이 글을 읽기 전에 : http://darkpgmr.tistory.com/64

Histograms

calcHist

Calculates a histogram of a set of arrays.

배열집합의 히스토그램을 계산한다.

```
C++: void calcHist(const Mat* images, int nimages, const int* channels, InputArray mask,
                   OutputArray hist, int dims, const int* histSize, const float** ranges,
                   bool uniform=true, bool accumulate=false )

C++: void calcHist(const Mat* images, int nimages, const int* channels, InputArray mask,
                   SparseMat& hist, int dims, const int* histSize, const float** ranges,
                   bool uniform=true, bool accumulate=false )

Python: cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) → hist

C: void cvCalcHist(IplImage** image, CvHistogram* hist, int accumulate=0,
                   const CvArr* mask=NULL )

Python: cv.CalcHist(image, hist, accumulate=0, mask=None) → None
```

Parameters:	
	● images - Source arrays. They all should have the same depth, CV_8U or CV_32F, and the same size. Each of them can have an arbitrary number of channels. 소스 배열들. 이미지는 모두 , 같은 depth, CV_8U or CV_32F, 같은 size 이어야 한다.각각은 임의 갯수 의 채널을 가질 수 있다. ● nimages - Number of source images. 소스 이미지의 수. ● channels - List of the dims channels used to compute the histogram. The first array channels are numerated from 0 to images[0].channels()-1, the second array channels are counted from images[0].channels() to

images[0].channels() + images[1].channels()-1, and so on.

히스토그램을 계산하는 데 사용되는 dims(아래에 있는 파라메터) 채널 목록이다.

- **mask** - Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as images[i]. The non-zero mask elements mark the array elements counted in the histogram.

  옵셔널 마스크. 만약 행렬이 비어있지 않으면, images[i]와 동일한 크기의 8-bit array이어야 한다. 0이 아닌 마스크 요소는 히스토그램에 집계된 배열 요소를 표시한다.

- **hist** - Output histogram, which is a dense or sparse dims -dimensional array.

  조밀하거나(dense) 희소한(sparse) dims(아래에 있는 파라메터) 배열 형태의, 아웃풋 히스토그램.

- **dims** - Histogram dimensionality that must be positive and not greater than CV_MAX_DIMS (equal to 32 in the current OpenCV version).

  히스토그램 dimensionality는 Positive여야 하며, CV_MAX_DIMS보다 크지 않아야 한다. (현재 OpenCV 버전에서 32와 동일)

- **histSize** - Array of histogram sizes in each dimension.

  각 차원의 히스토그램 크기 배열이다.

- **ranges** - Array of the dims arrays of the histogram bin boundaries in each dimension.

  각 차원(dimension)에서, 히스토그램 bin 경계(boundaries)의, dims array의 배열이다.

  When the histogram is uniform (uniform =true), then for each dimension i it is enough to specify the lower (inclusive) boundary L_0 of the 0-th histogram bin and the upper (exclusive) boundary $U_{histSize[i]-1}$ for the last histogram bin histSize[i]-1.

  히스토그램이 균일할 때(uniform = true), 각 차원 i에 대해 0 번째 히스토그램 bin의 더 낮은(포함하는) 경계 L_0와 마지막 히스토그램 bin histSize[i]-1에 대한 상위(배타적인) 경계 $U_{histSize[i]-1}$를 지정하면 충분합니다.

  That is, in case of a uniform histogram each of ranges[i] is an array of 2 elements.

  즉, 균일 히스토그램의 경우 ranges[i] 각각은 2개의 요소로 구성된 배열이다.

  When the histogram is not uniform (uniform=false), then each of ranges[i] containshistSize[i]+1 elements:

  히스토그램이 일정하지 않은 경우(uniform=false),

ranges[i] , containshistSize[i]+1의 각각의 요소 :

$L_0, U_0 = L_1, U_1 = L_2, ..., U_{histSize[i]-2} = L_{histSize[i]-1}, U_{histSize[i]-1}$

The array elements, that are not between L_0 and $U_{histSize[i]-1}$,

are not counted in the histogram.

L_0 와 $U_{histSize[i]-1}$ 사이에 있지 않은 배열요소는 히스토그램에서 계산되지 않는다.

- uniform - Flag indicating whether the histogram is uniform or not (see above).

  히스토그램이 균일한지 여부를 나타내는 플래그이다. (위를 참조)

- accumuulate - Accumulation flag. If it is set, the histogram is not cleared in

  the beginning when it is allocated. This feature enables you to compute

  a single histogram from several sets of arrays, or to update the histogram in time.

  누적 플래그. 이 값이 셋팅이 되면, 히스토그램이 할당될 때, 처음부터 지워지지 않는다. 이 기능을 사용하면 여러세트의 배열에서 단일 히스토그램을 계산하거나, 실행되는 동안에(in time) 히스토그램을 업데이트 할 수 있다.

The functions calcHist calculate the histogram of one or more arrays. The elements of a tuple

used to increment a histogram bin are taken from the corresponding input arrays at the same

location. The sample below shows how to compute a 2D Hue-Saturation histogram for a color image.

함수 calcHist는 하나 이상의 배열의 히스토그램을 계산합니다. 히스토그램 bin을 증가시키는 데 사용되는 튜플의 요소는 동일한 위치의 해당 입력 배열에서 가져옵니다. 아래 샘플은 컬러 이미지에 대한 2D 색조 - 채도 막대 그래프(2D Hue-Saturation histogram)를 계산하는 방법을 보여줍니다.

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

using namespace cv;

int main( int argc, char** argv )
{
    Mat src, hsv;
    if( argc != 2 || !(src=imread(argv[1], 1)).data )
        return -1;

    cvtColor(src, hsv, CV_BGR2HSV);

    // Quantize the hue to 30 levels
    // and the saturation to 32 levels
    int hbins = 30, sbins = 32;
    int histSize[] = {hbins, sbins};
    // hue varies from 0 to 179, see cvtColor
    float hranges[] = { 0, 180 };
    // saturation varies from 0 (black-gray-white) to
    // 255 (pure spectrum color)
    float sranges[] = { 0, 256 };
    const float* ranges[] = { hranges, sranges };
    MatND hist;
    // we compute the histogram from the 0-th and 1-st channels
    int channels[] = {0, 1};

    calcHist( &hsv, 1, channels, Mat(), // do not use mask
              hist, 2, histSize, ranges,
              true, // the histogram is uniform
              false );
    double maxVal=0;
    minMaxLoc(hist, 0, &maxVal, 0, 0);
```

```
    int scale = 10;
    Mat histImg = Mat::zeros(sbins*scale, hbins*10, CV_8UC3);

    for( int h = 0; h < hbins; h++ )
        for( int s = 0; s < sbins; s++ )
        {
            float binVal = hist.at<float>(h, s);
            int intensity = cvRound(binVal*255/maxVal);
            rectangle( histImg, Point(h*scale, s*scale),
                        Point( (h+1)*scale - 1, (s+1)*scale - 1),
                        Scalar::all(intensity),
                        CV_FILLED );
        }

    namedWindow( "Source", 1 );
    imshow( "Source", src );

    namedWindow( "H-S Histogram", 1 );
    imshow( "H-S Histogram", histImg );
    waitKey();
}
```

Note

- An example for creating histograms of an image can be found at opencv_source_code/samples/cpp/demhist.cpp
- (Python) An example for creating color histograms can be found at opencv_source/samples/python2/color_histogram.py
- (Python) An example illustrating RGB and grayscale histogram plotting can be found at opencv_source/samples/python2/hist.py

calcBackProject

Calculates the back projection of a histogram.

히스토그램의 역투영을 계산한다.

C++: void calcBackProject(const Mat* images, int nimages, const int* channels, InputArray hist, OutputArray backProject, const float** ranges, double scale=1, bool uniform=true)

C++: void calcBackProject(const Mat* images, int nimages, const int* channels, const SparseMat& hist, OutputArray backProject, const float** ranges, double scale=1, bool uniform=true)

Python: cv2.calcBackProject(images, channels, hist, ranges, scale[, dst]) → dst

C: void cvCalcBackProject(IplImage** image, CvArr* backProject, const CvHistogram* hist)

Python: cv.CalcBackProject(image, back_project, hist) → None

Parameters:	
	● images - Source arrays. They all should have the same depth, CV_8U or CV_32F, and the same size. Each of them can have an arbitrary number of channels. 소스 배열. 이미지들은 모두 같은 depth, CV_8U 또는 CV_32F, 같은 size를 가져야 한다. 각각은 임의의 수의 채널을 가질 수 있다.
	● nimages - Number of source images. 원본 이미지의 수.
	● channels - The list of channels used to compute the back projection. 역투영을 계산하는 데 사용 된 채널 목록이다. The number of channels must match the histogram dimensionality. 채널 수는 히스토그램 차원과 일치해야합니다. The first array channels are numerated from 0 to images[0].channels()-1 , the second array channels are counted from images[0].channels() to images[0].channels() + images[1].channels()-1, and so on. 첫번째 배열 채널은 0에서 images[0].channels()-1까지, 두번째 배열 채널은 images[0].channels() 에서 images[0].channels()+images[1].channels()-1 까지 계산된다.

	• hist - Input histogram that can be dense or sparse. dense 또는 sparse 히스토그램을 입력하라. • backProject - Destination back projection array that is a single-channel array of the same size and depth asimages[0]. 동일한 크기와 깊이의 단일 채널 배열인, 대상 역투영 배열(Destination back projection array) asimages[0] • ranges - Array of arrays of the histogram bin boundaries in each dimension. See calcHist() . 각 차원에서 히스토그램 bin 경계의 배열.. 의 배열 • scale - Optional scale factor for the output back projection. 아웃풋 역투영을 위한 선택적 축척 계수(Optional scale factor). • uniform - Flag indicating whether the histogram is uniform or not (see above). 히스토그램이 균일한지 여부를 나타내는 플래그입니다 (위 참조).

The functions calcBackProject calculate the back project of the histogram. That is, similarly to calcHist , at each location (x, y) the function collects the values from the selected channels in the input images and finds the corresponding histogram bin.
But instead of incrementing it, the function reads the bin value, scales it by scale , and stores in backProject(x,y) . In terms of statistics, the function computes probability of each element value in respect with the empirical probability distribution represented by the histogram. See how, for example, you can find and track a bright-colored object in a scene:
함수 calcBackProject는 히스토그램의 백 프로젝트를 계산합니다. 즉, calcHist와
마찬가지로 각 위치 (x, y)에서 함수는 입력 이미지에서 선택된 채널의 값을 수집하고
해당 히스토그램 bin을 찾습니다.
그러나 함수를 증가시키는 대신 함수는 bin 값을 읽고 크기에 따라 크기를 조정하고 backProject (x, y)에
저장합니다. 통계의 측면에서, 함수는 히스토그램에 의해 표현된 경험적 확률 분포와
관련하여 각 요소 값의 확률을 계산합니다. 예를 들어 장면에서 밝은 색상의 물체를 찾고
추적하는 방법을 확인하십시오.

1. Before tracking, show the object to the camera so that it covers almost the whole frame.
   Calculate a hue histogram. The histogram may have strong maximums, corresponding to the dominant colors in the object.

추적하기 전에 개체를 카메라에 표시하여 거의 전체 프레임을 덮습니다. 색조 막대 그래프를 계산합니다. 히스토그램은 개체의 지배적인 색에 해당하는 강한 최대 값을 가질 수 있습니다.

2. When tracking, calculate a back projection of a hue plane of each input video frame using that pre-computed histogram. Threshold the back projection to suppress weak colors.

   It may also make sense to suppress pixels with non-sufficient color saturation and too dark or too bright pixels.

   추적할 때 미리 계산된 히스토그램을 사용하여 각 입력 비디오 프레임의 색조 평면의 역투영을 계산합니다. 약한 색상을 억제하기 위해 역투영을 임계값으로 지정합니다. 또한 불충분한 채도와 너무 어둡거나 너무 밝은 픽셀을 억제하는 것이 좋습니다.

3. Find connected components in the resulting picture and choose, for example, the largest

   component.

   결과 그림에서 연결된 구성 요소를 찾아 예를 들어 가장 큰 구성 요소를 선택합니다.

This is an approximate algorithm of the CamShift() color object tracker.

이것은 CamShift() 색상 객체 추적의 대략적인 알고리즘입니다.

See also : calcHist()

compareHist

Compares two histograms.

두 개의 히스토그램을 비교한다.

C++: double compareHist(InputArray H1, InputArray H2, int method)

C++: double compareHist(const SparseMat& H1, const SparseMat& H2, int method)

Python: cv2.compareHist(H1, H2, method) → retval

C: double cvCompareHist(const CvHistogram* hist1, const CvHistogram* hist2, int method)

Python: cv.CompareHist(hist1, hist2, method) → float

Parameters:	
	● H1 - First compared histogram. ● H2 - Second compared histogram of the same size as H1 . ● method - Comparison method that could be one of the following: 비교방법 ○ CV_COMP_CORREL Correlation ○ CV_COMP_CHISQR Chi-Square ○ CV_COMP_INTERSECT Intersection ○ CV_COMP_BHATTACHARYYA Bhattacharyya distance ○ CV_COMP_HELLINGER Synonym for CV_COMP_BHATTACHARYYA

The functions compareHist compare two dense or two sparse histograms using the specified method:

compareHist 함수는 지정된 메서드를 사용하여 두 개의 밀도(sparse)가 높은, 또는 두 개의 희소(dense)
히스토그램을 비교합니다.

- Correlation (method=CV_COMP_CORREL)

  상관관계

  (상관관계 이해를 위한 참고자료 :

  https://m.blog.naver.com/PostView.nhn?blogId=istech7&logNo=50153047118&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F)

$$d(H_1,H_2)=\frac{\sum_I (H_1(I)-\bar{H_1})(H_2(I)-\bar{H_2})}{\sqrt{\sum_I (H_1(I)-\bar{H_1})^2 \sum_I (H_2(I)-\bar{H_2})^2}}$$

  where

$$\bar{H_k}=\frac{1}{N}\sum_J H_k(J)$$

  and N is a total number of histogram bins.

  N 은 히스토그램 bin 의 총 갯수 이다.

- Chi-Square (method=CV_COMP_CHISQR)

  카이-제곱

$$d(H_1,H_2)=\sum_I \frac{(H_1(I)-H_2(I))^2}{H_1(I)}$$

- Intersection (method=CV_COMP_INTERSECT)

  교집합

$$d(H_1,H_2)=\sum_I \min(H_1(I),H_2(I))$$

- Bhattacharyya distance

  (method=CV_COMP_BHATTACHARYYA or method=CV_COMP_HELLINGER).

  In fact, OpenCV computes Hellinger distance, which is related to Bhattacharyya coefficient.바타차랴 거리측정법. 사실,OpenCV는 Bhattacharyya계수와 관련된 Hellinger 거리를 계산합니다.

$$d(H_1,H_2)=\sqrt{1-\frac{1}{\sqrt{\bar{H_1}\bar{H_2}N^2}}\sum_I \sqrt{H_1(I)\cdot H_2(I)}}$$

  The function returns $d(H_1,H_2)$.

While the function works well with 1-, 2-, 3-dimensional dense histograms, it may not be suitable for high-dimensional sparse histograms. In such histograms, because of aliasing and sampling problems, the coordinates of non-zero histogram bins can slightly shift. To

compare such histograms or more general sparse configurations of weighted points, consider using the EMD() function.

이 함수는 1 차원, 2 차원, 3 차원 조밀 히스토그램에서 잘 작동하지만 고차원 희소(sparse) 히스토그램에는 적합하지 않을 수 있습니다. 이러한 히스토그램에서는 앨리어싱 및 샘플링 문제로 인해 0이 아닌 히스토그램 빈의 좌표가 약간 이동할 수 있습니다. 이러한 히스토그램 또는보다 일반적인 희소(sparse) 구성의 가중치 점을 비교하려면 EMD () 함수 사용을 고려하십시오.

EMD

Computes the "minimal work" distance between two weighted point configurations.

```
C++: float EMD(InputArray signature1, InputArray signature2, int distType,
                    InputArray cost=noArray(), float* lowerBound=0, OutputArray flow=noArray() )

C: float cvCalcEMD2(const CvArr* signature1, const CvArr* signature2, int distance_type,
                        CvDistanceFunction distance_func=NULL, const CvArr* cost_matrix=NULL,
                    CvArr* flow=NULL, float* lower_bound=NULL, void* userdata=NULL )

Python: cv.CalcEMD2(signature1, signature2, distance_type, distance_func=None,
                        cost_matrix=None, flow=None, lower_bound=None, userdata=None) → float
```

Parameters:

- signature1 - First signature, a $\mathbf{size1} \times \mathbf{dims} + 1$ floating-point matrix.
  Each row stores the point weight followed by the point coordinates. The matrix is allowed to have a single column (weights only) if the user-defined cost matrix is used.
  첫번째 signature, $\mathbf{size1} \times \mathbf{dims} + 1$ 부동소수점 행렬이다.
  각 행은 포인트 가중치와 포인트 좌표를 저장합니다.
  사용자 정의 비용 행렬이 (user-defined cost matrix) 사용되는 경우, 행렬에 단일 열 (가중치 만)이
  허용됩니다.
- signature2 - Second signature of the same format as signature1 , though the number of rows may be different. The total weights may be different. In this case an extra "dummy" point is added to eithersignature1 or signature2.

행의 수가 다를 수 있지만, signature1과 같은 포멧의 두번째 signature 이다. 총 무게는 다를수 있다. 이경우 추가"더미"포인트가 signature1또는 signature2 에 추가된다.

- distType - Used metric. CV_DIST_L1, CV_DIST_L2 and CV_DIST_C stand for one of the standard metrics. CV_DIST_USER means that a pre-calculated cost matrix cost is used.
  사용된 측정 항목이다. CV_DIST_L1, CV_DIST_L2 및 CV_DIST_Cst와 표준 메트릭 중 하나에 대한 값이다. CV_DIST_USER는 사전계산코스트 매트릭스의 비용이 사용됨을 의미한다.(pre-calculated cost matrix cost is used.)
- distance_func - Custom distance function supported by the old interface. CvDistanceFunction is defined as:
  이전 인터페이스가 지원하는 사용자 정의 거리 기능. CvDistanceFunction은 다음과 같이 정의된다.

```
typedef float (CV_CDECL * CvDistanceFunction)( const float* a,
                                               const float* b, void* userdata );
```

  where a and b are point coordinates and userdata is the same as the last parameter.
  여기서 a와 b는 점 좌표이고 userdata는 마지막 매개 변수와 같습니다.
- cost - User-defined **size1** × **size2** cost matrix. Also, if a cost matrix is used, lower boundary lowerBound cannot be calculated because it needs a metric function.
  사용자 정의 **size1** × **size2** 비용 매트릭스. 또한 비용 행렬을 사용하는 경우 메트릭 기능이 필요하기 때문에 lowerBound를 계산할 수 없다.
- lowerBound - Optional input/output parameter: lower boundary of a distance between the two signatures that is a distance between mass centers.
  선택적 입력/출력 매개변수:mass center사이거리인 두signature사이거리의 하한선.
  The lower boundary may not be calculated if the user-defined cost matrix is used, the total weights of point configurations are not equal, or if the signatures consist of weights only (the signature matrices have a single column).
  사용자 정의 비용 행렬이 사용되거나 포인트 구성의 총 가중치가 같지 않거나 서명이 가중치로만 구성된 경우 (signature 행렬에 단일 열 이 있는 경우) 하위 경계가 계산되지 않을 수 있다
  You must initialize *lowerBound . If the calculated distance between mass centers is greater or equal to*lowerBound (it means that the signatures are far enough), the function does not calculate EMD.
  반드시 *lowerBound를 초기화 해야 한다. mass center 간의 계산된 거리가 *lowerBound 보다

크거나 같다면 (signature가 충분히 멀리 있음을 의미함) 함수는 EMD를 계산하지 않는다.

In any case *lowerBound is set to the calculated distance between mass centers on return.

Thus, if you want to calculate both distance between mass centers and EMD,

*lowerBound should be set to 0.

어쨌든 *lowerBound는 반환할 mass center간의 계산된 거리로 설정된다. 따라서 mass center와

EMD 사이의 거리를 계산하려면 * lowerBound를 0으로 설정해야한다.

- flow - Resultant **size1** × **size2** flow matrix: $\mathbf{flow}_{i,j}$ is a flow from i -th point of signature1

  to j -th point of signature2 .

  결과 **size1** × **size2** 흐름 행렬 : $\mathbf{flow}_{i,j}$는 signature1의 첫 번째 지점에서 signature2의 j 번째 점까지의 흐름이다.

- userdata - Optional pointer directly passed to the custom distance function.

  맞춤형 거리 함수(custom distance function)에 직접 전달되는 선택적(Optional) 포인터.

The function computes the earth mover distance and/or a lower boundary of the distance between the two weighted point configurations. One of the applications described in [Rubner Sept98] is

multi-dimensional histogram comparison for image retrieval. EMD is a transportation problem

that is solved using some modification of a simplex algorithm, thus the complexity is exponential in the worst case, though, on average it is much faster.

In the case of a real metric the lower boundary can be calculated even faster (using linear-time

algorithm) and it can be used to determine roughly whether the two signatures are far enough

so that they cannot relate to the same object.

이 함수는 지구 이동자 거리(earth mover distance) 및 / 또는 두 개의 가중치 포인트 구성 사이의 거리의

하한 경계를 계산합니다. [RubnerSept98]에서 설명한 응용 프로그램 중 하나는 이미지 검색을 위한 다차원 히스토그램 비교입니다. EMD는 단방향 알고리즘의 일부 수정을

사용하여 해결되는 transportation 문제 이므로 최악의 경우 복잡도가 지수적(exponential)이다. 평균적으로 훨씬 빠르다.
실제 메트릭의 경우 하위 경계는 선형 시간 알고리즘(linear-time algorithm)을 사용하여 더욱 빠르게 계산할 수 있으며, 두 signature가 동일한 객체와 관련되지 않도록 충분히 멀리 있는지 여부를 대략적으로 결정 하는 데 사용할 수 있다.

equalizeHist

Equalizes the histogram of a grayscale image.

grayscale image 의 히스토그램을 균등화

C++: void equalizeHist(InputArray src, OutputArray dst)

Python: cv2.equalizeHist(src[, dst]) → dst

C: void cvEqualizeHist(const CvArr* src, CvArr* dst)

Parameters:	● src - Source 8-bit single channel image. ● dst - Destination image of the same size and type as src.

The function equalizes the histogram of the input image using the following algorithm:

이 함수는 다음 알고리즘을 사용하여 입력 이미지의 히스토그램을 균등화한다.

1. Calculate the histogram H for src.

   src의 히스토그램 H를 계산한다.

2. Normalize the histogram so that the sum of histogram bins is 255.

   히스토그램 bins의 합이 255가되도록 히스토그램을 표준화(Normalize) 하라.

3. Compute the integral of the histogram:

   히스토그램의 적분 값을 계산하라 :

   $$H'_i = \sum_{0 \le j < i} H(j)$$

4. Transform the image using H' as a look-up table: $\texttt{dst}(x,y) = H'(\texttt{src}(x,y))$

   룩업 테이블(look-up table)로 H' 를 사용하여, 이미지를 변환하라.

The algorithm normalizes the brightness and increases the contrast of the image.

이 알고리즘은 밝기를 표준화하고 이미지의 대비를 증가시킨다.

Extra Histogram Functions (C API)

The rest of the section describes additional C functions operating on CvHistogram.

이 절의 나머지 부분에서는 CvHistogram에서 작동하는 추가 C 함수에 대해 설명한다.

CalcBackProjectPatch

Locates a template within an image by using a histogram comparison.

히스토그램 비교를 사용하여, 이미지내 에서 템플릿을 찾는다.

C: void cvCalcBackProjectPatch(IplImage** images, CvArr* dst, CvSize patch_size,
CvHistogram* hist, int method, double factor)

Python: cv.CalcBackProjectPatch(images, dst, patch_size, hist, method, factor) → None

Parameters:

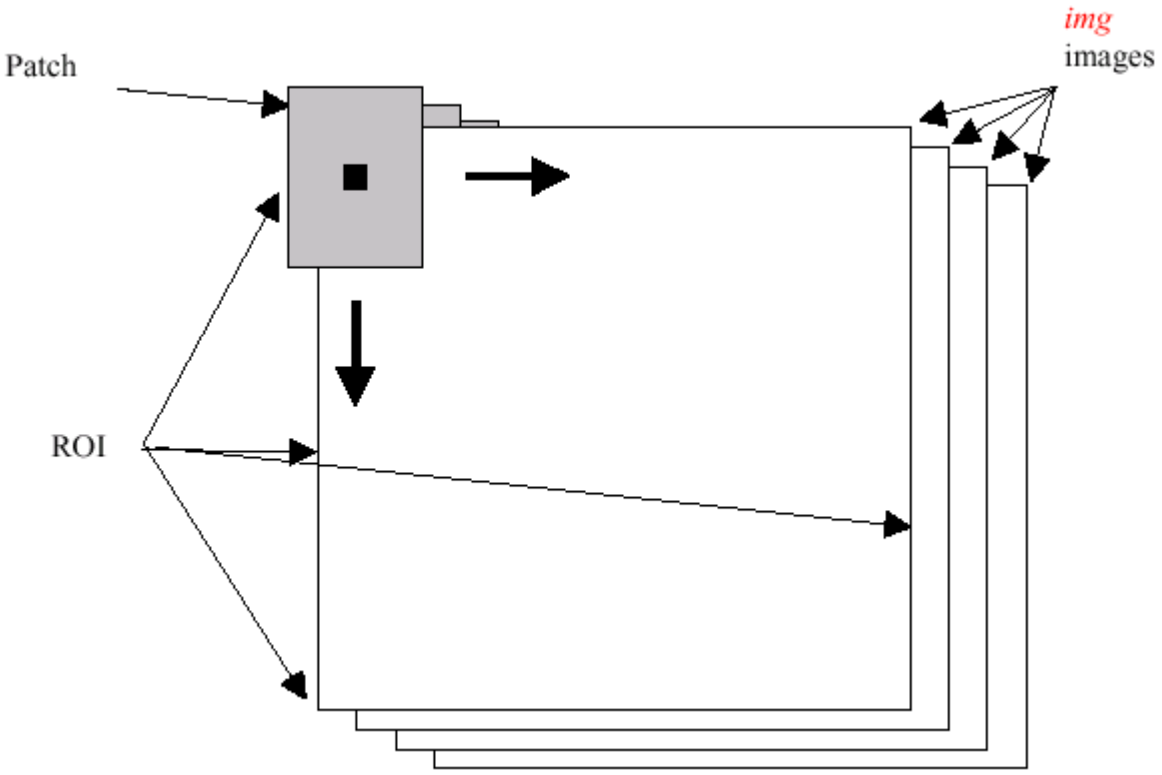
- images - Source images (though, you may pass CvMat** as well).
  소스 이미지 (CvMat** 도 전달할 수 있다.)
- dst - Destination image.
  대상 이미지
- patch_size - Size of the patch slide though the source image.
  소스 이미지를 통해 슬라이드 된 패치의 크기.
- hist - Histogram.
  히스토그램
- method - Comparison method passed to CompareHist() (see the function description).
  CompareHist()에 전달된 비교 메서드이다 (함수 설명 참조).
- factor - Normalization factor for histograms that affects the normalization scale of the destination image. Pass 1 if not sure.
  대상 이미지의 정규화 스케일에 영향을 주는 히스토그램에 대한 정규화 계수이다.
  확실하지 않으면 1을 전달하시오.

The function calculates the back projection by comparing histograms of the source image p atches with the given histogram. The function is similar to matchTemplate(), but instead of comparing

the raster patch with all its possible positions within the search window,

the function CalcBackProjectPatch compares histograms. See the algorithm diagram below:

이 함수는 소스 이미지 패치의 히스토그램을 주어진 히스토그램과 비교하여 역투영을 계산합니다. 이 함수는 matchTemplate()과 비슷하지만 래스터 패치(raster patch)를 검색 창의 모든 가능한 위치와 비교하는 대신 CalcBackProjectPatch 함수는 히스토그램을 비교합니다. 아래의 알고리즘 다이어그램을 참조하십시오.



CalcProbDensity

Divides one histogram by another.

한 히스토그램을 다른 히스토그램으로 나눕니다.

C: void cvCalcProbDensity(const CvHistogram* hist1, const CvHistogram* hist2,
CvHistogram* dst_hist, double scale=255)

Python: cv.CalcProbDensity(hist1, hist2, dst_hist, scale=255) → None

Parameters:	
	● hist1 - First histogram (the divisor). ● hist2 - Second histogram. ● dst_hist - Destination histogram. ● scale - Scale factor for the destination histogram. 대상 히스토그램의 Scale 계수.

The function calculates the object probability density from two histograms as:

이 함수는 두 개의 히스토그램에서 객체 확률 밀도(object probability density)를 다음과 같이 계산합니다.

$$\texttt{disthist}(I) = \begin{cases} 0 & \text{if } \texttt{hist1}(I) = 0 \\ \texttt{scale} & \text{if } \texttt{hist1}(I) \neq 0 \text{ and } \texttt{hist2}(I) > \texttt{hist1}(I) \\ \frac{\texttt{hist2}(I) \cdot \texttt{scale}}{\texttt{hist1}(I)} & \text{if } \texttt{hist1}(I) \neq 0 \text{ and } \texttt{hist2}(I) \leq \texttt{hist1}(I) \end{cases}$$

ClearHist

Clears the histogram.

히스토그램 지우기.

C: void cvClearHist(CvHistogram* hist)

Python: cv.ClearHist(hist) → None

Parameters:	● hist - Histogram.

The function sets all of the histogram bins to 0 in case of a dense histogram and removes all histogram bins in case of a sparse array.

함수는 밀도(dense)가 높은 히스토그램의 경우 모든 히스토그램 bins를 0으로 설정하고 희소(sparse) 배열의 경우 모든 히스토그램 bin을 제거한다.

CopyHist

Copies a histogram.

히스토그램 복사.

C: void cvCopyHist(const CvHistogram* src, CvHistogram** dst)

Parameters:	● src - Source histogram. ● dst - Pointer to the destination histogram. 목적 히스토그램 의 포인터

The function makes a copy of the histogram. If the second histogram pointer *dst is NULL, a new histogram of the same size as src is created. Otherwise, both histograms must have equal types and sizes.

Then the function copies the bin values of the source histogram to the destination histogram and sets the same bin value ranges as in src.

이 함수는 히스토그램의 복사본을 만든다. 두 번째 히스토그램 포인터 *dst가 NULL이면 src와 동일한 크기의 새로운 히스토그램이 생성된다. 그렇지 않으면 두 히스토그램의 유형과 크기가 동일해야한다.

그런 다음 함수는 소스 히스토그램의 bin 값을 대상(destination)히스토그램에 복사하고 src와 동일한 bin 값 범위를 설정한다.

CreateHist

Creates a histogram.

히스토그램 만들기.

C: CvHistogram* cvCreateHist(int dims, int* sizes, int type, float** ranges=NULL, int uniform=1)

Python: cv.CreateHist(dims, type, ranges=None, uniform=1) → hist

Parameters:

- dims - Number of histogram dimensions.
  히스토그램 차원(dimensions) 의 갯수
- sizes - Array of the histogram dimension sizes.
  히스토그램 차원(dimensions) 사이즈 의 배열.
- type - Histogram representation format. CV_HIST_ARRAY means that the histogram data is represented as a multi-dimensional dense array CvMatND. CV_HIST_SPARSE means that histogram data is represented as a multi-dimensional sparse array CvSparseMat.
  히스토그램 표현 형식. CV_HIST_ARRAY 히스토그램 데이터가 다차원 밀집(dense) 배열 CvMatND로 나타내지는 것을 의미한다. CV_HIST_SPARSE는 히스토그램 데이터가 다차원 희소(sparse) 배열 CvSparseMat로 표현된다는 것을 의미한다.
- ranges - Array of ranges for the histogram bins. Its meaning depends on the uniform parameter value. The ranges are used when the histogram is calculated or backprojected to
  determine which histogram bin corresponds to which value/tuple of values from the input
  image(s).
  히스토그램 bins의 범위 배열이다. 그 의미는 uniform parameter 값에 달려있다. 범위는 히스토그램이 계산되거나 역투영되어 입력 이미지의 값 / 튜플에 해당하는 히스토그램 bin을 결정하는 데 사용된다.

- uniform - Uniformity flag. If not zero, the histogram has evenly spaced bins and for every $0 <= i < cDims$ranges[i] is an array of two numbers:
  lower and upper boundaries for the i-th histogram dimension.
  균등성 플래그. 0이 아니면 히스토그램은 균등한 간격의 bins를 가지며 모든 $0 <= i < cDims$ ranges[i] 에 대해 두 개의 숫자 배열이다 : i번째 히스토그램 차원의 상한 및 하한 경계.
  The whole range [lower,upper] is then split into dims[i] equal parts to determine the i-th input tuple value ranges for every histogram bin. And if uniform=0 ,
  then the i-th element of the ranges array contains dims[i]+1 elements:
  전체 범위 [lower, upper]는 모든 히스토그램 bin에 대해 i 번째 입력 튜플 값 범위를 결정하기 위해 동일한 부분으로 dims[i]로 분할된다. 그리고 만약 uniform = 0이면 범위 배열의 i 번째 요소에 dims [i] +1 요소가 포함된다.
  $lower_0, upper_0, lower_1, upper_1 = lower_2, ...upper_{dims[i]-1}$ where $lower_j$
  and $upper_j$ are lower and upper boundaries of the i-th input tuple value for
  the j-th bin, respectively. In either case, the input values that are beyond the specified range for a histogram bin are not counted by CalcHist() and filled with 0 by CalcBackProject().
  $lower_j$ 와 $upper_j$ 가 낮은
  $lower_0, upper_0, lower_1, upper_1 = lower_2, ...upper_{dims[i]-1}$,
  그리고 j 번째 bin에 대한 i 번째 입력 튜플 값의 상한을 각각 나타낸 경계.
  두 경우 모두 히스토그램 bin의 지정된 범위를 벗어난 입력값은 CalcHist() 에 의해 계산되지 않고 CalcBackProject() 에 의해 0으로 채운다.

The function creates a histogram of the specified size and returns a pointer to the created histogram. 이 함수는 지정된 크기의 히스토그램을 만들고, 만든 히스토그램에 대한 포인터를 반환한다.

If the array ranges is 0, the histogram bin ranges must be specified later via the function SetHistBinRanges().
배열 범위가 0이면, 히스토그램 bin 범위는 나중에 SetHistBinRanges() 함수를 통해 지정해야 한다.

Though CalcHist() and CalcBackProject() may process 8-bit images without setting bin ranges,
they assume they are equally spaced in 0 to 255 bins.
CalcHist() 및 CalcBackProject()는 bin 범위를 설정하지 않고 8비트 이미지를 처리할 수

있지만,

0부터 255개의 bin으로 동일한 간격을 유지한다고 가정한다.

GetMinMaxHistValue

Finds the minimum and maximum histogram bins.

최소 및 최대 히스토그램 bins을 찾는다.

C: void cvGetMinMaxHistValue(const CvHistogram* hist, float* min_value, float* max_value,
int* min_idx=NULL, int* max_idx=NULL)

Python: cv.GetMinMaxHistValue(hist)-> (min_value, max_value, min_idx, max_idx)

Parameters:	
	● hist - Histogram. ● min_value - Pointer to the minimum value of the histogram. 히스토그램의 최소값을 가리키는 포인터 ● max_value - Pointer to the maximum value of the histogram. 히스토그램의 최대값을 가리키는 포인터 ● min_idx - Pointer to the array of coordinates for the minimum. 최소치의 좌표 배열을 가리키는 포인터 ● max_idx - Pointer to the array of coordinates for the maximum. 최대치의 좌표 배열을 가리키는 포인터

The function finds the minimum and maximum histogram bins and their positions. All of out put arguments are optional.

Among several extremas with the same value the ones with the minimum index (in the lexico gra-phical order) are returned. In case of several maximums or minimums, the earliest in the lexicographical order (extrema locations) is returned.

이 함수는 최소 및 최대 히스토그램 bins와 그 위치를 찾는다. 모든 출력 인수는 선택 사항이다.

같은 값을 가지는 여러 임시값 중 최소 인덱스를 갖는 값이 반환된다(사전식 순서로).

최대치 또는 최소치가

여러 개인 경우, 사전식 순서 (극한치 위치)에서 "가장 처음들어온(earliest) 값"이 반환된다.

MakeHistHeaderForArray

Makes a histogram out of an array.

배열에서 히스토그램을 만든다.

```
C: CvHistogram* cvMakeHistHeaderForArray(int dims, int* sizes, CvHistogram* hist,
                                         float* data, float** ranges=NULL, int uniform=1 )
```

Parameters:	
	● dims - Number of the histogram dimensions. 히스토그램 차원의 갯수 ● sizes - Array of the histogram dimension sizes. 히스토그램 차원 사이즈의 갯수 ● hist - Histogram header initialized by the function. 히스토그램 헤더는 함수에 의해 초기화된다. ● data - Array used to store histogram bins. 히스토그램 bins를 저장하는 데 사용되는 배열이다. ● ranges - Histogram bin ranges. See CreateHist() for details. 히스토그램 bin 범위. 자세한 내용은 CreateHist ()를 참조하라 ● uniform - Uniformity flag. See CreateHist() for details. 균등성 플래그. 자세한 내용은 CreateHist ()를 참조하라

The function initializes the histogram, whose header and bins are allocated by the user. ReleaseHist() does not need to be called afterwards. Only dense histograms can be initialized

this way. The function returns hist.

함수는 사용자가 헤더와 bins를 할당한 히스토그램을 초기화한다. ReleaseHist()는 나중에 호출할 필요가

없다. 이런 식으로 밀도(dense)가 높은 히스토그램만 초기화 할 수 있다. 이 함수는 hist를 반환한다.

NormalizeHist

Normalizes the histogram.

히스토그램을 정규화한다.

C: void cvNormalizeHist(CvHistogram* hist, double factor)

Python: cv.NormalizeHist(hist, factor) → None

Parameters:	● hist - Pointer to the histogram. 히스토그램의 포인터 ● factor - Normalization factor. 정규화 계수

The function normalizes the histogram bins by scaling them so that the sum of the bins be comes equal to factor.

이 함수는 bins의 합이 factor와 같아지도록 히스토그램을 크기조정(scaling)하여 히스토그램을 표준화(normalizes) 한다.

ReleaseHist

Releases the histogram.

히스토그램을 해제한다.

C: void cvReleaseHist(CvHistogram** hist)

Parameters:	● hist - Double pointer to the released histogram. 릴리스된 히스토그램에 대한 이중포인터

The function releases the histogram (header and the data). The pointer to the histogram is cleared by the function. If *hist pointer is already NULL, the function does nothing.

이 함수는 히스토그램 (헤더와 데이터)을 해제한다. 히스토그램에 대한 포인터는 함수에 의해 지워진다.

만약 *hist 포인터가 이미 NULL이면 함수는 아무 작업도 수행하지 않습니다.

SetHistBinRanges

Sets the bounds of the histogram bins.

히스토그램 bins의 범위를 설정한다.

C: void cvSetHistBinRanges(CvHistogram* hist, float** ranges, int uniform=1)

Parameters:	● hist - Histogram. ● ranges - Array of bin ranges arrays. See CreateHist() for details. bin 범위 배열의 배열입니다. 자세한 내용은 CreateHist ()를 참조하십시오. ● uniform - Uniformity flag. See CreateHist() for details.

This is a standalone function for setting bin ranges in the histogram. For a more detailed

description of the parameters ranges and uniform, see the CalcHist()function that can initialize

the ranges as well. Ranges for the histogram bins must be set before the histogram is calculated or the backproject of the histogram is calculated.

이것은 히스토그램에서 bin 범위를 설정하는 독립형 기능이다. 매개 변수 범위와 uniform에 대한 자세한

설명은 범위를 초기화 할 수있는 CalcHist () 함수를 참조하십시오. 히스토그램 저장 공간의 범위는

히스토그램이 계산되거나 히스토그램의 backproject 프로젝트가 계산되기 전에 설정되어야한다.

ThreshHist

Thresholds the histogram.

히스토그램을 임계값으로 지정하기.

C: void cvThreshHist(CvHistogram* hist, double threshold)

Python: cv.ThreshHist(hist, threshold) → None

Parameters:	● hist - Pointer to the histogram. 히스토그램의 포인터 ● threshold - Threshold level. 임계값 레벨

The function clears histogram bins that are below the specified threshold.

이 함수는 지정된 임계값보다 낮은 히스토그램 bins를 지운다.

[RubnerSept98] 1. Y. Rubner. C. Tomasi, L.J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. Technical Report STAN-CS-TN-98-86, Department of Computer Science, Stanford University, September 1998.