- **minDistance** – Minimum possible Euclidean distance between the returned corners.
- **mask** – Optional region of interest. If the image is not empty (it needs to have the type `CV_8UC1` and the same size as `image`), it specifies the region in which the corners are detected.
- **blockSize** – Size of an average block for computing a derivative covariation matrix over each pixel neighborhood. See `cornerEigenValsAndVecs()` .
- **useHarrisDetector** – Parameter indicating whether to use a Harris detector (see `cornerHarris()`) or `cornerMinEigenVal()`.
- **k** – Free parameter of the Harris detector.

The function finds the most prominent corners in the image or in the specified image region, as described in [Shi94]:

1. Function calculates the corner quality measure at every source image pixel using the `cornerMinEigenVal()` or `cornerHarris()` .
2. Function performs a non-maximum suppression (the local maximums in *3 x 3* neighborhood are retained).
3. The corners with the minimal eigenvalue less than $\texttt{qualityLevel} \cdot \max_{x,y} \texttt{qualityMeasureMap}(x,y)$ are rejected.
4. The remaining corners are sorted by the quality measure in the descending order.
5. Function throws away each corner for which there is a stronger corner at a distance less than `maxDistance`.

The function can be used to initialize a point-based tracker of an object.

Note: If the function is called with different values `A` and `B` of the parameter `qualityLevel` , and `A` > {B}, the vector of returned corners with `qualityLevel=A` will be the prefix of the output vector with `qualityLevel=B` .

See also: `cornerMinEigenVal()`, `cornerHarris()`, `calcOpticalFlowPyrLK()`, `estimateRigidTransform()`,

HoughCircles

Finds circles in a grayscale image using the Hough transform.

C++: void **`HoughCircles`**(InputArray **image**, OutputArray **circles**, int **method**, double **dp**, double **minDist**, double **param1**=100, double **param2**=100, int **minRadius**=0, int **maxRadius**=0)

C: CvSeq* **`cvHoughCircles`**(CvArr* **image**, void* **circle_storage**, int **method**, double **dp**, double **min_dist**, double **param1**=100, double **param2**=100, int **min_radius**=0, int **max_radius**=0)

Python: `cv2.HoughCircles`(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]]) → circles ¶

Parameters:
- **image** – 8-bit, single-channel, grayscale input image.
- **circles** – Output vector of found circles. Each vector is encoded as a 3-element floating-point vector $(x, y, radius)$.
- **circle_storage** – In C function this is a memory storage that will contain the output sequence of found circles.
- **method** – Detection method to use. Currently, the only implemented method is `CV_HOUGH_GRADIENT` , which is basically *21HT* , described in [Yuen90].
- **dp** – Inverse ratio of the accumulator resolution to the image resolution. For example, if `dp=1` , the accumulator has the same resolution as the input image. If `dp=2` , the accumulator has half as big width and height.
- **minDist** – Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
- **param1** – First method-specific parameter. In case of `CV_HOUGH_GRADIENT` , it is the higher threshold of the two passed to the `Canny()` edge detector (the lower one is twice smaller).
- **param2** – Second method-specific parameter. In case of `CV_HOUGH_GRADIENT` , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
- **minRadius** – Minimum circle radius.
- **maxRadius** – Maximum circle radius.

The function finds circles in a grayscale image using a modification of the Hough transform.

Example:

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

using namespace cv;

int main(int argc, char** argv)
{
    Mat img, gray;
    if( argc != 2 && !(img=imread(argv[1], 1)).data)
        return -1;
    cvtColor(img, gray, CV_BGR2GRAY);
    // smooth it, otherwise a lot of false circles may be detected
    GaussianBlur( gray, gray, Size(9, 9), 2, 2 );
    vector<Vec3f> circles;
    HoughCircles(gray, circles, CV_HOUGH_GRADIENT,
                 2, gray->rows/4, 200, 100 );
    for( size_t i = 0; i < circles.size(); i++ )
    {
         Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
         int radius = cvRound(circles[i][2]);
         // draw the circle center
```

```
            circle( img, center, 3, Scalar(0,255,0), -1, 8, 0 );
            // draw the circle outline
            circle( img, center, radius, Scalar(0,0,255), 3, 8, 0 );
        }
        namedWindow( "circles", 1 );
        imshow( "circles", img );
        return 0;
    }
```

Note: Usually the function detects the centers of circles well. However, it may fail to find correct radii. You can assist to the function by specifying the radius range (`minRadius` and `maxRadius`) if you know it. Or, you may ignore the returned radius, use only the center, and find the correct radius using an additional procedure.

See also: `fitEllipse()`, `minEnclosingCircle()`

Note:

- An example using the Hough circle detector can be found at opencv_source_code/samples/cpp/houghcircles.cpp

HoughLines

Finds lines in a binary image using the standard Hough transform.

C++: void **HoughLines**(InputArray **image**, OutputArray **lines**, double **rho**, double **theta**, int **threshold**, double **srn**=0, double **stn**=0)

Python: cv2.**HoughLines**(image, rho, theta, threshold[, lines[, srn[, stn]]]) → lines

C: CvSeq* **cvHoughLines2**(CvArr* **image**, void* **line_storage**, int **method**, double **rho**, double **theta**, int **threshold**, double **param1**=0, double **param2**=0)

Python: cv.**HoughLines2**(image, storage, method, rho, theta, threshold, param1=0, param2=0) → lines

Parameters:

- **image** – 8-bit, single-channel binary source image. The image may be modified by the function.
- **lines** – Output vector of lines. Each line is represented by a two-element vector (ρ, θ) . ρ is the distance from the coordinate origin $(0, 0)$ (top-left corner of the image). θ is the line rotation angle in radians ($0 \sim \text{vertical line}, \pi/2 \sim \text{horizontal line}$).
- **rho** – Distance resolution of the accumulator in pixels.
- **theta** – Angle resolution of the accumulator in radians.
- **threshold** – Accumulator threshold parameter. Only those lines are returned that get enough votes ($> \texttt{threshold}$).
- **srn** – For the multi-scale Hough transform, it is a divisor for the distance resolution `rho` . The coarse accumulator distance resolution is `rho` and the