calibrateCamera() [1/2]

```
double cv::calibrateCamera ( InputArrayOfArrays  objectPoints,
                             InputArrayOfArrays  imagePoints,
                             Size                imageSize,
                             InputOutputArray    cameraMatrix,
                             InputOutputArray    distCoeffs,
                             OutputArrayOfArrays rvecs,
                             OutputArrayOfArrays tvecs,
                             OutputArray         stdDeviationsIntrinsics,
                             OutputArray         stdDeviationsExtrinsics,
                             OutputArray         perViewErrors,
                             int                 flags = 0 ,
                             TermCriteria        criteria = TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON)
                           )
```

Python:

```
cv.calibrateCamera(           objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]]
```

```
cv.calibrateCameraExtended( objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, stdDeviationsIntrinsics[, stdDeviationsExtrinsics[, perViewErrors[, flags[, criteria]]]
```

```
#include <opencv2/calib3d.hpp>
```

Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.

Parameters

objectPoints	In the new interface it is a vector of vectors of calibration pattern points in the calibration pattern coordinate space (e.g. std::vector<std::vector<cv::Vec3f>>). T many elements as the number of pattern views. If the same calibration pattern is shown in each view and it is fully visible, all the vectors will be the same. Alth partially occluded patterns or even different patterns in different views. Then, the vectors will be different. Although the points are 3D, they all lie in the calibrati plane (thus 0 in the Z-coordinate), if the used calibration pattern is a planar rig. In the old interface all the vectors of object points from different views are conc
imagePoints	In the new interface it is a vector of vectors of the projections of calibration pattern points (e.g. std::vector<std::vector<cv::Vec2f>>). imagePoints.size() and ob imagePoints[i].size() and objectPoints[i].size() for each i, must be equal, respectively. In the old interface all the vectors of object points from different views are
imageSize	Size of the image used only to initialize the camera intrinsic matrix.
cameraMatrix	Input/output 3x3 floating-point camera intrinsic matrix $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$. If CALIB_USE_INTRINSIC_GUESS and/or CALIB_FIX_ASPECT_RATIO, CALIB_FIX_PRINCIPAL_POINT or CALIB_FIX_FOCAL_LENGTH are specified, some or all of fx, fy, cx, cy must be initialized before calling the function.
distCoeffs	Input/output vector of distortion coefficients $(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$ of $4, 5, 8, 12$ or 14 elements.
rvecs	Output vector of rotation vectors (Rodrigues) estimated for each pattern view (e.g. std::vector<cv::Mat>>). That is, each i-th rotation vector together with the translation vector (see the next output parameter description) brings the calibration pattern from the object coordinate space (in which object points are specifi coordinate space. In more technical terms, the tuple of the i-th rotation and translation vector performs a change of basis from object coordinate space to came to its duality, this tuple is equivalent to the position of the calibration pattern with respect to the camera coordinate space.
tvecs	Output vector of translation vectors estimated for each pattern view, see parameter describtion above.
stdDeviationsIntrinsics	Output vector of standard deviations estimated for intrinsic parameters. Order of deviations values: $(f_x, f_y, c_x, c_y, k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6, s_1, s_2, s_3, s_$ parameters is not estimated, it's deviation is equals to zero.
stdDeviationsExtrinsics	Output vector of standard deviations estimated for extrinsic parameters. Order of deviations values: $(R_0, T_0, \dots, R_{M-1}, T_{M-1})$ where M is the number of pa concatenated 1x3 vectors.
perViewErrors	Output vector of the RMS re-projection error estimated for each pattern view.
flags	Different flags that may be zero or a combination of the following values: • CALIB_USE_INTRINSIC_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set t imageSize is used), and focal distances are computed in a least-squares fashion. Note, that if intrinsic parameters are known, there is no need to use th extrinsic parameters. Use solvePnP instead. • CALIB_FIX_PRINCIPAL_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specifie CALIB_USE_INTRINSIC_GUESS is set too. • CALIB_FIX_ASPECT_RATIO The functions consider only fy as a free parameter. The ratio fx/fy stays the same as in the input cameraMatrix . When CALIB_USE_INTRINSIC_GUESS is not set, the actual input values of fx and fy are ignored, only their ratio is computed and used further. • CALIB_ZERO_TANGENT_DIST Tangential distortion coefficients (p_1, p_2) are set to zeros and stay zero. • CALIB_FIX_FOCAL_LENGTH The focal length is not changed during the global optimization if CALIB_USE_INTRINSIC_GUESS is set. • CALIB_FIX_K1,..., CALIB_FIX_K6 The corresponding radial distortion coefficient is not changed during the optimization. If CALIB_USE_INTRINSIC_G coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0. • CALIB_RATIONAL_MODEL Coefficients k4, k5, and k6 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified function use the rational model and return 8 coefficients or more. • CALIB_THIN_PRISM_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly spe calibration function use the thin prism model and return 12 coefficients or more. • CALIB_FIX_S1_S2_S3_S4 The thin prism distortion coefficients are not changed during the optimization. If CALIB_USE_INTRINSIC_GUESS is set, th supplied distCoeffs matrix is used. Otherwise, it is set to 0. • CALIB_TILTED_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to r function use the tilted sensor model and return 14 coefficients. • CALIB_FIX_TAUX_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If CALIB_USE_INTRINSIC_GUESS is se supplied distCoeffs matrix is used. Otherwise, it is set to 0.
criteria	Termination criteria for the iterative optimization algorithm.

Returns

the overall RMS re-projection error.

The function estimates the intrinsic camera parameters and extrinsic parameters for each of the views. The algorithm is based on [254] and [31] . The coordinates of 3D object points and their projections in each view must be specified. That may be achieved by using an object with known geometry and easily detectable feature points. Such an object is called a calibration rig or calib

OpenCV has built-in support for a chessboard as a calibration rig (see **findChessboardCorners**). Currently, initialization of intrinsic parameters (when **CALIB_USE_INTRINSIC_GUESS** is not for planar calibration patterns (where Z-coordinates of the object points must be all zeros). 3D calibration rigs can also be used as long as initial cameraMatrix is provided.

The algorithm performs the following steps:

- Compute the initial intrinsic parameters (the option only available for planar calibration patterns) or read them from the input parameters. The distortion coefficients are all set to zeros init CALIB_FIX_K? are specified.
- Estimate the initial camera pose as if the intrinsic parameters have been already known. This is done using **solvePnP** .
- Run the global Levenberg-Marquardt optimization algorithm to minimize the reprojection error, that is, the total sum of squared distances between the observed feature points imagePoin the current estimates for camera parameters and the poses) object points objectPoints. See **projectPoints** for details.

Note

If you use a non-square (i.e. non-N-by-N) grid and **findChessboardCorners** for calibration, and **calibrateCamera** returns bad values (zero distortion coefficients, c_x and c_y very far from large differences between f_x and f_y (ratios of 10:1 or more)), then you are probably using patternSize=cvSize(rows,cols) instead of using patternSize=cvSize(cols,rows) in **findChessbo**

See also

findChessboardCorners, **solvePnP**, **initCameraMatrix2D**, **stereoCalibrate**, **undistort**