaitKey (0);

cvDestroyWindow ("Image");
cvReleaseImage (&src_img1);
cvReleaseImage (&src_img2);
cvReleaseImage (&dst_img);
cvReleaseMat (&velx);
cvReleaseMat (&vely);

return 0;
}

```

// (1)속도벡터를 격납하는 구조체의 확보

사이즈  $\text{ceil}(\text{src.width}/\text{block\_size}) \times \text{ceil}(\text{src.height}/\text{block\_size})$ , 32 비트, 싱글 채널의, 속도벡터를 격납하는 구조체를 확보한다.

// (2)옵티컬 플로우의 계산

블록 매칭에 의해, 옵티컬 플로우를 계산한다. 3 번째의 인수는, 블록(템플릿) 사이즈를 나타낸다. 4 번째의 인수는, 5 번째의 인수로 나타내지는 하나의 블록에 대한 탐색 범위내를, 얼마나 늦추어져서면서 매칭을 실시하는지를 나타낸다.

// (3)계산된 플로우를 그리기

함수 `cvGetReal2D()`(을)를 이용하고, 인덱스(좌표)를 지정하고 속도벡터를 꺼내, 그것을 입력 이미지상에 그리기 한다.

#### 실행 결과예





## 옵티컬 플로우 3 cvCalcOpticalFlowPyrLK

드문드문한 특징에 대한 옵티컬 플로우의 계산

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    char *status;
    int i, corner_count = 150;
    CvPoint2D32f *corners1, *corners2;
    CvTermCriteria criteria;
    IplImage *src_img1, *src_img2, *dst_img;
    IplImage *eig_img, *temp_img;
    IplImage *prev_pyramid, *curr_pyramid;

    if (argc != 3 ||
        (src_img1 = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ||
        (src_img2 = cvLoadImage (argv[2], CV_LOAD_IMAGE_GRAYSCALE)) == 0)
        return -1;

    dst_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR);

    // (1)필요한 구조체의 확보
    eig_img = cvCreateImage (cvGetSize (src_img1), IPL_DEPTH_32F, 1);
    temp_img = cvCreateImage (cvGetSize (src_img1), IPL_DEPTH_32F, 1);
    corners1 = (CvPoint2D32f *) cvAlloc (corner_count * sizeof (CvPoint2D32f));
    corners2 = (CvPoint2D32f *) cvAlloc (corner_count * sizeof (CvPoint2D32f));
    prev_pyramid = cvCreateImage (cvSize (src_img1->width + 8, src_img1->height / 3),
    IPL_DEPTH_8U, 1);
    curr_pyramid = cvCreateImage (cvSize (src_img1->width + 8, src_img1->height / 3),
    IPL_DEPTH_8U, 1);
```

```

status = (char *) cvAlloc (corner_count);
criteria = cvTermCriteria (CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 64, 0.01);

// (2)드문드문한 특징점을 검출
cvGoodFeaturesToTrack (src_img1, eig_img, temp_img, corners1, &corner_count, 0.001, 5,
NULL);

// (3)옵티컬 플로우를 계산
cvCalcOpticalFlowPyrLK (src_img1, src_img2, prev_pyramid, curr_pyramid,
                        corners1, corners2, corner_count, cvSize (10, 10), 4, status,
NULL, criteria, 0);

// (4)계산된 플로우를 그리기
for (i = 0; i < corner_count; i++) {
    if (status[i])
        cvLine (dst_img, cvPointFrom32f (corners1[i]), cvPointFrom32f (corners2[i]), CV_RGB
(255, 0, 0), 1, CV_AA, 0);
}
cvNamedWindow ("Image", 1);
cvShowImage ("Image", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Image");
cvReleaseImage (&src_img1);
cvReleaseImage (&src_img2);
cvReleaseImage (&dst_img);
cvReleaseImage (&eig_img);
cvReleaseImage (&temp_img);
cvReleaseImage (&prev_pyramid);
cvReleaseImage (&curr_pyramid);

return 0;
}

```

// (1)필요한 구조체의 확보

함수 cvGoodFeaturesToTrack()및, 함수 cvCalcOpticalFlowPyrLK()에 필요 구조체를 확보한다.

// (2)드문드문한 특징점을 검출

함수 cvGoodFeaturesToTrack()(을)를 이용하고, 이미지중에서 큰 고유치를 가지는 코너 (분명히  
한 특징점)(을)를 찾아낸다. 검출된 코너는,4 번째의 인수로 나타나는 배열에 격납되어 그 개수는,  
5 번째의 인수로 나타나는 변수에 격납된다. 6 번째 , 및 7 번째의 인수는 각각, 특징점의 질과  
특징점끼리의 최저 거리를 나타내고 있어 이러한 값이 작아지면 검출되는 특징점이 증가한다.  
상세한 것에 대하여는, 레퍼런스를 참조하는 것.

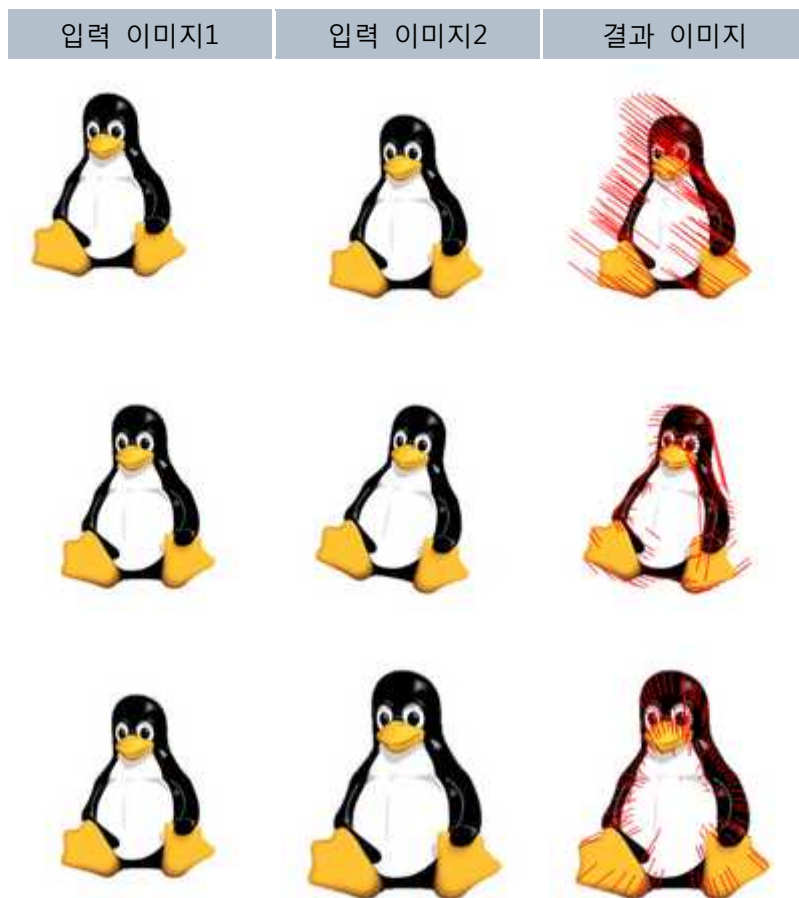
// (3)옵티컬 플로우를 계산

앞의 함수에 의해 검출된 드문드문한 특징점에 관해서, 옵티컬 플로우를 계산한다.

// (4)계산된 플로우를 그리기

플로우가 발견된 점 에 대해서만, 그 플로우를 그리기 한다.

#### 실행 결과예



#### Condensation

OpenCV에는, 추정기의 하나로써Condensation(파티클 필터)(이)가 실장되고 있다. 레퍼런스 메뉴얼에도 있도록(듯이), 알고리즘의 자세한 것은, [http://www.dai.ed.ac.uk/CVonline/LOCAL\\_COPIES/ISARD1/condensation.html](http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/ISARD1/condensation.html) (을)를 참조되고 싶다.

여기에서는, 특히OpenCV에 실장되고 있다Condensation알고리즘의 함수의 사용 방법에 대해 말한다.

샘플

## Condensation

Condensation 알고리즘을 이용해 이미지중의 붉은 영역을 트래킹 한다

#### 샘플 코드

표시의 변환

```
#include <cv.h>
```

```

#include <highgui.h>
#include <ctype.h>
#include <math.h>

// (1)우도의 계산을 행하는 함수
float
calc_likelihood (IplImage * img, int x, int y)
{
    float b, g, r;
    float dist = 0.0, sigma = 50.0;

    b = img->imageData[img->widthStep * y + x * 3];    //B
    g = img->imageData[img->widthStep * y + x * 3 + 1]; //G
    r = img->imageData[img->widthStep * y + x * 3 + 2]; //R
    dist = sqrt (b * b + g * g + (255.0 - r) * (255.0 - r));

    return 1.0 / (sqrt (2.0 * CV_PI) * sigma) * expf (-dist * dist / (2.0 * sigma *
sigma));
}

int
main (int argc, char **argv)
{
    int i, c;
    double w = 0.0, h = 0.0;
    CvCapture *capture = 0;
    IplImage *frame = 0;

    int n_stat = 4;
    int n_particle = 4000;
    CvConDensation *cond = 0;
    CvMat *lowerBound = 0;
    CvMat *upperBound = 0;

    int xx, yy;

    // (2)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다
    if (argc == 1 || (argc == 2 && strlen (argv[1]) == 1 && isdigit (argv[1][0])))
        capture = cvCreateCameraCapture (argc == 2 ? argv[1][0] - '0' : 0);

    // (3)1 프레임 캡처 해, 캡처사이즈를 취득한다.
    frame = cvQueryFrame (capture);
    w = frame->width;
    h = frame->height;

```

```

cvNamedWindow ("Condensation", CV_WINDOW_AUTOSIZE);

// (4)Condensation 구조체를 작성한다.
cond = cvCreateConDensation (n_stat, 0, n_particle);

// (5)상태 벡터 각 차원이 취할 수 있는 최소치 · 최대치를 지정한다.
lowerBound = cvCreateMat (4, 1, CV_32FC1);
upperBound = cvCreateMat (4, 1, CV_32FC1);

cvmSet (lowerBound, 0, 0, 0.0);
cvmSet (lowerBound, 1, 0, 0.0);
cvmSet (lowerBound, 2, 0, -10.0);
cvmSet (lowerBound, 3, 0, -10.0);
cvmSet (upperBound, 0, 0, w);
cvmSet (upperBound, 1, 0, h);
cvmSet (upperBound, 2, 0, 10.0);
cvmSet (upperBound, 3, 0, 10.0);

// (6)Condensation 구조체를 초기화한다
cvConDensInitSampleSet (cond, lowerBound, upperBound);

// (7)ConDensation 알고리즘에 있어서의 상태 벡터의 다이내믹스를 지정한다
cond->DynamMatr[0] = 1.0;
cond->DynamMatr[1] = 0.0;
cond->DynamMatr[2] = 1.0;
cond->DynamMatr[3] = 0.0;
cond->DynamMatr[4] = 0.0;
cond->DynamMatr[5] = 1.0;
cond->DynamMatr[6] = 0.0;
cond->DynamMatr[7] = 1.0;
cond->DynamMatr[8] = 0.0;
cond->DynamMatr[9] = 0.0;
cond->DynamMatr[10] = 1.0;
cond->DynamMatr[11] = 0.0;
cond->DynamMatr[12] = 0.0;
cond->DynamMatr[13] = 0.0;
cond->DynamMatr[14] = 0.0;
cond->DynamMatr[15] = 1.0;

// (8)노이즈 파라미터를 재설정한다.
cvRandInit (&(cond->RandS[0]), -25, 25, 0);
cvRandInit (&(cond->RandS[1]), -25, 25, 1);
cvRandInit (&(cond->RandS[2]), -5, 5, 2);
cvRandInit (&(cond->RandS[3]), -5, 5, 3);

while (1) {

```



```

frame = cvQueryFrame (capture);

// (9)각 파티클에 대해 우도를 계산한다.
for (i = 0; i < n_particle; i++) {
    xx = (int) (cond->fISamples[i][0]);
    yy = (int) (cond->fISamples[i][1]);
    if (xx < 0 || xx >= w || yy < 0 || yy >= h) {
        cond->fIConfidence[i] = 0.0;
    }
    else {
        cond->fIConfidence[i] = calc_likelihood (frame, xx, yy);
        cvCircle (frame, cvPoint (xx, yy), 2, CV_RGB (0, 0, 255), -1);
    }
}

cvShowImage ("Condensation", frame);
c = cvWaitKey (10);
if (c == 'Wx1b')
    break;

// (10)다음의 모델 상태를 추정한다
cvConDensUpdateByTime (cond);

}

cvDestroyWindow ("Condensation");
cvReleaseImage (&frame);
cvReleaseCapture (&capture);
cvReleaseConDensation (&cond);
cvReleaseMat (&lowerBound);
cvReleaseMat (&upperBound);

return 0;
}

```

// (1)우도의 계산을 행하는 함수

Condensation 알고리즘에서는, 각 파티클의 우도를 줄 필요가 있다. 샘플 코드에서는, 각 파티클의 위치의 정보로부터, 그 픽셀치의 빨강인것 같음을 우도로서 이용하는 일로 했다.직감적인 알기 쉬움을 위해서 RGB 겹(표)색계인 채 계산을 행한다.구체적으로는 이하와

같이, $R(255,0,0)$ (으)로부터의 Euclid 거리      에 대해서, $0$ (을)를 평균,  $\sigma$ (을)를 분산

(으)로서 가지는 정규 분포를 우도 함수

(으)로서 정의하고 있다.

샘플에서는 분산sigma=50.0(으)로 하고 있지만, 이 파라미터를 튜닝 하는 일에 따라서 트랙킹 성능이 바뀌므로, 각자 확인되고 싶다.

적색인것 같음 평가에는, 샘플과 같이RGB결(표)색계는 아니고HSV결(표)색계로 행하는 것이 밝기의 변화에 대해서 로 바스트인 결과를 얻을 수 있다.그러나, 샘플에서는 위에서 설명한 바와 같이 알고리즘의 직감적인 이해를 위해서RGB 결(표)색계로 처리를 행하고 있다.

// (2)커멘드 인수에 의해서 지정된 번호의 카메라에 대한다 캡처 구조체를 작성한다

함수 cvCreateCameraCapture()(을)를 이용하고, 커멘드 인수로서 주어진 번호의 카메라에 대한 캡처 구조체 CvCapture(을)를 작성한다. 인수가 지정되지 않는 경우는, 디폴트의 값이다"0"하지만 이용된다.

// (3)1 프레임 캡처 해, 캡처사이즈 (을)를 취득한다.

함수 cvQueryFrame() (을)를 호출해, 실제로 1 프레이 무캡처를 실시해, 취득한 이미지로부터, 이미지의 폭 w(와)과 높이 h(을)를 취득한다.

// (4)Condensation 구조체를 작성한다.

함수 cvCreateConDensation() (을)를, 상태 벡터의 차원수 n\_stat(와)과 파티클의 수 n\_particle(을)를 지정해 호출해,Condensatopn 구조체를 작성한다. 제 2 인수는, 관측 벡터수를 지정하는 일이 되어 있지만,OpenCV 의 현버전에서는 관측 벡터는 사용하고 있지 않기 때문에, 값에는 0(을)를 지정했으므로 상관없다.

샘플 코드에서는, 다이내믹스로서 이하와 같이 등속 직선운동을 가정하고 있으므로, 상태 벡터는 그 위치(x,y)(와)과 각 축방향의 속도(u,v)의 4 차원이 된다.

// (5)상태 벡터 각 차원이 취할 수 있는 최소치·최대치를 지정한다.

벡터 lowerBound, upperBound (을)를 각각 함수 cvCreateMat()그리고 파티션 해, 상태 벡터의 각 차원의 변수가 취할 수 있는 값의 하한치(최소치)와 상한치(최대치)를 설정한다. 이 값은, 초기치 설정 시에는 파티클을 일 모양 분포시키기 위해서 사용된다.

샘플 코드에서는, 위치에 대해서 최소치 0, 최대치는 이미지 사이즈(w,h)(을)를 지정, 또 속도에 대해 -10~10[pixel](을)를 지정해 있다.

// (6)Condensation 구조체를 초기화한다

함수 cvConDensInitSampleSet() 그리고, 구조체의 초기화, 초기 파티클 상태의 설정을 행한다.

// (7)ConDensation 알고리즘에 있어서의 상태 벡터의 다이내믹스를 지정한다

CvConDensation 구조체의 멤버 DynamMatr[]에 다이내믹스를 표현하는 행렬 요소를 순서로 지정한다.

샘플 코드에서는,(4)그리고 나타내 보인 등속 직선운동을 이하와 같이 행렬 표현했을 때의 4X4 의 배열을 주고 있다.

모델의 다이내믹스로서 랜덤 워크를 가정하는 경우는, 여기서 단위행렬을 지정한다.

// (8)노이즈 파라미터를 재설정한다.

초기화 시에, 상태 벡터의 각 차원의 변수에 대한 노이즈 파라미터는 자동적으로 설정된다 (상한치, 하한치 각각의 범위의 1/5의 값이 된다)의로, 그것이 무제한 경우는 여기서 재설정한다. 샘플 코드에서는, 위치에 관해서는-25~25[pixel], 속도에 관해서는-5~5[pixel]의 범위의 노이즈(을)를 지정해 있다.

// (9)각 파티클에 대해 우도를 계산한다.

루프 블록중에서 함수 cvQueryFrame() (을)를 호출해, 이미지를 취득한다. 그 후, 각 파티클의 위치(샘플 코드에서는,CvConDensation 구조체의 flSamples[i][0](와)과[i][1]) (을)를 이용해 그 우도를(1)그리고 정의한 우도 함수를 이용해 계산한다.

파티클의 위치가 이미지 범위내이면, 푸른 원을 캡처 한 이미지상에 추가해, 이미지를 그리기 한다. "Esc"키가 밀렸을 경우는, 트래킹을 종료한다.

// (10)다음의 모델 상태를 추정한다

함수 cvConDensUpdateByTime()(을)를 이용하고, 다음의 시각의 모델 상태를 추정한다.

condensation 알고리즘에 있어서의, 리산폴·이동·확산의 국면을 행하고 있다 (이)라고 생각해도 좋다.

#### 실행 결과예



#### CLOSE

#### 물체 검출

OpenCV그리고 준비되어 있는 물체 검출은, (Haar-like특징을 이용한) 부스트 된 약분류기의 캐스케이드를 이용하고 있다. 세세한 알고리즘에 관해서는, 레퍼런스 및 그 외의 참고 문헌에 양보하지만, 분류기는 대략적으로 이하와 같이 구성되어 있다.

- Haar-like 특징을 입력으로 취하는 결정목을, 기본 분류기로 한다.
- 부스 텅 기법을 이용하고, 몇개의 기본 분류기를 복합시켜 스테이지 분류기가 구성된다.
- 스테이지 분류기가 죽 늘어서 묶은 것에 연결되어 최종적인 캐스케이드가 구성된다.

인식시에 입력되는 이미지는, 캐스케이드의 각 스테이지에 있어 평가되어 도중에 기각되면 그 부분 이미지에는 물체가 없는, 모든 스테이지를 패스하면, 그 부분 이미지는 물체를 포함하고 있다고 여겨진다. Haar-like특징은, (형상이나 국소적인 특징은 아니고) 전체적인 아피아 랑스를 이용하고 있으므로, 이 특징을 이용한 만큼 류기에도, 검출의 특징, 서툼이 존재한다. 예를 들면, 다소 흔들린 이미지나 희미해진 이미지에서도, (대략의 texture)의 대국적인 특징은 변화하지 않기 때문에, 이것들을 검출할 수 있다. 그러나, 같은 종류에 대해서도 전체의 texture의 개체차이가 크고, 그것이 넓게 분포하는 오브젝트는, 그 형상이나 국소적인 특징이 얇고 있었다고 해도 검출하는 것은 곤란하다.

## 얼굴의 검출 cvHaarDetectObjects

미리 학습된 만큼 류기를 이용해 입력 이미지중의 얼굴을 검출한다

### 샘플 코드

[표시의 변환](#)
[Python판으로 변경](#)

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
    int i;
    IplImage *src_img = 0, *src_gray = 0;
    const char *cascade_name = "haarcascade_frontalface_default.xml";
    CvHaarClassifierCascade *cascade = 0;
    CvMemStorage *storage = 0;
    CvSeq *faces;
    static CvScalar colors[] = {
        {{0, 0, 255}}, {{0, 128, 255}},
        {{0, 255, 255}}, {{0, 255, 0}},
        {{255, 128, 0}}, {{255, 255, 0}},
        {{255, 0, 0}}, {{255, 0, 255}}
    };

    // (1)이미지를 읽어들인다
    if (argc < 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;
    src_gray = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);

    // (2)부스트 된 만큼 류기의 캐스케이드를 읽어들인다
    cascade = (CvHaarClassifierCascade *) cvLoad (cascade_name, 0, 0, 0);

    // (3)메모리를 확보해, 읽어들인 이미지의 그레이 스케일화, 히스토그램의 균일화를
    // 실시한다
    storage = cvCreateMemStorage (0);
    cvClearMemStorage (storage);
    cvCvtColor (src_img, src_gray, CV_BGR2GRAY);
    cvEqualizeHist (src_gray, src_gray);

    // (4)물체(얼굴) 검출
    faces = cvHaarDetectObjects (src_gray, cascade, storage, 1.11, 4, 0, cvSize (40, 40));
```

```

// (5)검출된 모든 얼굴 위치에, 엔을 그리기 한다
for (i = 0; i < (faces ? faces->total : 0); i++) {
    CvRect *r = (CvRect *) cvGetSeqElem (faces, i);
    CvPoint center;
    int radius;
    center.x = cvRound (r->x + r->width * 0.5);
    center.y = cvRound (r->y + r->height * 0.5);
    radius = cvRound ((r->width + r->height) * 0.25);
    cvCircle (src_img, center, radius, colors[i % 8], 3, 8, 0);
}

// (6)이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Face Detection", CV_WINDOW_AUTOSIZE);
cvShowImage ("Face Detection", src_img);
cvWaitKey (0);

cvDestroyWindow ("Face Detection");
cvReleaseImage (&src_img);
cvReleaseImage (&src_gray);
cvReleaseMemStorage (&storage);

return 0;
}

```

// (1)이미지를 읽어들인다

함수 cvLoadImage()(을)를 이용하고, 얼굴을 검출하는 대상이 되는(커멘드 인수로 지정된) 이미지를 읽어들인다. 2 번째의 인수에 CV\_LOAD\_IMAGE\_COLOR(을)를 지정하는 것으로, 칼라 이미지로서 읽어들인다. 검출 처리 자체는 그레이 스케일 이미지를 대상에 행해지지만, 검출 결과를 그리기 하기 위해서, 칼라 이미지로서 읽어들여 둔다. 또, 그 처리를 위해서, 입력이 상과 동사이즈의 그레이 스케일 이미지(IplImage)(을)를 작성해 둔다.

// (2)부스트 된 만큼 류기의 캐스케이드를 읽어들인다

학습에 의해서 미리 획득된, 분류기의 캐스케이드가 기술되었다 xml 파일을 읽어들인다. 여기에서는,OpenCV 의 샘플에 부속되는 정면얼굴 이미지 학습 결과의 1 개인,"haarcascade\_frontalface\_default.xml"(을)를 이용한다.

// (3)메모리를 확보해, 읽어들인 이미지의 그레이 스케일화, 히스토그램의 균일화를 실시한다  
실제의 얼굴 검출 시에 이용하는 메모리를 확보, 초기화한다. 또, 읽힌 칼라 이미지를 그레이 스케일 이미지로 변경해, 게다가 함수 cvEqualizeHist()에 의해, 그 히스토그램을 균일화한다.

// (4)물체(얼굴) 검출

함수 cvHaarDetectObjects()에 의해, 이미지로부터 얼굴을 검출한다. 4 번째의 인수(1.11)(은)는, 탐색 윈도우의 스케일 변화를 나타내고 있어 이번 경우는, 스케일 마다의 탐색에 대해서, 윈도우사이즈가 11[%]변화하는 일을 나타내고 있다. 또,5 번째의 인수는, 오브젝트를 구성하는 근방 구형의 최소수를 나타내고 있어 이것보다 적은 구형으로부터 구성되는 오브젝트는, 노이즈로서 무시된다. 즉, 실제의 오브젝트가 존재하는 부근에서는, 다소 어긋난 장소에 있어도

오브젝트를 포함한 영역으로서 인식될 것이므로, 1 개의 오브젝트는, 복수의 구형 영역의 집합으로서 나타내진다. 그렇지 않은 개소(예를 들어, 그 주변에 단일의 구형 밖에 존재하지 않는 오브젝트 후보)는, 우연히 캐스케이드를 패스한 영역으로서 무시된다.

// (5)검출된 모든 얼굴 위치에, 엔을 그리기 한다

검출된 모든 얼굴에 대해서, 사이즈와 중심을 요구해 그 자리소에, 엔을 그리기 한다. 엔의 그리기에는, 미리 정했다 8 색을 차례차례 이용한다.

// (6)이미지를 표시, 키가 밀렸을 때에 종료

검출 결과적으로 엔이 그리기 된 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예



### 카메라 calibration

카메라 calibration란, (어떤 시점에 있어) 카메라 고유의 내부 파라미터와 월드 좌표계에 있어서의 위치 자세를 의미하는 외부 파라미터를 요구하는 처리이다. 카메라의 calibration가 되면, 있다3차원 좌표를 가진 점이 카메라 이미지의 어디에 투영 되는지, 혹은 복수의 카메라에 투영 된 점이3차원 공간안의 어디에 있는지, 등을 계산할 수 있다. 또, 카메라 특유의(원주 방향 및 반경 방향) 일그러짐의 보정을 실시할 수도 성과 (가, 이것은 calibration 수법으로 의존해, 일그러짐을 가지지 않는 모델을 이용하는 단순한 수법도 존재한다). 적응할 수 있는 calibration 수법은, 카메라의 대수나 준비할 수 있는 치구에 의해서 변화한다. OpenCV의 카메라 calibration는,Z.Zhang의 수법을 기본으로 실장되고 있다 (Zhang의 수법에 대해서는,"A flexible new technique for camera calibration". IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330-1334, 2000.(을)를 참조되고 싶다). Ver.2(으)로부터(cvCalibrateCamera2등)은,Matlab의 calibration 엔진(이것도Zhang의 수법)을 이식한 것이 되었다.

샘플

## 카메라 calibration cvCalibrateCamera2, cvFindExtrinsicCameraParams2

Zhang 의 수법을 이용해 카메라의 calibration 를 실시해, 결과를 파일에 보존한다

### 샘플 코드

표시의 변환

```
#include <stdio.h>
#include <cv.h>
#include <highgui.h>

#define IMAGE_NUM (25)          /* 이미지수 */
#define PAT_ROW (7)             /* 패턴의 행수 */
#define PAT_COL (10)           /* 패턴의 열수 */
```

```

#define PAT_SIZE (PAT_ROW*PAT_COL)
#define ALL_POINTS (IMAGE_NUM*PAT_SIZE)
#define CHESS_SIZE (24.0) /* 패턴 1 매스의 1 옆사이즈[mm] */

int
main (int argc, char *argv[])
{
    int i, j, k;
    int corner_count, found;
    int p_count[IMAGE_NUM];
    IplImage *src_img[IMAGE_NUM];
    CvSize pattern_size = cvSize (PAT_COL, PAT_ROW);
    CvPoint3D32f objects[ALL_POINTS];
    CvPoint2D32f *corners = (CvPoint2D32f *) cvAlloc (sizeof (CvPoint2D32f) * ALL_POINTS);
    CvMat object_points;
    CvMat image_points;
    CvMat point_counts;
    CvMat *intrinsic = cvCreateMat (3, 3, CV_32FC1);
    CvMat *rotation = cvCreateMat (1, 3, CV_32FC1);
    CvMat *translation = cvCreateMat (1, 3, CV_32FC1);
    CvMat *distortion = cvCreateMat (1, 4, CV_32FC1);

    // (1)calibration 이미지의 읽기
    for (i = 0; i < IMAGE_NUM; i++) {
        char buf[32];
        sprintf (buf, "calib_img/%02d.png", i);
        src_img[i] = cvLoadImage (buf, CV_LOAD_IMAGE_COLOR);
    }

    // (2)3 차원 공간 좌표의 설정
    for (i = 0; i < IMAGE_NUM; i++) {
        for (j = 0; j < PAT_ROW; j++) {
            for (k = 0; k < PAT_COL; k++) {
                objects[i * PAT_SIZE + j * PAT_COL + k].x = j * CHESS_SIZE;
                objects[i * PAT_SIZE + j * PAT_COL + k].y = k * CHESS_SIZE;
                objects[i * PAT_SIZE + j * PAT_COL + k].z = 0.0;
            }
        }
    }

    cvInitMatHeader (&object_points, ALL_POINTS, 3, CV_32FC1, objects);

    // (3)체스 보드(calibration 패턴)의 코너 검출
    int found_num = 0;
    cvNamedWindow ("Calibration", CV_WINDOW_AUTOSIZE);
    for (i = 0; i < IMAGE_NUM; i++) {
        found = cvFindChessboardCorners (src_img[i], pattern_size, &corners[i * PAT_SIZE],

```

```

&corner_count);
    fprintf (stderr, "%02d...", i);
    if (found) {
        fprintf (stderr, "ok\n");
        found_num++;
    }
    else {
        fprintf (stderr, "fail\n");
    }
    // (4)코너 위치를 사브픽셀 정도에 수정, 그리기
    IplImage *src_gray = cvCreateImage (cvGetSize (src_img[i]), IPL_DEPTH_8U, 1);
    cvCvtColor (src_img[i], src_gray, CV_BGR2GRAY);
    cvFindCornerSubPix (src_gray, &corners[i * PAT_SIZE], corner_count,
                        cvSize (3, 3), cvSize (-1, -1), cvTermCriteria (CV_TERMCRIT_ITER
| CV_TERMCRIT_EPS, 20, 0.03));
    cvDrawChessboardCorners (src_img[i], pattern_size, &corners[i * PAT_SIZE],
corner_count, found);
    p_count[i] = corner_count;
    cvShowImage ("Calibration", src_img[i]);
    cvWaitKey (0);
}
cvDestroyWindow ("Calibration");

if (found_num != IMAGE_NUM)
    return -1;
cvInitMatHeader (&image_points, ALL_POINTS, 1, CV_32FC2, corners);
cvInitMatHeader (&point_counts, IMAGE_NUM, 1, CV_32SC1, p_count);

// (5)내부 파라미터, 일그러짐 계수의 추정
cvCalibrateCamera2 (&object_points, &image_points, &point_counts, cvSize (640, 480),
intrinsic, distortion);

// (6)외부 파라미터의 추정
CvMat sub_image_points, sub_object_points;
int base = 0;
cvGetRows (&image_points, &sub_image_points, base * PAT_SIZE, (base + 1) * PAT_SIZE);
cvGetRows (&object_points, &sub_object_points, base * PAT_SIZE, (base + 1) * PAT_SIZE);
cvFindExtrinsicCameraParams2 (&sub_object_points, &sub_image_points, intrinsic,
distortion, rotation, translation);

// (7)XML 파일에의 써내
CvFileStorage *fs;
fs = cvOpenFileStorage ("camera.xml", 0, CV_STORAGE_WRITE);
cvWrite (fs, "intrinsic", intrinsic);
cvWrite (fs, "rotation", rotation);
cvWrite (fs, "translation", translation);

```



```

cvWrite (fs, "distortion", distortion);
cvReleaseFileStorage (&fs);

for (i = 0; i < IMAGE_NUM; i++) {
    cvReleaseImage (&src_img[i]);
}

return 0;
}

```

// (1)calibration 이미지의 읽기

calibration 에 필요한 이미지열을 읽어들인다.여기에서는,calib\_img 이하에 미리 준비되어 있는,"00.png"~"24.png"(이)라는 이름의 이미지 파일을 이용했다. zhang 의 calibration 에 필요한 이미지 매수(뷰의 수)는 정해져 있지 않지만, 일반적으로는, 치우침등이 없는 적절한 이미지를 수십매 정도 준비하면 충분한 정도를 얻을 수 있다.

// (2)3 차원 공간 좌표의 설정

이번은,1 매스의 길이가 24.0[mm], 안쪽의 코너수가 7×10 의 체스 보드 패턴을 이용했다 ([실제의 체스 보드 패턴 파일](#)). 후술 하는 이유에 의해, 코너수는, 홀수×짝수, 혹은 짝수×홀수의 편성을 이용하는 것이 바람직하다. x 및 y 의 값은, 패턴의 매스의 사이즈에 맞추어 값을 대입한다. 여기서 설정되는 축방향은, 카메라의 외부 파라미터에 영향을 준다. 또,z(은)는 모두 0 또는 1(으)로 설정할 필요가 있다.

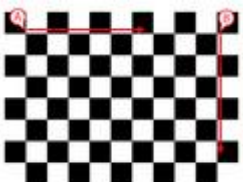
// (3)체스 보드(calibration 패턴)의 코너 검출

함수 cvFindChessboardCorners()에 의해, 체스 보드의 코너를 검출한다. 코너는 좌하로부터 우상을 향해서 검출되어 제 2 인수로 설정된 사이즈의 코너가 모두 검출되었을 경우에는,0 이외의 값을 돌려준다.

덧붙여서, 본프로그램과는 너무 관계없지만,2 매이상의 calibration 패턴을 인식시키고 싶은 경우는, (종횡의 구형수가) 다른 타입의 calibration 패턴을 준비하는, 혹은, 하나의 패턴을 검출할 때마다 거기를 전부 칠해 재탐색하는, 등의 수법을 생각할 수 있다.

OpenCV 의 함수 cvFindChessboardCorners()하지만 검출하는 코너의 차례는, 패턴의 행과 열의 정의에 의존한다. 이하의 이미지 체스 패턴을, 샘플 프로그램과 같이 7×10(7 행 10 열)이라고 정의하면, 검출 코너의 최초, 즉 calibration 결과의 월드 좌표 원점은, 이미지의(A)의 장소가 된다. 또, 같은 이미지를 10×7(10 행 7 열)이라고 정의하면, 원점은(B)된다.

함수 cvFindChessboardCorners()(은)는, 체스 패턴의 코너를 검출하기 위해서 구형의 검출을 실시하지만, 구형(흑색)에 비스듬하게 인접하는 구형의 수가 1 개의 것이 있으면(이미지중의 좌상과 우상), 그 구형이 포함한 코너의 최초의 점으로 한다.그것이 존재하지 않는 경우는, 기울기 인접 구형수가 2 개의 것을 이용한다. 코너수가, 홀수×짝수, 혹은 짝수×홀수의 편성의 체스 패턴이라면, 그 최초의 코너가 일의로 결정되지만, 그렇지 않은 경우는, 코너가 일의로 결정되지 않고, 체스 패턴의 이미지에 따라서는 검출되는 코너의 차례가 달라 버릴 가능성이 있다.



// (4)코너 위치를 사브픽셀 정도에 수정, 그리기

함수 `cvFindChessboardCorners()`냄새나도,0.5[pixel]단위로 코너가 검출되지만, 한층 더 검출된 코너를 사브픽셀 정도로 수정한다. 또, 그러한 코너를 이미지상에 그리기 해, 실제로 표시한다. 모든 코너가 올바르게 검출되었을 경우는 내부에서 정의되었다 7 색으로 그리기 되어 그렇지 않은 경우는 모든 코너가 적색으로 그리기 된다. 이 샘플 코드에서는, 모든 이미지(25 매)로 코너가 올바르게 검출되지 않는 경우는, 프로그램이 종료한다.

// (5)내부 파라미터, 일그러짐 계수의 추정

함수 `cvCalibrateCamera2()`에 의해, 카메라의 내부 파라미터, 및 일그러짐 계수를 추정한다. 내부 파라미터는, 이하의 식의 **A**에 해당한다.

// (6)외부 파라미터의 추정

카메라의 외부 파라미터를 추정한다.이것은, 상술의 식의 **[R|t]**에 해당한다. 즉, 삼차원이 있는 월드 좌표계 원점으로부터, 카메라 좌표계에의 변환을 나타내는 파라미터이다. 월드 좌표계 원점으로부터의 회전 및 병진을 나타내는 외부 파라미터를 결정하기 위해서는, 기준이 되는 월드 좌표계 원점이 필요하다. 여기에서는,base=0(으)로서 0 번째의 이미지("00.png")(을)를 지정하고 있어, 이 이미지 에 비치는 패턴의 3 차원 좌표계((2)그리고 주어진)에 있어서의 외부 파라미터가 추정된다. 이 때의 월드 좌표계의 원점은,(3)의 향으로 설명한 것처럼 결정된다. 이 샘플 프로그램에서는, 실행 결과예에 있는 이미지의 적색의 행의 최초의 코너가 원점이 된다.

// (7)XML 파일에의 써내

구할 수 있던, 내부 파라미터, 외부 파라미터, 일그러짐 계수를 파일에 써낸다. 여기에서는, 확장자(extension)에 의해 XML 형식을 지정해 있지만,YAML 형식에서 출력하는 것도 가능하다.

### 실행 결과예

카메라 calibration에 이용한 이미지의 일부(무순서)의 인식 결과. [실제의 calibration 결과 파일\(XML\)](#). 이 calibration 결과는,DFW-VL500(sony)에 광각렌즈VCL-00637S(sony)(을)를 단 것으로 갔다.



---

## 일그러짐 보정 cvUndistort2

calibration 데이터를 이용하고, 일그러짐을 보정한다

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
```

```

int
main (int argc, char *argv[])
{
    IplImage *src_img, *dst_img;
    CvMat *intrinsic, *distortion;
    CvFileStorage *fs;
    CvFileNode *param;

    // (1)보정 대상이 되는 이미지의 읽기
    if (argc < 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    // (2)파라미터 파일의 읽기
    fs = cvOpenFileStorage ("camera.xml", 0, CV_STORAGE_READ);
    param = cvGetFileNodeByName (fs, NULL, "intrinsic");
    intrinsic = (CvMat *) cvRead (fs, param);
    param = cvGetFileNodeByName (fs, NULL, "distortion");
    distortion = (CvMat *) cvRead (fs, param);
    cvReleaseFileStorage (&fs);

    // (3)일그러짐 보정
    cvUndistort2 (src_img, dst_img, intrinsic, distortion);

    // (4)이미지를 표시, 키가 밀렸을 때에 종료
    cvNamedWindow ("Distortion", CV_WINDOW_AUTOSIZE);
    cvShowImage ("Distortion", src_img);
    cvNamedWindow ("UnDistortion", CV_WINDOW_AUTOSIZE);
    cvShowImage ("UnDistortion", dst_img);
    cvWaitKey (0);

    cvDestroyWindow ("Distortion");
    cvDestroyWindow ("UnDistortion");
    cvReleaseImage (&src_img);
    cvReleaseImage (&dst_img);
    cvReleaseMat (&intrinsic);
    cvReleaseMat (&distortion);

    return 0;
}

```

// (1)보정 대상이 되는 이미지의 읽기  
 보정을 실시하는 이미지를 읽어들인다. 물론, 보정 대상이 되는 이미지를 촬영한 카메라는, calibration 가 끝난 상태이라고 한다.

// (2)파라미터 파일의 읽기

calibration 결과를 격납한 파라미터 파일을 읽어들인다. 일그러짐 보정에 필요한 파라미터는, 내부 파라미터, 및 일그러짐 계수 뿐이므로, 이것을 파일로부터 읽어들인다.

// (3)일그러짐 보정

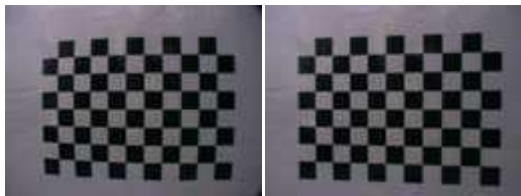
함수 cvUndisort2()에 의해, 입력 이미지의 일그러짐 보정을 행한다.

// (4)이미지를 표시, 키가 밀렸을 때에 종료

일그러짐 보정전 이미지와 비뚤어져 보정 후 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

### 실행 결과예

[좌]일그러짐 보정전. [우]일그러짐 보정 후.



---

## 맵을 이용한 일그러짐 보정 cvInitUndistortMap

calibration 데이터를 이용해 맵을 작성해, 일그러짐을 보정한다

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char *argv[])
{
    IplImage *src_img, *dst_img;
    CvMat *intrinsic, *distortion;
    CvFileStorage *fs;
    CvFileNode *param;
    IplImage *mapx = cvCreateImage (cvSize (640, 480), IPL_DEPTH_32F, 1);
    IplImage *mapy = cvCreateImage (cvSize (640, 480), IPL_DEPTH_32F, 1);

    if (argc < 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
        return -1;
    dst_img = cvCloneImage (src_img);

    fs = cvOpenFileStorage ("camera.xml", 0, CV_STORAGE_READ);
    param = cvGetFileNodeByName (fs, NULL, "intrinsic");
```

```

intrinsic = (CvMat *) cvRead (fs, param);
param = cvGetFileNodeByName (fs, NULL, "distortion");
distortion = (CvMat *) cvRead (fs, param);
cvReleaseFileStorage (&fs);

// (1)일그러짐 보정을 위한 맵 초기화
cvInitUndistortMap (intrinsic, distortion, mapx, mapy);
// (2)일그러짐 보정
cvRemap (src_img, dst_img, mapx, mapy);

cvNamedWindow ("Distortion", CV_WINDOW_AUTOSIZE);
cvShowImage ("Distortion", src_img);
cvNamedWindow ("UnDistortion", CV_WINDOW_AUTOSIZE);
cvShowImage ("UnDistortion", dst_img);
cvWaitKey (0);

cvDestroyWindow ("Distortion");
cvDestroyWindow ("UnDistortion");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);
cvReleaseImage (&mapx);
cvReleaseImage (&mapy);
cvReleaseMat (&intrinsic);
cvReleaseMat (&distortion);

return 0;
}

```

// (1)일그러짐 보정을 위한 맵 초기화

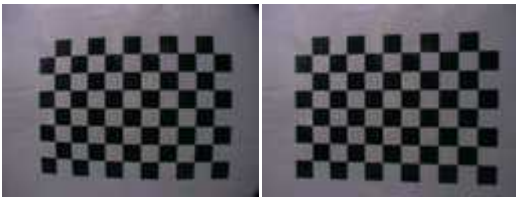
함수 cvInitUndistortMap()(을)를 이용하고, 일그러짐을 보정하기 위한 맵 mapx,mapy 의 초기화를 실시한다.

// (2)일그러짐 보정

작성된 맵을 이용하고, 이미지의 기하 변환(cvRemap)(을)를 실시한다. 함수 cvUndistort2()(을)를 이용하는 경우와 비교해서, 계산 속도가 향상한다. 그러나, 일반적인 기하 변환을 실시하는 함수를 이용하고 있기 때문에인가, 이전의 버전으로 실장되고 있던 일그러짐 보정 함수에 비해 퍼포먼스가 뒤떨어진다. 물론, 이전의 함수 cvUnDistortInit,cvUnDistort 도 cvcompat.h 안에서 선언되고 있지만, 장래적으로 폐지될 예정이므로 이용은 추천 되지 않는다 (또 코멘트에 의하면, 이러한 함수는,quite hackerish implementations 이다). 단순히 일그러짐 보정의 속도만을 요구한다면, Yahoo!Groups 의 투고([Speeding up undistort](#))에도 있도록(듯이), 스스로 LUT(을)를 작성하면 좋다. 그 경우는, 이미지의 보완을 별도 실시할 필요가 있다.

## 실행 결과예

[좌]일그러짐 보정전. [우]일그러짐 보정 후.



## 서포트 벡터 머신

OpenCV-1.0.0 그리고 제공된 SVM에 관한 함수는, libsvm 라이브러리(version 2.6)의 기능을 실장한 것이다. 또, 이 버전에서는, 학습 후의 파라미터를 보존, 읽는 함수 (save, load)에 버그가 존재해, Yahoo!Groups 그리고 보고되고 있는 패치 (<http://tech.groups.yahoo.com/group/OpenCV/message/48635>) 등을 수신자명 차면, 해당 기능을 이용할 수 없기 때문에 주의하는 것.

샘플

---

## 서포트 벡터 머신 CvSVM

SVM(을)를 이용해 2 차원 벡터의 3 클래스 분류 문제를 푼다

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <time.h>

int
main (int argc, char **argv)
{
    const int s = 1000;
    int size = 400;
    int i, j, sv_num;
    IplImage *img;
    CvSVM svm = CvSVM ();
    CvSVMParams param;
    CvTermCriteria criteria;
    CvRNG rng = cvRNG (time (NULL));
    CvPoint pts[s];
    float data[s * 2];
    int res[s];
    CvMat data_mat, res_mat;
    CvScalar rcolor;
    const float *support;

    // (1) 이미지 영역의 확보와 초기화
    img = cvCreateImage (cvSize (size, size), IPL_DEPTH_8U, 3);
```

```

cvZero (img);

// (2)학습 데이터의 생성
for (i = 0; i < s; i++) {
    pts[i].x = cvRandInt (&rng) % size;
    pts[i].y = cvRandInt (&rng) % size;
    if (pts[i].y > 50 * cos (pts[i].x * CV_PI / 100) + 200) {
        cvLine (img, cvPoint (pts[i].x - 2, pts[i].y - 2), cvPoint (pts[i].x + 2, pts[i].y
+ 2), CV_RGB (255, 0, 0));
        cvLine (img, cvPoint (pts[i].x + 2, pts[i].y - 2), cvPoint (pts[i].x - 2, pts[i].y
+ 2), CV_RGB (255, 0, 0));
        res[i] = 1;
    }
    else {
        if (pts[i].x > 200) {
            cvLine (img, cvPoint (pts[i].x - 2, pts[i].y - 2), cvPoint (pts[i].x + 2,
pts[i].y + 2), CV_RGB (0, 255, 0));
            cvLine (img, cvPoint (pts[i].x + 2, pts[i].y - 2), cvPoint (pts[i].x - 2,
pts[i].y + 2), CV_RGB (0, 255, 0));
            res[i] = 2;
        }
        else {
            cvLine (img, cvPoint (pts[i].x - 2, pts[i].y - 2), cvPoint (pts[i].x + 2,
pts[i].y + 2), CV_RGB (0, 0, 255));
            cvLine (img, cvPoint (pts[i].x + 2, pts[i].y - 2), cvPoint (pts[i].x - 2,
pts[i].y + 2), CV_RGB (0, 0, 255));
            res[i] = 3;
        }
    }
}

// (3)학습 데이터의 표시
cvNamedWindow ("SVM", CV_WINDOW_AUTOSIZE);
cvShowImage ("SVM", img);
cvWaitKey (0);

// (4)학습 파라미터의 생성
for (i = 0; i < s; i++) {
    data[i * 2] = float (pts[i].x) / size;
    data[i * 2 + 1] = float (pts[i].y) / size;
}
cvInitMatHeader (&data_mat, s, 2, CV_32FC1, data);
cvInitMatHeader (&res_mat, s, 1, CV_32SC1, res);
criteria = cvTermCriteria (CV_TERMCRIT_EPS, 1000, FLT_EPSILON);
param = CvSVMParams (CvSVM::C_SVC, CvSVM::RBF, 10.0, 8.0, 1.0, 10.0, 0.5, 0.1, NULL,
criteria);

```

```
// (5)SVM 의 학습
```

```
svm.train (&data_mat, &res_mat, NULL, NULL, param);
```

```
// (6)학습 결과의 그리기
```

```
for (i = 0; i < size; i++) {  
    for (j = 0; j < size; j++) {  
        CvMat m;  
        float ret = 0.0;  
        float a[] = { float (j) / size, float (i) / size };  
        cvInitMatHeader (&m, 1, 2, CV_32FC1, a);  
        ret = svm.predict (&m);  
        switch ((int) ret) {  
            case 1:  
                rcolor = CV_RGB (100, 0, 0);  
                break;  
            case 2:  
                rcolor = CV_RGB (0, 100, 0);  
                break;  
            case 3:  
                rcolor = CV_RGB (0, 0, 100);  
                break;  
        }  
        cvSet2D (img, i, j, rcolor);  
    }  
}
```

```
// (7)트레이닝 데이터의 재그리기
```

```
for (i = 0; i < s; i++) {  
    CvScalar rcolor;  
    switch (res[i]) {  
        case 1:  
            rcolor = CV_RGB (255, 0, 0);  
            break;  
        case 2:  
            rcolor = CV_RGB (0, 255, 0);  
            break;  
        case 3:  
            rcolor = CV_RGB (0, 0, 255);  
            break;  
    }  
    cvLine (img, cvPoint (pts[i].x - 2, pts[i].y - 2), cvPoint (pts[i].x + 2, pts[i].y +  
2), rcolor);  
    cvLine (img, cvPoint (pts[i].x + 2, pts[i].y - 2), cvPoint (pts[i].x - 2, pts[i].y +  
2), rcolor);  
}
```



```

// (8)서포트 벡터의 그리기
sv_num = svm.get_support_vector_count ();
for (i = 0; i < sv_num; i++) {
    support = svm.get_support_vector (i);
    cvCircle (img, cvPoint ((int) (support[0] * size), (int) (support[1] * size)), 5,
CV_RGB (200, 200, 200));
}

// (9)이미지의 표시
cvNamedWindow ("SVM", CV_WINDOW_AUTOSIZE);
cvShowImage ("SVM", img);
cvWaitKey (0);

cvDestroyWindow ("SVM");
cvReleaseImage (&img);

return 0;
}

```

// (1)이미지 영역의 확보와 초기화  
이미지 영역을 확보해, 제로 클리어(흑색으로 초기화) 한다.

// (2)트레이닝 데이터의 생성  
2 차원의 트레이닝 데이터를 랜덤에 생성해, 그 값을 CvPoint 형태의 배열 pts[]에 격납한다. 또, 각 트레이닝 데이터의 클래스를 반응을 일으키는 최소의 물리량에 의해서 결정해, 배열 res[]에 격납한다.

// (3)트레이닝 데이터의 표시  
생성된 트레이닝 데이터를, 최초로 확보한 이미지상에 그리기 해, 표시한다. 클래스 1-3 까지가, 적, 록, 청의 각 색으로 나타난다. 여기서, 무엇인가 키가 밀릴 때까지 기다린다.

// (4)학습 파라미터의 생성  
SVM 의 학습 파라미터를 이하와 같이 결정한다.

svm 의 종류 : CvSVM::C\_SVC  
커널의 종류 : CvSVM::RBF  
degree : 10.0(이번은, 이용되지 않는다)  
gamma : 8.0  
coef0 : 1.0(이번은, 이용되지 않는다)  
C : 10.0  
nu : 0.5(이번은, 이용되지 않는다)  
p : 0.1(이번은, 이용되지 않는다)  
또, 트레이닝 데이터를 정규화해,CvMat형태의 행렬에 격납한다.

// (5)SVM 의 학습  
트레이닝 데이터와 결정할 수 있던 학습 파라미터를 이용하고,SVM 의 학습을 행한다.

// (6)학습 결과의 그리기

학습 결과를 나타내기 위해서, 이미지 영역내의 모든 픽셀(특징 벡터)을 입력으로서 클래스 분류를 실시한다. 또, 입력 픽셀을, 그것이 속하는 클래스에 대응한 색으로 그리기 한다.

// (7)트레이닝 데이터의 재그리기

트레이닝 데이터를, 결과 이미지 우에에 겹쳐 그리기 한다.

// (8)서포트 벡터의 그리기

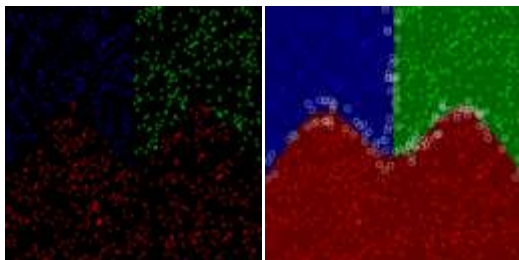
트레이닝 데이터 가운데, 서포트 벡터를 흰 동그라미로 둘러싸 표시한다.

// (9)이미지의 표시

실제로, 처리 결과의 이미지를 표시해, 무엇인가 키가 밀리면 종료한다.

### 실행 결과예

[좌]학습 샘플,[우]분류 결과



---

## 이미지의 각 픽셀치를 특징 벡터로 했다 SVM 의 학습

학습용의 이미지를 읽어들여, 그 픽셀치를 특징 벡터로서 SVM 의 학습을 실시한다

### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    int i, j, ii, jj;
    int width = 28, height = 30; /* 샘플 이미지 사이즈 */
    int image_dim = width * height;
    int pimage_num = 500; /* 포지티브 샘플수 */
    int nimage_num = 1000; /* 네가티브 샘플수 */
    int all_image_num = pimage_num + nimage_num;
    IplImage *img_org;
```

```

IplImage *sample_img;
int res[all_image_num];
float data[all_image_num * image_dim];
CvMat data_mat, res_mat;
CvTermCriteria criteria;
CvSVM svm = CvSVM ();
CvSVMParams param;
char filename[64];

// (1)포지티브 샘플의 읽기
for (i = 0; i < pimage_num; i++) {
    sprintf (filename, "positive/%03d.png", i);
    img_org = cvLoadImage (filename, CV_LOAD_IMAGE_GRAYSCALE);
    sample_img = cvCreateImage (cvSize (width, height), IPL_DEPTH_8U, 1);
    cvResize (img_org, sample_img);
    cvSmooth (sample_img, sample_img, CV_GAUSSIAN, 3, 0, 0, 0);
    for (ii = 0; ii < height; ii++) {
        for (jj = 0; jj < width; jj++) {
            data[i * image_dim + (ii * width) + jj] =
                float ((int) ((uchar) (sample_img->imageData[ii * sample_img->widthStep + jj]))
/ 255.0);
        }
    }
    res[i] = 1;
}

// (2)네가티브 샘플의 읽기
j = i;
for (i = j; i < j + nimage_num; i++) {
    sprintf (filename, "negative/%03d.jpg", i - j);
    img_org = cvLoadImage (filename, CV_LOAD_IMAGE_GRAYSCALE);
    sample_img = cvCreateImage (cvSize (width, height), IPL_DEPTH_8U, 1);
    cvResize (img_org, sample_img);
    cvSmooth (sample_img, sample_img, CV_GAUSSIAN, 3, 0, 0, 0);
    for (ii = 0; ii < height; ii++) {
        for (jj = 0; jj < width; jj++) {
            data[i * image_dim + (ii * width) + jj] =
                float ((int) ((uchar) (sample_img->imageData[ii * sample_img->widthStep + jj]))
/ 255.0);
        }
    }
    res[i] = 0;
}

// (3)SVM 학습 데이터와 파라미터의 초기화
cvInitMatHeader (&data_mat, all_image_num, image_dim, CV_32FC1, data);

```

```

cvInitMatHeader (&res_mat, all_image_num, 1, CV_32SC1, res);
criteria = cvTermCriteria (CV_TERMCRIT_EPS, 1000, FLT_EPSILON);
param = CvSVMParams (CvSVM::C_SVC, CvSVM::RBF, 10.0, 0.09, 1.0, 10.0, 0.5, 1.0, NULL,
criteria);

// (4)SVM 의 학습과 데이터의 보존
svm.train (&data_mat, &res_mat, NULL, NULL, param);
svm.save ("svm_image.xml");

cvReleaseImage (&img_org);
cvReleaseImage (&sample_img);

return 0;
}

```

#### // (1)포지티브 샘플의 읽기

포지티브 샘플이 되는 이미지군을 읽어들이며, 각 픽셀의 값을 float 형태의 배열로 변환한다. 여기에서는 간단을 위해서, 포지티브 샘플("positive/디렉토리 이하에 있다")(은)는, 3 자리수 연번의 파일명의 이미지로서 미리 준비되어 있다고 한다. 우선, 읽어들이는 각 이미지를 동일한 사이즈(여기에서는, 28×30)에 리사이즈 해, 노이즈의 영향을 경감하기 위해서 스무딩을 실시한다. 다음에, 그 이미지의 각 픽셀의 휘도치(여기서, 이미지는 그레이 스케일 이미지로서 읽히고 있다)를 특징 벡터로서 이용하기 위해서, 배열로 변환한다. 즉, 1 개의 이미지에 대한 특징 벡터는, 이미지의 높이×이미지의 폭, 이 되어, 그것이 샘플 이미지의 매수분만큼 준비된다. 판별치로서 "1"(을)를 이용하고 있다. 또, 포지티브 샘플로서 500 매의 얼굴 이미지(거의 정면, 기울기 없음)을 이용하고 있다.

OpenCV 에는, haar-like 특징을 이용한 물체 검출 수법이 실장되고 있어 그 쪽을 이용한 얼굴 검출이 정도도 처리 속도도 좋기 때문에, 얼굴 이미지를 검출하는 의미는 별로 없지만, 샘플의 입수의 하기 쉬움이라고 알기 쉬움으로부터 이번은 얼굴 이미지를 이용한 학습을 실시했다.

#### // (2)네가티브 샘플의 읽기

네가티브 샘플이 되는 이미지군을 읽어들이며, 포지티브 샘플과 같게 배열로 변환한다. 판별치로서 "0"(을)를 이용하고 있다. 또, 네가티브 샘플로서 1000 매의 임의의 이미지(얼굴 이외의 이미지)을 이용하고 있다.

#### // (3)SVM 학습 데이터와 파라미터의 초기화

샘플 이미지의 픽셀치의 배열과 판별치의 배열을, 행렬로 변환한다. 또, SVM 의 학습을 위한 파라미터를 초기화한다. 여기에서는, 꽤 적당하게 파라미터를 지정해 있으므로, 필요에 따라서 적절한 파라미터를 설정할 필요가 있다.

#### // (4)SVM 의 학습과 데이터의 보존

포지티브, 네가티브 샘플의 픽셀치, 및 지정된 파라미터를 이용하고, svm.train()메소드에 의해 SVM 의 학습을 실시한다. 샘플수는, 포지티브 500, 네가티브 1000, 특징 벡터는, 28×30=840 차원이다. 또, 학습되었다 SVM 의 파라미터를, svm.save()메소드에 의해 XML 형식의 파일로서 보존한다. 이 페이지의 최초로도 말한 것처럼, save 및 load 의 기능을 이용하기 위해서는, OpenCV 의 소스를 수정할 필요가 있다.

## 실행 결과예

[svm\\_image\\_xml.zip](#) 하지만, 실제로 보존되는 파라미터의 일례이다. 여기에서는, XML 형식을 이용했지만, YAML 형식에서 보존하는 것도 가능하다.



---

## 이미지의 각 픽셀치를 특징 벡터로 했다 SVM 에 의한 물체 검출

학습되었다 SVM 파라미터를 읽어들이어, 주어진 이미지중으로부터 물체를 검출한다

## 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    int i, j;
    int width = 28, height = 30; /* 샘플 이미지 사이즈 */
    int image_dim = width * height;
    CvMat m;
    float a[image_dim];
    float ret = -1.0;
    float scale;
    IplImage *src, *src_color, *src_tmp;
    int sx, sy, tw, th;
    int stepx = 3, stepy = 3;
    double steps = 1.2;
    int iterate;
    CvSVM svm = CvSVM ();

    // (1)이미지의 읽기
    if (argc < 2 ||
        (src = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE)) == 0 ||
        (src_color = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0) {
        return -1;
    }
}
```

```

// (2)SVM 데이터의 읽기
svm.load ("svm_image.xml");

/* 읽어들이는 이미지를 부분 이미지마다 처리 */
cvInitMatHeader (&m, 1, image_dim, CV_32FC1, NULL);
tw = src->width;
th = src->height;

for (iterate = 0; iterate < 1; iterate++) {
    // (3)이미지를 축소해, 현재의 부분 이미지를 행렬에 변경
    src_tmp = cvCreateImage (cvSize ((int) (tw / steps), (int) (th / steps)),
IPL_DEPTH_8U, 1);
    cvResize (src, src_tmp);
    tw = src_tmp->width;
    th = src_tmp->height;
    for (sy = 0; sy <= src_tmp->height - height; sy += stepy) {
        for (sx = 0; sx <= src_tmp->width - width; sx += stepx) {
            for (i = 0; i < height; i++) {
                for (j = 0; j < width; j++) {
                    a[i * width + j] =
                        float ((int) ((uchar) (src_tmp->imageData[(i + sy) * src_tmp->widthStep +
(j + sx)])) / 255.0);
                }
            }
            cvSetData (&m, a, sizeof (float) * image_dim);

            // (4)SVM 에 의한 판정과 결과의 그리기
            ret = svm.predict (&m);
            if ((int) ret == 1) {
                scale = (float) src->width / tw;
                cvRectangle (src_color, cvPoint ((int) (sx * scale), (int) (sy * scale)),
                    cvPoint ((int) ((sx + width) * scale), (int) ((sy + height) *
scale)), CV_RGB (255, 0, 0), 2);
            }
        }
    }
    cvReleaseImage (&src_tmp);
}

// (5)검출 결과 이미지의 표시
cvNamedWindow ("svm_predict", CV_WINDOW_AUTOSIZE);
cvShowImage ("svm_predict", src_color);
cvWaitKey (0);

cvDestroyWindow ("svm_predict");

```

```

cvReleaseImage (&src);
cvReleaseImage (&src_color);
cvReleaseImage (&src_tmp);

return 0;
}

```

#### // (1)이미지의 읽기

SVM 에 의한 판별의 대상이 되는 이미지를, 커멘드 인수로 지정된 파일로부터 읽어들인다. 처리는 그레이 스케일 이미지에 대해서 행해지지만, 마지막 결과 표시를 위해서 칼라 이미지도 별도 준비해 둔다.

#### // (2)SVM 데이터의 읽기

미리 학습되었다 SVM 의 파라미터를,svm.load()메소드에 의해 파일(여기에서는,"xvm\_image.xml")(으)로부터 읽어들인다. 이 페이지의 최초로도 말한 것처럼,save 및 load 의 기능을 이용하기 위해서는,OpenCV 의 소스를 수정할 필요가 있다.

#### // (3)이미지를 축소해, 현재의 부분 이미지를 행렬에 변경

읽힌 이미지를 부분 이미지마다 처리하기 위해서,stepx=3, stepy=3 픽셀 마다,width×height=28×30 의 사이즈의 부분 이미지의 픽셀치를 배열에 대입한다. 또,SVM(을)를 적용하기 위해서, 함수 cvSetData()에 의해, 그 배열을 1×(28×30)의 행렬의 데이터로서 세트 한다.

#### // (4)SVM 에 의한 판정과 결과의 그리기

svm.predict()메소드에 의해, 행렬에 변환된 부분 이미지가, 어느 클래스에 속하는지를 판별한다. 여기에서는, 전술의 학습 파라미터 파일을 이용하고 있으므로, 처리 대상이 되는 부분 이미지가 그 학습된 물체(얼굴)인지 아닌지를 판별하고 있다. 얼굴이라면 판별된 부분 이미지는, 붉은 구형으로 표시된다. 파라미터의 최적화나, 특별한 처리상의 궁리(처리 영역의 선정이나 특징 벡터의 추출등)를 하고 있는 것도 없기 때문에, 검출 정도는 높지 않다. 게다가 이미지 전체에 대해서 처리를 행하는 경우는, 특징 벡터가 큰 일도 있어, 상당한 처리 시간이 필요하다. 또, 이번은,iterate 변수에 의한 루프를 1 번 밖에 가서 않지만, 이미지중으로부터 다른 크기의 물체를 검출하고 싶은 경우에는, 원이미지를 수시 축소(여기에서는,1/steps=1/1.2) 해 같은 처리를 실시한다.

#### // (5)검출 결과 이미지의 표시

검출 결과적으로 붉은 구형이 그리기 된 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

#### 실행 결과예



#### 위젯(컨트롤)

OpenCV-1.0.0그리고 이용할 수 있는 위젯(컨트롤)는, 이미지를 표시하기 위한 윈도우, 및 트랙 바뿐이다. 복잡한G

UI디자인이나 이벤트를 적절히 읽어 날린다고 하는 처리가 어렵고, 카메라 캡처냄새나도 해상도의 변경이나 frame rate의 설정 등에 난점 하지만 있으므로, 간단한 테스트 프로그램 이외에서는, 다른 라이브러리를 이용하는 편이 무난하다.

샘플

## 트랙 바의 이용 cvCreateTrackbar, cvGetTrackbarPos, cvSetTrackbarPos

트랙 바의 작성, 그 위치의 취득과 설정

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

/* 글로벌 변수 */
CvFont font;
IplImage *img = 0;

/* 프로토 타입 선언 */
void on_trackbar1 (int val);
void on_trackbar2 (int val);

int
main (int argc, char *argv[])
{
    // (1)이미지 영역을 확보해, 초기화한다
    img = cvCreateImage (cvSize (400, 200), IPL_DEPTH_8U, 3);
    cvZero (img);
    cvInitFont (&font, CV_FONT_HERSHEY_DUPLEX, 1.0, 1.0);

    // (2)윈도우, 및 트랙 바를 작성한다
    cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
    cvCreateTrackbar ("Trackbar1", "Image", 0, 100, on_trackbar1);
    cvCreateTrackbar ("Trackbar2", "Image", 0, 100, on_trackbar2);
    cvShowImage ("Image", img);
    cvWaitKey (0);

    cvDestroyWindow ("Image");
    cvReleaseImage (&img);

    return 0;
}
```



```

/* 콜백 함수 */
void
on_trackbar1 (int val)
{
    char str[64];

    // (3)트랙 바 2(을)를, 트랙 바 1 에 동기 시킨다
    cvSetTrackbarPos ("Trackbar2", "Image", val);

    // (4)트랙 바 1 의 값을 그리기 한다
    cvRectangle (img, cvPoint (0, 0), cvPoint (400, 50), cvScalar (0), CV_FILLED);
    snprintf (str, 64, "%d", val);
    cvPutText (img, str, cvPoint (15, 30), &font, CV_RGB (0, 200, 100));
    cvShowImage ("Image", img);
}

void
on_trackbar2 (int val)
{
    char str[64];
    int pos1, pos2;

    // (5)트랙 바 2 의 이동 범위를, 트랙 바 1 의 값±20(으)로 한정한다
    pos1 = cvGetTrackbarPos ("Trackbar1", "Image");
    if (pos1 > val)
        pos2 = pos1 - 20 < val ? val : pos1 - 20;
    else
        pos2 = pos1 + 20 > val ? val : pos1 + 20;
    cvSetTrackbarPos ("Trackbar2", "Image", pos2);

    // (6)트랙 바 2 의 값을 그리기 한다
    cvRectangle (img, cvPoint (0, 50), cvPoint (400, 100), cvScalar (0), CV_FILLED);
    snprintf (str, 64, "%d", val);
    cvPutText (img, str, cvPoint (15, 80), &font, CV_RGB (200, 200, 0));
    cvShowImage ("Image", img);
}

```

// (1)이미지 영역을 확보해, 초기화한다

이미지 영역을 확보해, 초기화한다. 또, 후에 결과의 문자열을 그리기하기 위한 폰트 구조체를 초기화한다.

// (2)윈도우, 및 트랙 바를 작성한다

윈도우를 작성 후, 함수 cvCreateTrackbar()에 의해, 윈도우상에 트랙 바를 추가한다. 트랙 바 작성시에, 트랙 바 이벤트에 대한 콜백 함수(on\_trackbar1,on\_trackbar2)(을)를 설정한다.

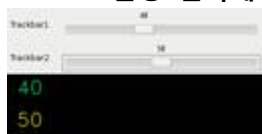
// (3)트랙 바 2(을)를, 트랙 바 1 에 동기 시킨다  
함수 cvSetTrackbarPos()(을)를 이용하고, 트랙 바 2 의 값을, 트랙 바 1 의 현재의 값과 동기 시킨다.

// (4)트랙 바 1 의 값을 그리기 한다  
트랙 바 1 의 현재의 값을, 이미지 영역 상부에 그리기 한다.

// (5)트랙 바 2 의 이동 범위를, 트랙 바 1 의 값 $\pm 20$ (으)로 한정한다  
함수 cvGetTrackbarPos()(을)를 이용하고, 트랙 바 1 의 현재의 값을 취득해, 그 값의 $\pm 20$  의 범위내에서 트랙 바 2 하지만 움직이도록(듯이) 설정한다.

// (6)트랙 바 2 의 값을 그리기 한다  
트랙 바 2 의 현재의 값을, 이미지 영역 하부에 그리기 한다.

### 실행 결과예



### ■ 이벤트

OpenCV그림, 윈도우상에서 발생한 마우스 이벤트, 및 윈도우가 액티브한 상태로 밀린 키코드를 취득할 수 있다.  
샘플

---

## 마우스 이벤트의 취득 cvSetMouseCallback

마우스 이벤트를 취득하고, 윈도우에 표시한다

### 샘플 코드

#### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

/* 글로벌 변수 */
CvFont font;
IplImage *img = 0;

/* 프로토 타입 선언 */
void on_mouse (int event, int x, int y, int flags, void *param);

int
main (int argc, char *argv[])
{
    int c;
```

```

// (1)이미지 영역을 확보해, 초기화한다
img = cvCreateImage (cvSize (640, 480), IPL_DEPTH_8U, 3);
cvZero (img);
cvInitFont (&font, CV_FONT_HERSHEY_DUPLEX, 0.4, 0.4);

// (2)윈도우를 작성해, 마우스 이벤트에 대한 콜백 함수를 등록한다
cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
cvSetMouseCallback ("Image", on_mouse);
cvShowImage ("Image", img);

// (3)'Esc'키가 밀렸을 경우에 종료한다
while (1) {
    c = cvWaitKey (0);
    if (c == 'Wx1b')
        return 1;
}

cvDestroyWindow ("Image");
cvReleaseImage (&img);

return 0;
}

/* 콜백 함수 */
void
on_mouse (int event, int x, int y, int flags, void *param = NULL)
{
    char str[64];
    static int line = 0;
    const int max_line = 15, w = 15, h = 30;

    // (4)마우스 이벤트를 취득
    switch (event) {
    case CV_EVENT_MOUSEMOVE:
        snprintf (str, 64, "(%d,%d) %s", x, y, "MOUSE_MOVE");
        break;
    case CV_EVENT_LBUTTONDOWN:
        snprintf (str, 64, "(%d,%d) %s", x, y, "LBUTTON_DOWN");
        break;
    case CV_EVENT_RBUTTONDOWN:
        snprintf (str, 64, "(%d,%d) %s", x, y, "RBUTTON_DOWN");
        break;
    case CV_EVENT_MBUTTONDOWN:
        snprintf (str, 64, "(%d,%d) %s", x, y, "MBUTTON_DOWN");
        break;
    }
}

```

```

case CV_EVENT_LBUTTONDOWN:
    snprintf (str, 64, "(%d,%d) %s", x, y, "LBUTTON_UP");
    break;
case CV_EVENT_RBUTTONDOWN:
    snprintf (str, 64, "(%d,%d) %s", x, y, "RBUTTON_UP");
    break;
case CV_EVENT_MBUTTONDOWN:
    snprintf (str, 64, "(%d,%d) %s", x, y, "MBUTTON_UP");
    break;
case CV_EVENT_LBUTTONDOWNDBLCLK:
    snprintf (str, 64, "(%d,%d) %s", x, y, "LBUTTON_DOUBLE_CLICK");
    break;
case CV_EVENT_RBUTTONDOWNDBLCLK:
    snprintf (str, 64, "(%d,%d) %s", x, y, "RBUTTON_DOUBLE_CLICK");
    break;
case CV_EVENT_MBUTTONDOWNDBLCLK:
    snprintf (str, 64, "(%d,%d) %s", x, y, "MBUTTON_DOUBLE_CLICK");
    break;
}

// (5)mouse button, 수식 키를 취득
if (flags & CV_EVENT_FLAG_LBUTTON)
    strncat (str, " + LBUTTON", 64);
if (flags & CV_EVENT_FLAG_RBUTTON)
    strncat (str, " + RBUTTON", 64);
if (flags & CV_EVENT_FLAG_MBUTTON)
    strncat (str, " + MBUTTON", 64);
if (flags & CV_EVENT_FLAG_CTRLKEY)
    strncat (str, " + CTRL", 64);
if (flags & CV_EVENT_FLAG_SHIFTKEY)
    strncat (str, " + SHIFT", 64);
if (flags & CV_EVENT_FLAG_ALTKEY)
    strncat (str, " + ALT", 64);

// (6)마우스 좌표, 이벤트, 수식 키등을 이미지에 그리기, 표시
if (line > max_line) {
    cvGetRectSubPix (img, img, cvPoint2D32f (320 - 0.5, 240 - 0.5 + h));
    cvPutText (img, str, cvPoint (w, 20 + h * max_line), &font, CV_RGB (0, 200, 100));
}
else {
    cvPutText (img, str, cvPoint (w, 20 + h * line), &font, CV_RGB (0, 200, 100));
}
line++;
cvShowImage ("Image", img);
}

```

이미지 영역을 확보해, 초기화한다. 또, 후에 결과의 문자열을 그리기하기 위한 폰트 구조체를 초기화한다.

초기화한 이미지를 표시해, 함수 `cvSetMouseCallback()`(을)를 이용하고, 마우스 이벤트에 대한 콜백 함수(`on mouse`)(을)를 등록한다.

"Esc"키가 밀렸을 경우에 종료한다.'Wx1b'(은)는,"Esc"키를 나타내지만, 이 문자 코드는"27"그래서,

(이)라고 해도 좋다. 또, 그 외의 키의 경우는, 이하와 같이 지정하면 좋다.

마우스 이벤트를 취득한다.취득할 수 있는 마우스 이벤트는, 마우스의 이동과 좌, 오른쪽, 중앙 버튼의 DOWN,UP, 더블 클릭이다. 덧붙여서, 더블 클릭을 실시했을 때의 마우스 이벤트는,

이벤트시에 밀리고 있는 mouse button, 수식 키(CTRL,ALT,SHIFT)(을)를 취득한다. 이러한 플래그는, 각 이벤트 플래그의 논리합으로 표현된다. 또, mouse button 플래그는, 버튼 DOWN 시간에는 발생하지 않지만, 버튼 UP 시간에는 발생한다.

현재의 마우스 좌표, 마우스 이벤트, 수식 키 모두를 이미지에 그리기 해, 표시한다. 또, 함수 `cvGetRectSubPix()` (을)를 이용하고, 화면을 스크롤 시킨다.

[illegible]

OpenCV그림, 실제의 카메라나 동이미지 파일로부터 캡처를 실시하거나 반대로 이미지열을 동이미지 파일로서 써내거나 하는 것이 가능하다.

## 341

```

#include <cv.h>
#include <highgui.h>
#include <ctype.h>

int
main (int argc, char **argv)
{
    CvCapture *capture = 0;
    IplImage *frame = 0;
    double w = 320, h = 240;
    int c;

    // (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다
    if (argc == 1 || (argc == 2 && strlen (argv[1]) == 1 && isdigit (argv[1][0])))
        capture = cvCreateCameraCapture (argc == 2 ? argv[1][0] - '0' : 0);

    /* 이 설정은, 이용하는 카메라에 의존한다 */
    // (2)캡처사이즈를 설정한다.
    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_WIDTH, w);
    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_HEIGHT, h);

    cvNamedWindow ("Capture", CV_WINDOW_AUTOSIZE);

    // (3)카메라로부터 이미지를 캡처 한다
    while (1) {
        frame = cvQueryFrame (capture);
        cvShowImage ("Capture", frame);
        c = cvWaitKey (10);
        if (c == 'Wx1b')
            break;
    }

    cvReleaseCapture (&capture);
    cvDestroyWindow ("Capture");

    return 0;
}

```

// (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다  
 커멘드 인수로서 주어진 번호의 카메라에 대한, 캡처 구조체 CvCapture(을)를 작성한다. 인수가 지정되지 않는 경우는, 디폴트의 값이다"0"하지만 이용된다.

// (2)캡처사이즈를 설정한다

함수 cvSetCaptureProperty()에 의해, 캡처를 행할 때의 화면 사이즈(폭과 높이)를 지정한다. 다만, 실제의 카메라가 서포트하고 있지 않는 캡처사이즈는 지정할 수 없다.

// (3)카메라로부터 이미지를 캡처 한다

루프 블록중에 함수 cvQueryFrame() (을)를 호출하는 것으로, 실제로 캡처를 실시해, 그것을 표시한다. 캡처 안에 "Esc"키가 밀렸을 경우는, 종료한다.

#### 실행 결과예



#### CLOSE

카메라, 비디오 파일

OpenCV그림, 실제의 카메라나 동이미지 파일로부터 캡처를 실시하거나 반대로 이미지열을 동이미지 파일로서 써내거나 하는 것이 가능하다.

---

## 카메라로부터의 이미지 캡처 cvCreateCameraCapture, cvQueryFrame

지정된 번호의 카메라로부터 이미지를 캡처 해 표시한다

#### 샘플 코드

##### 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ctype.h>

int
main (int argc, char **argv)
{
    CvCapture *capture = 0;
    IplImage *frame = 0;
    double w = 320, h = 240;
    int c;

    // (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다
    if (argc == 1 || (argc == 2 && strlen (argv[1]) == 1 && isdigit (argv[1][0])))
        capture = cvCreateCameraCapture (argc == 2 ? argv[1][0] - '0' : 0);

    /* 이 설정은, 이용하는 카메라에 의존한다 */
    // (2)캡처사이즈를 설정한다.
```

```

cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_WIDTH, w);
cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_HEIGHT, h);

cvNamedWindow ("Capture", CV_WINDOW_AUTOSIZE);

// (3)카메라로부터 이미지를 캡처 한다
while (1) {
    frame = cvQueryFrame (capture);
    cvShowImage ("Capture", frame);
    c = cvWaitKey (10);
    if (c == 'Wx1b')
        break;
}

cvReleaseCapture (&capture);
cvDestroyWindow ("Capture");

return 0;
}

```

// (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다  
 함수 cvCreateCameraCapture()(을)를 이용하고, 커멘드 인수로서 주어진 번호의 카메라에 대한  
 캡처 구조체 CvCapture(을)를 작성한다. 인수가 지정되지 않는 경우는, 디폴트의  
 값이다"0"하지만 이용된다. OpenCV-1.0.0 의 샘플  
 프로그램에서는,cvCreateCameraCapture 대신에, cvCaptureFromCAM 하지만 이용되고 있지만,  
 실제로는 highgui.h 안에서,

```
#define cvCaptureFromCAM cvCreateCameraCapture
```

(와)과 같이 정의되고 있어 이러한 함수는 등가이다. 여기에서는, 메뉴얼에 따라 함수 cvCreateCameraCapture (을)를  
 이용한다.

// (2)캡처체사이즈를 설정한다

함수 cvSetCaptureProperty()에 의해, 캡처를 행할 때의 화면 사이즈(폭과 높이)를 지정한다.  
 다만, 실제의 카메라가 서포트하고 있지 않는 캡처체사이즈는 지정할 수 없다.

// (3)카메라로부터 이미지를 캡처 한다

루프 블록중에 함수 cvQueryFrame() (을)를 호출하는 것으로, 실제로 캡처를 실시해, 그것을  
 표시한다. 캡처 안에"Esc"키가 밀렸을 경우는, 종료한다.

#### 실행 결과예



CLOSE



동영상으로서 파일에 써낸다 cvCreateVideoWriter, cvWriteFrame, cvReleaseVideoWriter

카메라로부터 캡처 한 프레임을, 차례차례 파일에 써낸다

#### 샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <ctype.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
    CvCapture *capture = 0;
    IplImage *frame = 0;
    CvVideoWriter *vw;
    double w = 320, h = 240;
    int c, num = 0;
    CvFont font;
    char str[64];

    // (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다
    if (argc == 1 || (argc == 2 && strlen (argv[1]) == 1 && isdigit (argv[1][0])))
        capture = cvCaptureFromCAM (argc == 2 ? argv[1][0] - '0' : 0);

    // (2)캡처사이즈를 설정한다
    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_WIDTH, w);
    cvSetCaptureProperty (capture, CV_CAP_PROP_FRAME_HEIGHT, h);

    // (3)비디오 라이터 구조체를 작성한다
    vw = cvCreateVideoWriter ("cap.avi", CV_FOURCC ('X', 'V', 'I', 'D'), 15, cvSize ((int)
w, (int) h));

    cvInitFont (&font, CV_FONT_HERSHEY_COMPLEX, 0.7, 0.7);
    cvNamedWindow ("Capture", CV_WINDOW_AUTOSIZE);

    // (4)카메라로부터 이미지를 캡처 해, 파일에 써낸다
    while (1) {
        frame = cvQueryFrame (capture);
        sprintf (str, 64, "%03d[frame]", num);
```

```

cvPutText (frame, str, cvPoint (10, 20), &font, CV_RGB (0, 255, 100));
cvWriteFrame (vw, frame);
cvShowImage ("Capture", frame);
num++;
c = cvWaitKey (10);
if (c == 'Wx1b')
    break;
}
// (5)기입을 종료해, 구조체를 해방한다
cvReleaseVideoWriter (&vw);
cvReleaseCapture (&capture);
cvDestroyWindow ("Capture");

return 0;
}

```

// (1)커멘드 인수에 의해서 지정된 번호의 카메라에 대한 캡처 구조체를 작성한다  
전술의 캡처 샘플과 같게, 구조체를 작성한다.

// (2)캡처체사이즈를 설정한다  
전술의 캡처 샘플과 같게, 캡처체사이즈를 설정한다.

// (3)비디오 라이터 구조체를 작성한다  
파일에 써내기 위한 비디오 라이터 구조체를 작성한다.인수에는, 순서에, (파일명, 코덱 지정자, frame rate, 사이즈)(을)를 지정한다. 여기에서는, 코덱의 지정에, CV\_FOURCC('X','V','I','D')(XVID)(을)를 이용하고 있지만, 레퍼런스 메뉴얼에 있도록(듯이), CV\_FOURCC('M','J','P','G')(모션 JPEG)(이)나 CV\_FOURCC('P','I','M','1')(MPEG-1)(을)를 이용할 수도 있다. 비압축으로의 보존을 희망하는 경우는, CV\_FOURCC('D','I','B','I')(을)를 지정하면 좋다. 물론, 이용되는 코덱이 인스톨 되고 있을 필요가 있다. 또, MacOSX 그럼, 파일명을 풀 패스로 지정하든가, 혹은, touch 커멘드등에서 미리 파일 작성해 둘 필요가 있는 것 같다.

// (4)카메라로부터 이미지를 캡처 해, 파일에 써낸다  
카메라로부터 이미지를 캡처 해, 그 이미지에 대해서 문자열을 쓴 후에, 함수 cvWriteFrame()에 의해서 차례차례 파일에 써낸다.

// (5)기입을 종료해, 구조체를 해방한다  
"Esc"키로 루프를 빠진 후에는, 함수 cvReleaseVideoWriter()에 의해서, 기입을 종료시켜, 구조체를 해방한다.

#### 실행 결과예



## 라벨링

OpenCV 자체에는(OpenCV-1.0.0 시점에 있고) 라벨링을 실시하는 함수는 존재하지 않지만, OpenCV Library Wiki 그리고 소개되고 있다 ["Blob extraction library"](#)(을)를 이용하고, 이미지의 라벨링을 실시할 수 있다. 이 라이브러리는, MIL (Matrox Imaging Library)에 있어서의, MIL blob detection 모듈의 기능을 대체하는 것을, OpenCV(을)를 이용해 실장되어 있다. Win32 환경에서 동작하는 것과 Linux 환경에서 동작하는 것이, 각각 배포되고 있다. 디폴트에서는, 스태틱 라이브러리(.lib 혹은 .a)의 형식에서 제공되지만, 소스로부터 다이내믹 링크(혹은, 공유) 라이브러리를 작성할 수도 있다. blob(이)란, 데이터베이스로 말할 곳의 형태가 아니고, "작은 덩어리" 그렇다고 한다 의미로 이용되고 있다(컴퓨터 그래픽스나 비전의 세계에서는, 메타보르를 표현할 때에 이용되기도 한다). 즉, 이미지중으로부터 개별의 작은 덩어리(blob)(을)를 추출하는 작업(≈라벨링) (을)를 행하기 위한 클래스이며, 이하의 특징이 있다(자세한 것은, 상술의 홈 페이지를 참조하는 것).

- 2 치 이미지 혹은 그레이 스케일 이미지로부터, 8 근방에 있어서 연결되어 있는 영역을 추출한다. 이러한 영역은 blob (으)로서 참조된다.
- 이미지중의 주목 오브젝트를 얻기 위해서, 취득했다 blob 집합에 대해서 필터링을 실시한다. 이 처리는, CBlobResult 의 Filter 메소드에 의해서 실현된다.

## 샘플

## 라벨링 CBlobResult

입력 이미지에 대해서 라벨링을 실시해, 면적과 주위장을 표시한다

## 샘플 코드

## 표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include "BlobResult.h"

int
main (int argc, char **argv)
{
    int i;
    IplImage *src_img = 0, *dst_img = 0;
    CBlobResult blobs;
    CBlob blob;
    CvPoint p1, p2;
    CvFont font;
    char text[64];
    CBlob blobWithBiggestPerimeter, blobWithLessArea;

    // (1) 이미지를 읽어들인다
    if (argc != 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
```

```

    return -1;
dst_img = cvCreateImage (cvSize (src_img->width, src_img->height), IPL_DEPTH_8U, 1);
cvCvtColor (src_img, dst_img, CV_BGR2GRAY);
cvInitFont (&font, CV_FONT_HERSHEY_SIMPLEX, 0.5, 0.5, 0, 1, CV_AA);

// (2)라벨링을 실시한다
blobs = CBlobResult (dst_img, NULL, 100, false);

// (3)면적으로 필터링
blobs.Filter (blobs, B_INCLUDE, CBlobGetArea (), B_INSIDE, 1000, 50000);

for (i = 0; i < blobs.GetNumBlobs (); i++) {
    // (4)각 blob 의 최소 포함 구형(기울기 없음)을 요구한다
    blob = blobs.GetBlob (i);
    p1.x = (int) blob.MinX ();
    p1.y = (int) blob.MinY ();
    p2.x = (int) blob.MaxX ();
    p2.y = (int) blob.MaxY ();

    // (5)각 blob 의 포함 구형, 인덱스, 면적, 주장을 그리기
    cvRectangle (src_img, p1, p2, CV_RGB (255, 0, 0), 2, 8, 0);
    sprintf (text, "[%d] %d,%d", blob.Label (), (int) blob.area, (int) blob.Perimeter
());
    cvPutText (src_img, text, cvPoint (p1.x, p1.y - 5), &font, cvScalarAll (255));
}

// (6)최대의 주장을 가진다 blob(을)를, 녹색으로 전부 칠한다
blobs.GetNthBlob (CBlobGetPerimeter (), 0, blobWithBiggestPerimeter);
blobWithBiggestPerimeter.FillBlob (src_img, CV_RGB (0, 255, 0));

// (7)최소의 면적을 가진다 blob(을)를, 청색으로 전부 칠한다
blobs.GetNthBlob (CBlobGetArea (), blobs.GetNumBlobs () - 1, blobWithLessArea);
blobWithLessArea.FillBlob (src_img, CV_RGB (0, 0, 255));

// (8)이미지를 표시, 키가 밀렸을 때에 종료
cvNamedWindow ("Labeling", CV_WINDOW_AUTOSIZE);
cvShowImage ("Labeling", src_img);
cvWaitKey (0);

cvDestroyWindow ("Labeling");
cvReleaseImage (&src_img);
cvReleaseImage (&dst_img);

return 0;
}

```

```
// (1)이미지를 읽어들인다
입력 이미지를, 칼라 이미지로서 읽어들인다.또, 라벨링 대상이 되는 이미지는, 그레이 스케일
혹은 2 치 이미지일 필요가 있으므로, 이미지의 변환을 실시한다.

// (2)라벨링을 실시한다
1 번째의 인수에, 라벨링 대상이 되는 이미지를 준다. 2 번째의 인수는, 마스크 이미지,3 번째의
인수는,blob 인가 그 이외인지를 결정하는 반응을 일으키는 최소의 물리량(0-255), 4 번째의
인수는, 각 blob 모멘트의 계산을 실시하는지 아닌지의 플래그이다. 반환값으로서
오브젝트 CBlobResult(을)를 돌려준다.

// (3)면적으로 필터링
Filter()메소드는, 지정된 프롭퍼티에 의해,blob(을)를 필터링 한다. 1 번째의 인수는, 전의 라벨링
처리로 얻을 수 있던 오브젝트. 2 번째의 인수는, 필터링 작업(조건에 들어맞았다 blob(을)를,
포함한다"B_INCLUDE"인가, 포함하지 않는다"B_EXCLUDE"인가). 3 번째의 인수는, 필터링에
이용하는 프롭퍼티를 결정하는 함수. 4 번째의 인수는, 필터링
조건(B_EQUAL,B_NOT_EQUAL,B_GREATER,B_LESS,B_GREATER_OR_EQUAL,B_LESS_OR_EQU
AL,B_INSIDE,B_OUTSIDE) 5 번째와 6 번째의 인수는, 하한 및 상한(이것은, 지정되지 않는 것도
있다)이 되는 반응을 일으키는 최소의 물리량.
이번은, 면적이,1000 - 50000 [pixel] 의 범위에 있다 blob 만을 남기고 있다.

// (4)각 blob 의 최소 포함 구형(기울기 없음)을 요구한다
각 blob 에 두어 최소, 최대의 x,y 좌표를 가지는 픽셀을 참조하는 것으로, 그 blob(을)를 포함
하는 구형을 요구한다.

// (5)각 blob 의 포함 구형, 인덱스, 면적, 주위장을 그리기
각 blob 의 포함 구형, 및 blob 인덱스, 면적, 주위장을 그리기 한다. 여기서,blob 의
주위장은,blob 의 경계가 되는 픽셀의 개수이며, Edges()메소드에 의해 참조(CvSeq*형태) 할 수
있다. 다만, 이러한 픽셀은, 래스터 스캔 되었을 때의 주사순서에 줄지어 있을 뿐(만큼)이므로,
그대로 윤곽을 그리는 일은 할 수 없다.

// (6)최대의 주위장을 가진다 blob(을)를, 녹색으로 전부 칠한다
CBlobGetPerimeter()에 의해서 얻을 수 있었다 blob 의 주위의 길이가, 최대(0 번째 )의 것을,
CBlob 오브젝트로서 취득해,FillBlob()메소드에 의해서 전부 칠한다.

// (7)최소의 면적을 가진다 blob(을)를, 청색으로 전부 칠한다
(6)와)과 같게,CBlobGetArea()에 의해서 얻을 수 있었다 blob 의 면적이, 최소
(blobs.GetNumBlobs()-1 번째 , 즉 최후)의 것을, CBlob 오브젝트로서 취득해,FillBlob()메소드에
의해서 전부 칠한다.

// (8)이미지를 표시, 키가 밀렸을 때에 종료
이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.
```

### 실행 결과예

[좌]처리전 [우]처리 후

