JavaScript is disabled on your browser.

* [Overview](http://docs.google.com/overview-summary.html)
* [Package](http://docs.google.com/package-summary.html)
* Class
* [Tree](http://docs.google.com/package-tree.html)
* [Index](http://docs.google.com/index-all.html)
* [Help](http://docs.google.com/help-doc.html)
* Prev Class
* [Next Class](http://docs.google.com/org/opencv/calib3d/StereoBM.html)
* [Frames](http://docs.google.com/index.html?org/opencv/calib3d/Calib3d.html)
* [No Frames](http://docs.google.com/Calib3d.html)
* [All Classes](http://docs.google.com/allclasses-noframe.html)
* Summary:
* Nested |
* [Field](#3znysh7) |
* [Constr](#2et92p0) |
* [Method](#tyjcwt)
* Detail:
* [Field](#1t3h5sf) |
* [Constr](#pkwqa1) |
* [Method](#1opuj5n)

org.opencv.calib3d

## Class Calib3d

* java.lang.Object
  + org.opencv.calib3d.Calib3d
* public class Calib3d  
  extends java.lang.Object

### Field SummaryFields

| Modifier and Type | Field and Description |
| --- | --- |
| static int | [**CALIB\_CB\_ADAPTIVE\_THRESH**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_ADAPTIVE_THRESH) |
| static int | [**CALIB\_CB\_ASYMMETRIC\_GRID**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_ASYMMETRIC_GRID) |
| static int | [**CALIB\_CB\_CLUSTERING**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_CLUSTERING) |
| static int | [**CALIB\_CB\_FAST\_CHECK**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_FAST_CHECK) |
| static int | [**CALIB\_CB\_FILTER\_QUADS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_FILTER_QUADS) |
| static int | [**CALIB\_CB\_NORMALIZE\_IMAGE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_NORMALIZE_IMAGE) |
| static int | [**CALIB\_CB\_SYMMETRIC\_GRID**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_CB_SYMMETRIC_GRID) |
| static int | [**CALIB\_FIX\_ASPECT\_RATIO**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_ASPECT_RATIO) |
| static int | [**CALIB\_FIX\_FOCAL\_LENGTH**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_FOCAL_LENGTH) |
| static int | [**CALIB\_FIX\_INTRINSIC**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_INTRINSIC) |
| static int | [**CALIB\_FIX\_K1**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K1) |
| static int | [**CALIB\_FIX\_K2**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K2) |
| static int | [**CALIB\_FIX\_K3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K3) |
| static int | [**CALIB\_FIX\_K4**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K4) |
| static int | [**CALIB\_FIX\_K5**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K5) |
| static int | [**CALIB\_FIX\_K6**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_K6) |
| static int | [**CALIB\_FIX\_PRINCIPAL\_POINT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_PRINCIPAL_POINT) |
| static int | [**CALIB\_FIX\_S1\_S2\_S3\_S4**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_S1_S2_S3_S4) |
| static int | [**CALIB\_FIX\_TANGENT\_DIST**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_TANGENT_DIST) |
| static int | [**CALIB\_FIX\_TAUX\_TAUY**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_FIX_TAUX_TAUY) |
| static int | [**CALIB\_HAND\_EYE\_ANDREFF**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_HAND_EYE_ANDREFF) |
| static int | [**CALIB\_HAND\_EYE\_DANIILIDIS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_HAND_EYE_DANIILIDIS) |
| static int | [**CALIB\_HAND\_EYE\_HORAUD**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_HAND_EYE_HORAUD) |
| static int | [**CALIB\_HAND\_EYE\_PARK**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_HAND_EYE_PARK) |
| static int | [**CALIB\_HAND\_EYE\_TSAI**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_HAND_EYE_TSAI) |
| static int | [**CALIB\_RATIONAL\_MODEL**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_RATIONAL_MODEL) |
| static int | [**CALIB\_SAME\_FOCAL\_LENGTH**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_SAME_FOCAL_LENGTH) |
| static int | [**CALIB\_THIN\_PRISM\_MODEL**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_THIN_PRISM_MODEL) |
| static int | [**CALIB\_TILTED\_MODEL**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_TILTED_MODEL) |
| static int | [**CALIB\_USE\_EXTRINSIC\_GUESS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_USE_EXTRINSIC_GUESS) |
| static int | [**CALIB\_USE\_INTRINSIC\_GUESS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_USE_INTRINSIC_GUESS) |
| static int | [**CALIB\_USE\_LU**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_USE_LU) |
| static int | [**CALIB\_USE\_QR**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_USE_QR) |
| static int | [**CALIB\_ZERO\_DISPARITY**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_ZERO_DISPARITY) |
| static int | [**CALIB\_ZERO\_TANGENT\_DIST**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CALIB_ZERO_TANGENT_DIST) |
| static int | [**CirclesGridFinderParameters\_ASYMMETRIC\_GRID**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CirclesGridFinderParameters_ASYMMETRIC_GRID) |
| static int | [**CirclesGridFinderParameters\_SYMMETRIC\_GRID**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CirclesGridFinderParameters_SYMMETRIC_GRID) |
| static int | [**CV\_DLS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CV_DLS) |
| static int | [**CV\_EPNP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CV_EPNP) |
| static int | [**CV\_ITERATIVE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CV_ITERATIVE) |
| static int | [**CV\_P3P**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CV_P3P) |
| static int | [**CvLevMarq\_CALC\_J**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CvLevMarq_CALC_J) |
| static int | [**CvLevMarq\_CHECK\_ERR**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CvLevMarq_CHECK_ERR) |
| static int | [**CvLevMarq\_DONE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CvLevMarq_DONE) |
| static int | [**CvLevMarq\_STARTED**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#CvLevMarq_STARTED) |
| static int | [**fisheye\_CALIB\_CHECK\_COND**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_CHECK_COND) |
| static int | [**fisheye\_CALIB\_FIX\_INTRINSIC**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_INTRINSIC) |
| static int | [**fisheye\_CALIB\_FIX\_K1**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_K1) |
| static int | [**fisheye\_CALIB\_FIX\_K2**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_K2) |
| static int | [**fisheye\_CALIB\_FIX\_K3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_K3) |
| static int | [**fisheye\_CALIB\_FIX\_K4**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_K4) |
| static int | [**fisheye\_CALIB\_FIX\_PRINCIPAL\_POINT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_PRINCIPAL_POINT) |
| static int | [**fisheye\_CALIB\_FIX\_SKEW**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_FIX_SKEW) |
| static int | [**fisheye\_CALIB\_RECOMPUTE\_EXTRINSIC**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_RECOMPUTE_EXTRINSIC) |
| static int | [**fisheye\_CALIB\_USE\_INTRINSIC\_GUESS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_USE_INTRINSIC_GUESS) |
| static int | [**fisheye\_CALIB\_ZERO\_DISPARITY**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_CALIB_ZERO_DISPARITY) |
| static int | [**FM\_7POINT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#FM_7POINT) |
| static int | [**FM\_8POINT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#FM_8POINT) |
| static int | [**FM\_LMEDS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#FM_LMEDS) |
| static int | [**FM\_RANSAC**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#FM_RANSAC) |
| static int | [**LMEDS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#LMEDS) |
| static int | [**RANSAC**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RANSAC) |
| static int | [**RHO**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RHO) |
| static int | [**SOLVEPNP\_AP3P**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_AP3P) |
| static int | [**SOLVEPNP\_DLS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_DLS) |
| static int | [**SOLVEPNP\_EPNP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_EPNP) |
| static int | [**SOLVEPNP\_IPPE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_IPPE) |
| static int | [**SOLVEPNP\_IPPE\_SQUARE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_IPPE_SQUARE) |
| static int | [**SOLVEPNP\_ITERATIVE**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_ITERATIVE) |
| static int | [**SOLVEPNP\_MAX\_COUNT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_MAX_COUNT) |
| static int | [**SOLVEPNP\_P3P**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_P3P) |
| static int | [**SOLVEPNP\_SQPNP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_SQPNP) |
| static int | [**SOLVEPNP\_UPNP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#SOLVEPNP_UPNP) |

### Constructor SummaryConstructors

| Constructor and Description |
| --- |
| [**Calib3d**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#Calib3d())() |

### Method SummaryMethods

| Modifier and Type | Method and Description |
| --- | --- |
| static double | [**calibrateCamera**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCamera(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) ) |
| static double | [**calibrateCamera**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCamera(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) ) |
| static double | [**calibrateCamera**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCamera(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) ) |
| static double | [**calibrateCameraExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCameraExtended(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors) Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern. |
| static double | [**calibrateCameraExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCameraExtended(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags) Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern. |
| static double | [**calibrateCameraExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateCameraExtended(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern. |
| static void | [**calibrateHandEye**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateHandEye(java.util.List,%20java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_target2cam, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_target2cam, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R\_cam2gripper, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t\_cam2gripper) Computes Hand-Eye calibration: \(\_{}^{g}\textrm{T}\_c\) |
| static void | [**calibrateHandEye**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrateHandEye(java.util.List,%20java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_target2cam, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_target2cam, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R\_cam2gripper, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t\_cam2gripper, int method) Computes Hand-Eye calibration: \(\_{}^{g}\textrm{T}\_c\) |
| static void | [**calibrationMatrixValues**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#calibrationMatrixValues(org.opencv.core.Mat,%20org.opencv.core.Size,%20double,%20double,%20double%5B%5D,%20double%5B%5D,%20double%5B%5D,%20org.opencv.core.Point,%20double%5B%5D))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double apertureWidth, double apertureHeight, double[] fovx, double[] fovy, double[] focalLength, [Point](http://docs.google.com/org/opencv/core/Point.html) principalPoint, double[] aspectRatio) Computes useful camera characteristics from the camera intrinsic matrix. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr2) Combines two rotation-and-shift transformations. |
| static void | [**composeRT**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#composeRT(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt2) Combines two rotation-and-shift transformations. |
| static void | [**computeCorrespondEpilines**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#computeCorrespondEpilines(org.opencv.core.Mat,%20int,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points, int whichImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) lines) For points in an image of a stereo pair, computes the corresponding epilines in the other image. |
| static void | [**convertPointsFromHomogeneous**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#convertPointsFromHomogeneous(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts points from homogeneous to Euclidean space. |
| static void | [**convertPointsToHomogeneous**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#convertPointsToHomogeneous(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts points from Euclidean to homogeneous space. |
| static void | [**correctMatches**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#correctMatches(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) newPoints1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) newPoints2) Refines coordinates of corresponding points. |
| static void | [**decomposeEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t) Decompose an essential matrix to possible rotations and translation. |
| static int | [**decomposeHomographyMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeHomographyMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20java.util.List))([Mat](http://docs.google.com/org/opencv/core/Mat.html) H, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> translations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals) Decompose a homography matrix to rotation(s), translation(s) and plane normal(s). |
| static void | [**decomposeProjectionMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeProjectionMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix. |
| static void | [**decomposeProjectionMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeProjectionMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix. |
| static void | [**decomposeProjectionMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeProjectionMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix. |
| static void | [**decomposeProjectionMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeProjectionMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixZ) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix. |
| static void | [**decomposeProjectionMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#decomposeProjectionMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixZ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eulerAngles) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix. |
| static void | [**drawChessboardCorners**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#drawChessboardCorners(org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.MatOfPoint2f,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners, boolean patternWasFound) Renders the detected chessboard corners. |
| static void | [**drawFrameAxes**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#drawFrameAxes(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20float))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, float length) Draw axes of the world/object coordinate system from pose estimation. |
| static void | [**drawFrameAxes**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#drawFrameAxes(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20float,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, float length, int thickness) Draw axes of the world/object coordinate system from pose estimation. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence) Computes an optimal affine transformation between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffine2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long,%20double,%20long))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence, long refineIters) Computes an optimal affine transformation between two 2D point sets. |
| static int | [**estimateAffine3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal affine transformation between two 3D point sets. |
| static int | [**estimateAffine3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, double ransacThreshold) Computes an optimal affine transformation between two 3D point sets. |
| static int | [**estimateAffine3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffine3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, double ransacThreshold, double confidence) Computes an optimal affine transformation between two 3D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**estimateAffinePartial2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#estimateAffinePartial2D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20long,%20double,%20long))([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence, long refineIters) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets. |
| static void | [**filterHomographyDecompByVisibleRefpoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#filterHomographyDecompByVisibleRefpoints(java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals, [Mat](http://docs.google.com/org/opencv/core/Mat.html) beforePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) afterPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) possibleSolutions) Filters homography decompositions based on additional information. |
| static void | [**filterHomographyDecompByVisibleRefpoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#filterHomographyDecompByVisibleRefpoints(java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals, [Mat](http://docs.google.com/org/opencv/core/Mat.html) beforePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) afterPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) possibleSolutions, [Mat](http://docs.google.com/org/opencv/core/Mat.html) pointsMask) Filters homography decompositions based on additional information. |
| static void | [**filterSpeckles**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#filterSpeckles(org.opencv.core.Mat,%20double,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, double newVal, int maxSpeckleSize, double maxDiff) Filters off small noise blobs (speckles) in the disparity map |
| static void | [**filterSpeckles**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#filterSpeckles(org.opencv.core.Mat,%20double,%20int,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, double newVal, int maxSpeckleSize, double maxDiff, [Mat](http://docs.google.com/org/opencv/core/Mat.html) buf) Filters off small noise blobs (speckles) in the disparity map |
| static boolean | [**find4QuadCornerSubpix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#find4QuadCornerSubpix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, [Mat](http://docs.google.com/org/opencv/core/Mat.html) corners, [Size](http://docs.google.com/org/opencv/core/Size.html) region\_size) |
| static boolean | [**findChessboardCorners**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findChessboardCorners(org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.MatOfPoint2f))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners) Finds the positions of internal corners of the chessboard. |
| static boolean | [**findChessboardCorners**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findChessboardCorners(org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.MatOfPoint2f,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners, int flags) Finds the positions of internal corners of the chessboard. |
| static boolean | [**findCirclesGrid**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findCirclesGrid(org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers) |
| static boolean | [**findCirclesGrid**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findCirclesGrid(org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers, int flags) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point,%20int,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob, double threshold) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point,%20int,%20double,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob, double threshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix) Calculates an essential matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method) Calculates an essential matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob) Calculates an essential matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob, double threshold) Calculates an essential matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findEssentialMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findEssentialMat(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob, double threshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an essential matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20double))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20double,%20int))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, int maxIters) Calculates a fundamental matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20double,%20int,%20org.opencv.core.Mat))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, int maxIters, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates a fundamental matrix from the corresponding points in two images. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findFundamentalMat**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findFundamentalMat(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20double,%20org.opencv.core.Mat))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints) Finds a perspective transformation between two planes. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method) Finds a perspective transformation between two planes. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold) Finds a perspective transformation between two planes. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20org.opencv.core.Mat))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Finds a perspective transformation between two planes. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20org.opencv.core.Mat,%20int))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int maxIters) Finds a perspective transformation between two planes. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**findHomography**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#findHomography(org.opencv.core.MatOfPoint2f,%20org.opencv.core.MatOfPoint2f,%20int,%20double,%20org.opencv.core.Mat,%20int,%20double))([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int maxIters, double confidence) Finds a perspective transformation between two planes. |
| static double | [**fisheye\_calibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_calibrate(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs) Performs camera calibaration |
| static double | [**fisheye\_calibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_calibrate(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) Performs camera calibaration |
| static double | [**fisheye\_calibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_calibrate(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Performs camera calibaration |
| static void | [**fisheye\_distortPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_distortPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Distorts 2D points using fisheye model. |
| static void | [**fisheye\_distortPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_distortPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha) Distorts 2D points using fisheye model. |
| static void | [**fisheye\_estimateNewCameraMatrixForUndistortRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_estimateNewCameraMatrixForUndistortRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P) Estimates new camera intrinsic matrix for undistortion or rectification. |
| static void | [**fisheye\_estimateNewCameraMatrixForUndistortRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_estimateNewCameraMatrixForUndistortRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance) Estimates new camera intrinsic matrix for undistortion or rectification. |
| static void | [**fisheye\_estimateNewCameraMatrixForUndistortRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_estimateNewCameraMatrixForUndistortRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size) Estimates new camera intrinsic matrix for undistortion or rectification. |
| static void | [**fisheye\_estimateNewCameraMatrixForUndistortRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_estimateNewCameraMatrixForUndistortRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Size,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size, double fov\_scale) Estimates new camera intrinsic matrix for undistortion or rectification. |
| static void | [**fisheye\_initUndistortRectifyMap**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_initUndistortRectifyMap(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20int,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, [Size](http://docs.google.com/org/opencv/core/Size.html) size, int m1type, [Mat](http://docs.google.com/org/opencv/core/Mat.html) map1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) map2) Computes undistortion and rectification maps for image transform by cv::remap(). |
| static void | [**fisheye\_projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_projectPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) |
| static void | [**fisheye\_projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_projectPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha) |
| static void | [**fisheye\_projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_projectPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian) |
| static double | [**fisheye\_stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T) Performs stereo calibration |
| static double | [**fisheye\_stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, int flags) Performs stereo calibration |
| static double | [**fisheye\_stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Performs stereo calibration |
| static void | [**fisheye\_stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags) Stereo rectification for fisheye camera model |
| static void | [**fisheye\_stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize) Stereo rectification for fisheye camera model |
| static void | [**fisheye\_stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Size,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, double balance) Stereo rectification for fisheye camera model |
| static void | [**fisheye\_stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.Size,%20double,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, double balance, double fov\_scale) Stereo rectification for fisheye camera model |
| static void | [**fisheye\_undistortImage**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortImage(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Transforms an image to compensate for fisheye lens distortion. |
| static void | [**fisheye\_undistortImage**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortImage(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Knew) Transforms an image to compensate for fisheye lens distortion. |
| static void | [**fisheye\_undistortImage**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortImage(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Knew, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size) Transforms an image to compensate for fisheye lens distortion. |
| static void | [**fisheye\_undistortPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Undistorts 2D points using fisheye model |
| static void | [**fisheye\_undistortPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R) Undistorts 2D points using fisheye model |
| static void | [**fisheye\_undistortPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#fisheye_undistortPoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P) Undistorts 2D points using fisheye model |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**getOptimalNewCameraMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#getOptimalNewCameraMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha) Returns the new camera intrinsic matrix based on the free scaling parameter. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**getOptimalNewCameraMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#getOptimalNewCameraMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20double,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize) Returns the new camera intrinsic matrix based on the free scaling parameter. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**getOptimalNewCameraMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#getOptimalNewCameraMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20double,%20org.opencv.core.Size,%20org.opencv.core.Rect))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI) Returns the new camera intrinsic matrix based on the free scaling parameter. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**getOptimalNewCameraMatrix**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#getOptimalNewCameraMatrix(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20double,%20org.opencv.core.Size,%20org.opencv.core.Rect,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI, boolean centerPrincipalPoint) Returns the new camera intrinsic matrix based on the free scaling parameter. |
| static [Rect](http://docs.google.com/org/opencv/core/Rect.html) | [**getValidDisparityROI**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#getValidDisparityROI(org.opencv.core.Rect,%20org.opencv.core.Rect,%20int,%20int,%20int))([Rect](http://docs.google.com/org/opencv/core/Rect.html) roi1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi2, int minDisparity, int numberOfDisparities, int blockSize) |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**initCameraMatrix2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#initCameraMatrix2D(java.util.List,%20java.util.List,%20org.opencv.core.Size))(java.util.List<[MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html)> objectPoints, java.util.List<[MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize) Finds an initial camera intrinsic matrix from 3D-2D point correspondences. |
| static [Mat](http://docs.google.com/org/opencv/core/Mat.html) | [**initCameraMatrix2D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#initCameraMatrix2D(java.util.List,%20java.util.List,%20org.opencv.core.Size,%20double))(java.util.List<[MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html)> objectPoints, java.util.List<[MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double aspectRatio) Finds an initial camera intrinsic matrix from 3D-2D point correspondences. |
| static void | [**matMulDeriv**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#matMulDeriv(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) A, [Mat](http://docs.google.com/org/opencv/core/Mat.html) B, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dABdA, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dABdB) Computes partial derivatives of the matrix product for each multiplied matrix. |
| static void | [**projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#projectPoints(org.opencv.core.MatOfPoint3f,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.MatOfPoint2f))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints) Projects 3D points to an image plane. |
| static void | [**projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#projectPoints(org.opencv.core.MatOfPoint3f,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian) Projects 3D points to an image plane. |
| static void | [**projectPoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#projectPoints(org.opencv.core.MatOfPoint3f,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20double))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian, double aspectRatio) Projects 3D points to an image plane. |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Point,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t) Recovers the relative camera rotation and the translation from an estimated essential matrix and the corresponding points in two images, using cheirality check. |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, [Mat](http://docs.google.com/org/opencv/core/Mat.html) triangulatedPoints) |
| static int | [**recoverPose**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#recoverPose(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Recovers the relative camera rotation and the translation from an estimated essential matrix and the corresponding points in two images, using cheirality check. |
| static float | [**rectify3Collinear**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#rectify3Collinear(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double,%20org.opencv.core.Size,%20org.opencv.core.Rect,%20org.opencv.core.Rect,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs3, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imgpt1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imgpt3, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R12, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T12, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R13, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T13, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi2, int flags) |
| static void | [**reprojectImageTo3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#reprojectImageTo3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q) Reprojects a disparity image to 3D space. |
| static void | [**reprojectImageTo3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#reprojectImageTo3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, boolean handleMissingValues) Reprojects a disparity image to 3D space. |
| static void | [**reprojectImageTo3D**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#reprojectImageTo3D(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, boolean handleMissingValues, int ddepth) Reprojects a disparity image to 3D space. |
| static void | [**Rodrigues**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#Rodrigues(org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts a rotation matrix to a rotation vector or vice versa. |
| static void | [**Rodrigues**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#Rodrigues(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian) Converts a rotation matrix to a rotation vector or vice versa. |
| static double[] | [**RQDecomp3x3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RQDecomp3x3(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ) Computes an RQ decomposition of 3x3 matrices. |
| static double[] | [**RQDecomp3x3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RQDecomp3x3(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx) Computes an RQ decomposition of 3x3 matrices. |
| static double[] | [**RQDecomp3x3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RQDecomp3x3(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qy) Computes an RQ decomposition of 3x3 matrices. |
| static double[] | [**RQDecomp3x3**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#RQDecomp3x3(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qy, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qz) Computes an RQ decomposition of 3x3 matrices. |
| static double | [**sampsonDistance**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#sampsonDistance(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) pt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) pt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F) Calculates the Sampson Distance between two points. |
| static int | [**solveP3P**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solveP3P(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) Finds an object pose from 3 3D-2D point correspondences. |
| static boolean | [**solvePnP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnP(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Finds an object pose from 3D-2D point correspondences. |
| static boolean | [**solvePnP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnP(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess) Finds an object pose from 3D-2D point correspondences. |
| static boolean | [**solvePnP**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnP(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int flags) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20boolean))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20boolean,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20boolean,%20int,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20boolean,%20int,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Finds an object pose from 3D-2D point correspondences. |
| static int | [**solvePnPGeneric**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPGeneric(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20java.util.List,%20java.util.List,%20boolean,%20int,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) reprojectionError) Finds an object pose from 3D-2D point correspondences. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int,%20float))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int,%20float,%20double))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int,%20float,%20double,%20org.opencv.core.Mat))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static boolean | [**solvePnPRansac**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRansac(org.opencv.core.MatOfPoint3f,%20org.opencv.core.MatOfPoint2f,%20org.opencv.core.Mat,%20org.opencv.core.MatOfDouble,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20boolean,%20int,%20float,%20double,%20org.opencv.core.Mat,%20int))([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int flags) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme. |
| static void | [**solvePnPRefineLM**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRefineLM(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution. |
| static void | [**solvePnPRefineLM**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRefineLM(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.TermCriteria))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution. |
| static void | [**solvePnPRefineVVS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRefineVVS(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution. |
| static void | [**solvePnPRefineVVS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRefineVVS(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.TermCriteria))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution. |
| static void | [**solvePnPRefineVVS**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#solvePnPRefineVVS(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.TermCriteria,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, double VVSlambda) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution. |
| static double | [**stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F) |
| static double | [**stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, int flags) |
| static double | [**stereoCalibrate**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrate(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) |
| static double | [**stereoCalibrateExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrateExtended(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors) Calibrates a stereo camera set up. |
| static double | [**stereoCalibrateExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrateExtended(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags) Calibrates a stereo camera set up. |
| static double | [**stereoCalibrateExtended**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoCalibrateExtended(java.util.List,%20java.util.List,%20java.util.List,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20org.opencv.core.TermCriteria))(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Calibrates a stereo camera set up. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q) Computes rectification transforms for each head of a calibrated stereo camera. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags) Computes rectification transforms for each head of a calibrated stereo camera. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha) Computes rectification transforms for each head of a calibrated stereo camera. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20org.opencv.core.Size))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize) Computes rectification transforms for each head of a calibrated stereo camera. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20org.opencv.core.Size,%20org.opencv.core.Rect))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI1) Computes rectification transforms for each head of a calibrated stereo camera. |
| static void | [**stereoRectify**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectify(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20double,%20org.opencv.core.Size,%20org.opencv.core.Rect,%20org.opencv.core.Rect))([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI2) Computes rectification transforms for each head of a calibrated stereo camera. |
| static boolean | [**stereoRectifyUncalibrated**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectifyUncalibrated(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Size](http://docs.google.com/org/opencv/core/Size.html) imgSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H2) Computes a rectification transform for an uncalibrated stereo camera. |
| static boolean | [**stereoRectifyUncalibrated**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#stereoRectifyUncalibrated(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Size,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20double))([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Size](http://docs.google.com/org/opencv/core/Size.html) imgSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H2, double threshold) Computes a rectification transform for an uncalibrated stereo camera. |
| static void | [**triangulatePoints**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#triangulatePoints(org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat,%20org.opencv.core.Mat))([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projPoints1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projPoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points4D) This function reconstructs 3-dimensional points (in homogeneous coordinates) by using their observations with a stereo camera. |
| static void | [**validateDisparity**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#validateDisparity(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cost, int minDisparity, int numberOfDisparities) |
| static void | [**validateDisparity**](http://docs.google.com/org/opencv/calib3d/Calib3d.html#validateDisparity(org.opencv.core.Mat,%20org.opencv.core.Mat,%20int,%20int,%20int))([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cost, int minDisparity, int numberOfDisparities, int disp12MaxDisp) |

### Methods inherited from class java.lang.Objectequals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

#### CALIB\_CB\_ADAPTIVE\_THRESH public static final int CALIB\_CB\_ADAPTIVE\_THRESHSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_ADAPTIVE_THRESH)

#### CALIB\_CB\_ASYMMETRIC\_GRID public static final int CALIB\_CB\_ASYMMETRIC\_GRIDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_ASYMMETRIC_GRID)

#### CALIB\_CB\_CLUSTERING public static final int CALIB\_CB\_CLUSTERINGSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_CLUSTERING)

#### CALIB\_CB\_FAST\_CHECK public static final int CALIB\_CB\_FAST\_CHECKSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_FAST_CHECK)

#### CALIB\_CB\_FILTER\_QUADS public static final int CALIB\_CB\_FILTER\_QUADSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_FILTER_QUADS)

#### CALIB\_CB\_NORMALIZE\_IMAGE public static final int CALIB\_CB\_NORMALIZE\_IMAGESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_NORMALIZE_IMAGE)

#### CALIB\_CB\_SYMMETRIC\_GRID public static final int CALIB\_CB\_SYMMETRIC\_GRIDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_CB_SYMMETRIC_GRID)

#### CALIB\_FIX\_ASPECT\_RATIO public static final int CALIB\_FIX\_ASPECT\_RATIOSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_ASPECT_RATIO)

#### CALIB\_FIX\_FOCAL\_LENGTH public static final int CALIB\_FIX\_FOCAL\_LENGTHSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_FOCAL_LENGTH)

#### CALIB\_FIX\_INTRINSIC public static final int CALIB\_FIX\_INTRINSICSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_INTRINSIC)

#### CALIB\_FIX\_K1 public static final int CALIB\_FIX\_K1See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K1)

#### CALIB\_FIX\_K2 public static final int CALIB\_FIX\_K2See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K2)

#### CALIB\_FIX\_K3 public static final int CALIB\_FIX\_K3See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K3)

#### CALIB\_FIX\_K4 public static final int CALIB\_FIX\_K4See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K4)

#### CALIB\_FIX\_K5 public static final int CALIB\_FIX\_K5See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K5)

#### CALIB\_FIX\_K6 public static final int CALIB\_FIX\_K6See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_K6)

#### CALIB\_FIX\_PRINCIPAL\_POINT public static final int CALIB\_FIX\_PRINCIPAL\_POINTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_PRINCIPAL_POINT)

#### CALIB\_FIX\_S1\_S2\_S3\_S4 public static final int CALIB\_FIX\_S1\_S2\_S3\_S4See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_S1_S2_S3_S4)

#### CALIB\_FIX\_TANGENT\_DIST public static final int CALIB\_FIX\_TANGENT\_DISTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_TANGENT_DIST)

#### CALIB\_FIX\_TAUX\_TAUY public static final int CALIB\_FIX\_TAUX\_TAUYSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_FIX_TAUX_TAUY)

#### CALIB\_HAND\_EYE\_ANDREFF public static final int CALIB\_HAND\_EYE\_ANDREFFSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_HAND_EYE_ANDREFF)

#### CALIB\_HAND\_EYE\_DANIILIDIS public static final int CALIB\_HAND\_EYE\_DANIILIDISSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_HAND_EYE_DANIILIDIS)

#### CALIB\_HAND\_EYE\_HORAUD public static final int CALIB\_HAND\_EYE\_HORAUDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_HAND_EYE_HORAUD)

#### CALIB\_HAND\_EYE\_PARK public static final int CALIB\_HAND\_EYE\_PARKSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_HAND_EYE_PARK)

#### CALIB\_HAND\_EYE\_TSAI public static final int CALIB\_HAND\_EYE\_TSAISee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_HAND_EYE_TSAI)

#### CALIB\_RATIONAL\_MODEL public static final int CALIB\_RATIONAL\_MODELSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_RATIONAL_MODEL)

#### CALIB\_SAME\_FOCAL\_LENGTH public static final int CALIB\_SAME\_FOCAL\_LENGTHSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_SAME_FOCAL_LENGTH)

#### CALIB\_THIN\_PRISM\_MODEL public static final int CALIB\_THIN\_PRISM\_MODELSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_THIN_PRISM_MODEL)

#### CALIB\_TILTED\_MODEL public static final int CALIB\_TILTED\_MODELSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_TILTED_MODEL)

#### CALIB\_USE\_EXTRINSIC\_GUESS public static final int CALIB\_USE\_EXTRINSIC\_GUESSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_USE_EXTRINSIC_GUESS)

#### CALIB\_USE\_INTRINSIC\_GUESS public static final int CALIB\_USE\_INTRINSIC\_GUESSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_USE_INTRINSIC_GUESS)

#### CALIB\_USE\_LU public static final int CALIB\_USE\_LUSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_USE_LU)

#### CALIB\_USE\_QR public static final int CALIB\_USE\_QRSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_USE_QR)

#### CALIB\_ZERO\_DISPARITY public static final int CALIB\_ZERO\_DISPARITYSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_ZERO_DISPARITY)

#### CALIB\_ZERO\_TANGENT\_DIST public static final int CALIB\_ZERO\_TANGENT\_DISTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CALIB_ZERO_TANGENT_DIST)

#### CirclesGridFinderParameters\_ASYMMETRIC\_GRID public static final int CirclesGridFinderParameters\_ASYMMETRIC\_GRIDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CirclesGridFinderParameters_ASYMMETRIC_GRID)

#### CirclesGridFinderParameters\_SYMMETRIC\_GRID public static final int CirclesGridFinderParameters\_SYMMETRIC\_GRIDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CirclesGridFinderParameters_SYMMETRIC_GRID)

#### CV\_DLS public static final int CV\_DLSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CV_DLS)

#### CV\_EPNP public static final int CV\_EPNPSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CV_EPNP)

#### CV\_ITERATIVE public static final int CV\_ITERATIVESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CV_ITERATIVE)

#### CV\_P3P public static final int CV\_P3PSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CV_P3P)

#### CvLevMarq\_CALC\_J public static final int CvLevMarq\_CALC\_JSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CvLevMarq_CALC_J)

#### CvLevMarq\_CHECK\_ERR public static final int CvLevMarq\_CHECK\_ERRSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CvLevMarq_CHECK_ERR)

#### CvLevMarq\_DONE public static final int CvLevMarq\_DONESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CvLevMarq_DONE)

#### CvLevMarq\_STARTED public static final int CvLevMarq\_STARTEDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.CvLevMarq_STARTED)

#### fisheye\_CALIB\_CHECK\_COND public static final int fisheye\_CALIB\_CHECK\_CONDSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_CHECK_COND)

#### fisheye\_CALIB\_FIX\_INTRINSIC public static final int fisheye\_CALIB\_FIX\_INTRINSICSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_INTRINSIC)

#### fisheye\_CALIB\_FIX\_K1 public static final int fisheye\_CALIB\_FIX\_K1See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_K1)

#### fisheye\_CALIB\_FIX\_K2 public static final int fisheye\_CALIB\_FIX\_K2See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_K2)

#### fisheye\_CALIB\_FIX\_K3 public static final int fisheye\_CALIB\_FIX\_K3See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_K3)

#### fisheye\_CALIB\_FIX\_K4 public static final int fisheye\_CALIB\_FIX\_K4See Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_K4)

#### fisheye\_CALIB\_FIX\_PRINCIPAL\_POINT public static final int fisheye\_CALIB\_FIX\_PRINCIPAL\_POINTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_PRINCIPAL_POINT)

#### fisheye\_CALIB\_FIX\_SKEW public static final int fisheye\_CALIB\_FIX\_SKEWSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_FIX_SKEW)

#### fisheye\_CALIB\_RECOMPUTE\_EXTRINSIC public static final int fisheye\_CALIB\_RECOMPUTE\_EXTRINSICSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_RECOMPUTE_EXTRINSIC)

#### fisheye\_CALIB\_USE\_INTRINSIC\_GUESS public static final int fisheye\_CALIB\_USE\_INTRINSIC\_GUESSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_USE_INTRINSIC_GUESS)

#### fisheye\_CALIB\_ZERO\_DISPARITY public static final int fisheye\_CALIB\_ZERO\_DISPARITYSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.fisheye_CALIB_ZERO_DISPARITY)

#### FM\_7POINT public static final int FM\_7POINTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.FM_7POINT)

#### FM\_8POINT public static final int FM\_8POINTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.FM_8POINT)

#### FM\_LMEDS public static final int FM\_LMEDSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.FM_LMEDS)

#### FM\_RANSAC public static final int FM\_RANSACSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.FM_RANSAC)

#### LMEDS public static final int LMEDSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.LMEDS)

#### RANSAC public static final int RANSACSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.RANSAC)

#### RHO public static final int RHOSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.RHO)

#### SOLVEPNP\_AP3P public static final int SOLVEPNP\_AP3PSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_AP3P)

#### SOLVEPNP\_DLS public static final int SOLVEPNP\_DLSSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_DLS)

#### SOLVEPNP\_EPNP public static final int SOLVEPNP\_EPNPSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_EPNP)

#### SOLVEPNP\_IPPE public static final int SOLVEPNP\_IPPESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_IPPE)

#### SOLVEPNP\_IPPE\_SQUARE public static final int SOLVEPNP\_IPPE\_SQUARESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_IPPE_SQUARE)

#### SOLVEPNP\_ITERATIVE public static final int SOLVEPNP\_ITERATIVESee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_ITERATIVE)

#### SOLVEPNP\_MAX\_COUNT public static final int SOLVEPNP\_MAX\_COUNTSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_MAX_COUNT)

#### SOLVEPNP\_P3P public static final int SOLVEPNP\_P3PSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_P3P)

#### SOLVEPNP\_SQPNP public static final int SOLVEPNP\_SQPNPSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_SQPNP)

#### SOLVEPNP\_UPNP public static final int SOLVEPNP\_UPNPSee Also:[Constant Field Values](http://docs.google.com/constant-values.html#org.opencv.calib3d.Calib3d.SOLVEPNP_UPNP)

### Constructor Detail

#### Calib3d public Calib3d()

### Method Detail

#### calibrateCamera public static double calibrateCamera(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) )Parameters:objectPoints - automatically generatedimagePoints - automatically generatedimageSize - automatically generatedcameraMatrix - automatically generateddistCoeffs - automatically generatedrvecs - automatically generatedtvecs - automatically generated Returns:automatically generated

#### calibrateCamera public static double calibrateCamera(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) )Parameters:objectPoints - automatically generatedimagePoints - automatically generatedimageSize - automatically generatedcameraMatrix - automatically generateddistCoeffs - automatically generatedrvecs - automatically generatedtvecs - automatically generatedflags - automatically generated Returns:automatically generated

#### calibrateCamera public static double calibrateCamera(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) double calibrateCamera( InputArrayOfArrays objectPoints, InputArrayOfArrays imagePoints, Size imageSize, InputOutputArray cameraMatrix, InputOutputArray distCoeffs, OutputArrayOfArrays rvecs, OutputArrayOfArrays tvecs, OutputArray stdDeviations, OutputArray perViewErrors, int flags = 0, TermCriteria criteria = TermCriteria( TermCriteria::COUNT + TermCriteria::EPS, 30, DBL\_EPSILON) )Parameters:objectPoints - automatically generatedimagePoints - automatically generatedimageSize - automatically generatedcameraMatrix - automatically generateddistCoeffs - automatically generatedrvecs - automatically generatedtvecs - automatically generatedflags - automatically generatedcriteria - automatically generated Returns:automatically generated

#### calibrateCameraExtended public static double calibrateCameraExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors) 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>>). The outer vector contains as 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. Although, it is possible to use 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 calibration pattern's XY coordinate 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 concatenated together.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 objectPoints.size(), and 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 concatenated together.imageSize - Size of the image used only to initialize the camera intrinsic matrix.cameraMatrix - Input/output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: CALIB\_USE\_INTRINSIC\_GUESS and/or REF: CALIB\_FIX\_ASPECT\_RATIO are specified, some or all of fx, fy, cx, cy must be initialized before calling the function.distCoeffs - Input/output vector of distortion coefficients \(\distcoeffs\).rvecs - Output vector of rotation vectors (REF: Rodrigues ) estimated for each pattern view (e.g. std::vector<cv::Mat>>). That is, each i-th rotation vector together with the corresponding i-th translation vector (see the next output parameter description) brings the calibration pattern from the object coordinate space (in which object points are specified) to the camera 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 camera coordinate space. Due 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: \((f\_x, f\_y, c\_x, c\_y, k\_1, k\_2, p\_1, p\_2, k\_3, k\_4, k\_5, k\_6 , s\_1, s\_2, s\_3, s\_4, \tau\_x, \tau\_y)\) If one of 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: \((R\_0, T\_0, \dotsc , R\_{M - 1}, T\_{M - 1})\) where M is the number of pattern views. \(R\_i, T\_i\) are concatenated 1x3 vectors.perViewErrors - Output vector of the RMS re-projection error estimated for each pattern view.

* + - * REF: CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( 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 this function just to estimate extrinsic parameters. Use solvePnP instead.
      * REF: CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: CALIB\_USE\_INTRINSIC\_GUESS is set too.
      * REF: 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 REF: 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.
      * REF: CALIB\_ZERO\_TANGENT\_DIST Tangential distortion coefficients \((p\_1, p\_2)\) are set to zeros and stay zero.
      * REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 The corresponding radial distortion coefficient is not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_RATIONAL\_MODEL Coefficients k4, k5, and k6 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.

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 CITE: Zhang2000 and CITE: BouguetMCT . The coordinates of 3D object points and their corresponding 2D 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 calibration pattern, and OpenCV has built-in support for a chessboard as a calibration rig (see REF: findChessboardCorners). Currently, initialization of intrinsic parameters (when REF: CALIB\_USE\_INTRINSIC\_GUESS is not set) is only implemented 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 initially unless some of 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 imagePoints and the projected (using 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 REF: findChessboardCorners for calibration, and REF: calibrateCamera returns bad values (zero distortion coefficients, \(c\_x\) and \(c\_y\) very far from the image center, and/or 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 REF: findChessboardCorners. SEE: findChessboardCorners, solvePnP, initCameraMatrix2D, stereoCalibrate, undistort

#### calibrateCameraExtended public static double calibrateCameraExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags) 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>>). The outer vector contains as 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. Although, it is possible to use 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 calibration pattern's XY coordinate 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 concatenated together.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 objectPoints.size(), and 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 concatenated together.imageSize - Size of the image used only to initialize the camera intrinsic matrix.cameraMatrix - Input/output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: CALIB\_USE\_INTRINSIC\_GUESS and/or REF: CALIB\_FIX\_ASPECT\_RATIO are specified, some or all of fx, fy, cx, cy must be initialized before calling the function.distCoeffs - Input/output vector of distortion coefficients \(\distcoeffs\).rvecs - Output vector of rotation vectors (REF: Rodrigues ) estimated for each pattern view (e.g. std::vector<cv::Mat>>). That is, each i-th rotation vector together with the corresponding i-th translation vector (see the next output parameter description) brings the calibration pattern from the object coordinate space (in which object points are specified) to the camera 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 camera coordinate space. Due 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: \((f\_x, f\_y, c\_x, c\_y, k\_1, k\_2, p\_1, p\_2, k\_3, k\_4, k\_5, k\_6 , s\_1, s\_2, s\_3, s\_4, \tau\_x, \tau\_y)\) If one of 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: \((R\_0, T\_0, \dotsc , R\_{M - 1}, T\_{M - 1})\) where M is the number of pattern views. \(R\_i, T\_i\) are 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:

* + - * REF: CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( 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 this function just to estimate extrinsic parameters. Use solvePnP instead.
      * REF: CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: CALIB\_USE\_INTRINSIC\_GUESS is set too.
      * REF: 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 REF: 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.
      * REF: CALIB\_ZERO\_TANGENT\_DIST Tangential distortion coefficients \((p\_1, p\_2)\) are set to zeros and stay zero.
      * REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 The corresponding radial distortion coefficient is not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_RATIONAL\_MODEL Coefficients k4, k5, and k6 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.

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 CITE: Zhang2000 and CITE: BouguetMCT . The coordinates of 3D object points and their corresponding 2D 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 calibration pattern, and OpenCV has built-in support for a chessboard as a calibration rig (see REF: findChessboardCorners). Currently, initialization of intrinsic parameters (when REF: CALIB\_USE\_INTRINSIC\_GUESS is not set) is only implemented 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 initially unless some of 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 imagePoints and the projected (using 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 REF: findChessboardCorners for calibration, and REF: calibrateCamera returns bad values (zero distortion coefficients, \(c\_x\) and \(c\_y\) very far from the image center, and/or 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 REF: findChessboardCorners. SEE: findChessboardCorners, solvePnP, initCameraMatrix2D, stereoCalibrate, undistort

#### calibrateCameraExtended public static double calibrateCameraExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsIntrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) stdDeviationsExtrinsics, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) 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>>). The outer vector contains as 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. Although, it is possible to use 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 calibration pattern's XY coordinate 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 concatenated together.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 objectPoints.size(), and 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 concatenated together.imageSize - Size of the image used only to initialize the camera intrinsic matrix.cameraMatrix - Input/output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: CALIB\_USE\_INTRINSIC\_GUESS and/or REF: CALIB\_FIX\_ASPECT\_RATIO are specified, some or all of fx, fy, cx, cy must be initialized before calling the function.distCoeffs - Input/output vector of distortion coefficients \(\distcoeffs\).rvecs - Output vector of rotation vectors (REF: Rodrigues ) estimated for each pattern view (e.g. std::vector<cv::Mat>>). That is, each i-th rotation vector together with the corresponding i-th translation vector (see the next output parameter description) brings the calibration pattern from the object coordinate space (in which object points are specified) to the camera 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 camera coordinate space. Due 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: \((f\_x, f\_y, c\_x, c\_y, k\_1, k\_2, p\_1, p\_2, k\_3, k\_4, k\_5, k\_6 , s\_1, s\_2, s\_3, s\_4, \tau\_x, \tau\_y)\) If one of 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: \((R\_0, T\_0, \dotsc , R\_{M - 1}, T\_{M - 1})\) where M is the number of pattern views. \(R\_i, T\_i\) are 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:

* + - * REF: CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( 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 this function just to estimate extrinsic parameters. Use solvePnP instead.
      * REF: CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: CALIB\_USE\_INTRINSIC\_GUESS is set too.
      * REF: 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 REF: 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.
      * REF: CALIB\_ZERO\_TANGENT\_DIST Tangential distortion coefficients \((p\_1, p\_2)\) are set to zeros and stay zero.
      * REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 The corresponding radial distortion coefficient is not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_RATIONAL\_MODEL Coefficients k4, k5, and k6 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
      * REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
      * REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the 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 CITE: Zhang2000 and CITE: BouguetMCT . The coordinates of 3D object points and their corresponding 2D 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 calibration pattern, and OpenCV has built-in support for a chessboard as a calibration rig (see REF: findChessboardCorners). Currently, initialization of intrinsic parameters (when REF: CALIB\_USE\_INTRINSIC\_GUESS is not set) is only implemented 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 initially unless some of 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 imagePoints and the projected (using 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 REF: findChessboardCorners for calibration, and REF: calibrateCamera returns bad values (zero distortion coefficients, \(c\_x\) and \(c\_y\) very far from the image center, and/or 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 REF: findChessboardCorners. SEE: findChessboardCorners, solvePnP, initCameraMatrix2D, stereoCalibrate, undistort

#### calibrateHandEye public static void calibrateHandEye(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_target2cam, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_target2cam, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R\_cam2gripper, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t\_cam2gripper) Computes Hand-Eye calibration: \(\_{}^{g}\textrm{T}\_c\)Parameters:R\_gripper2base - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the gripper frame to the robot base frame (\(\_{}^{b}\textrm{T}\_g\)). This is a vector (vector<Mat>) that contains the rotation, (3x3) rotation matrices or (3x1) rotation vectors, for all the transformations from gripper frame to robot base frame.t\_gripper2base - Translation part extracted from the homogeneous matrix that transforms a point expressed in the gripper frame to the robot base frame (\(\_{}^{b}\textrm{T}\_g\)). This is a vector (vector<Mat>) that contains the (3x1) translation vectors for all the transformations from gripper frame to robot base frame.R\_target2cam - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the target frame to the camera frame (\(\_{}^{c}\textrm{T}\_t\)). This is a vector (vector<Mat>) that contains the rotation, (3x3) rotation matrices or (3x1) rotation vectors, for all the transformations from calibration target frame to camera frame.t\_target2cam - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the target frame to the camera frame (\(\_{}^{c}\textrm{T}\_t\)). This is a vector (vector<Mat>) that contains the (3x1) translation vectors for all the transformations from calibration target frame to camera frame.R\_cam2gripper - Estimated (3x3) rotation part extracted from the homogeneous matrix that transforms a point expressed in the camera frame to the gripper frame (\(\_{}^{g}\textrm{T}\_c\)).t\_cam2gripper - Estimated (3x1) translation part extracted from the homogeneous matrix that transforms a point expressed in the camera frame to the gripper frame (\(\_{}^{g}\textrm{T}\_c\)). The function performs the Hand-Eye calibration using various methods. One approach consists in estimating the rotation then the translation (separable solutions) and the following methods are implemented:

* + - R. Tsai, R. Lenz A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/EyeCalibration \cite Tsai89
    - F. Park, B. Martin Robot Sensor Calibration: Solving AX = XB on the Euclidean Group \cite Park94
    - R. Horaud, F. Dornaika Hand-Eye Calibration \cite Horaud95

Another approach consists in estimating simultaneously the rotation and the translation (simultaneous solutions), with the following implemented method:

* + - N. Andreff, R. Horaud, B. Espiau On-line Hand-Eye Calibration \cite Andreff99
    - K. Daniilidis Hand-Eye Calibration Using Dual Quaternions \cite Daniilidis98

The following picture describes the Hand-Eye calibration problem where the transformation between a camera ("eye") mounted on a robot gripper ("hand") has to be estimated. ![](pics/hand-eye\_figure.png) The calibration procedure is the following:

* + - a static calibration pattern is used to estimate the transformation between the target frame and the camera frame
    - the robot gripper is moved in order to acquire several poses
    - for each pose, the homogeneous transformation between the gripper frame and the robot base frame is recorded using for instance the robot kinematics \( \begin{bmatrix} X\_b\\ Y\_b\\ Z\_b\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{b}\textrm{R}\_g & \_{}^{b}\textrm{t}\_g \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_g\\ Y\_g\\ Z\_g\\ 1 \end{bmatrix} \)
    - for each pose, the homogeneous transformation between the calibration target frame and the camera frame is recorded using for instance a pose estimation method (PnP) from 2D-3D point correspondences \( \begin{bmatrix} X\_c\\ Y\_c\\ Z\_c\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{c}\textrm{R}\_t & \_{}^{c}\textrm{t}\_t \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_t\\ Y\_t\\ Z\_t\\ 1 \end{bmatrix} \)

The Hand-Eye calibration procedure returns the following homogeneous transformation \( \begin{bmatrix} X\_g\\ Y\_g\\ Z\_g\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{g}\textrm{R}\_c & \_{}^{g}\textrm{t}\_c \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_c\\ Y\_c\\ Z\_c\\ 1 \end{bmatrix} \) This problem is also known as solving the \(\mathbf{A}\mathbf{X}=\mathbf{X}\mathbf{B}\) equation: \( \begin{align\*} ^{b}{\textrm{T}\_g}^{(1)} \hspace{0.2em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(1)} &= \hspace{0.1em} ^{b}{\textrm{T}\_g}^{(2)} \hspace{0.2em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(2)} \\ (^{b}{\textrm{T}\_g}^{(2)})^{-1} \hspace{0.2em} ^{b}{\textrm{T}\_g}^{(1)} \hspace{0.2em} ^{g}\textrm{T}\_c &= \hspace{0.1em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(2)} (^{c}{\textrm{T}\_t}^{(1)})^{-1} \\ \textrm{A}\_i \textrm{X} &= \textrm{X} \textrm{B}\_i \\ \end{align\*} \) \note Additional information can be found on this [website](http://campar.in.tum.de/Chair/HandEyeCalibration). \note A minimum of 2 motions with non parallel rotation axes are necessary to determine the hand-eye transformation. So at least 3 different poses are required, but it is strongly recommended to use many more poses.

#### calibrateHandEye public static void calibrateHandEye(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_gripper2base, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> R\_target2cam, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> t\_target2cam, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R\_cam2gripper, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t\_cam2gripper, int method) Computes Hand-Eye calibration: \(\_{}^{g}\textrm{T}\_c\)Parameters:R\_gripper2base - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the gripper frame to the robot base frame (\(\_{}^{b}\textrm{T}\_g\)). This is a vector (vector<Mat>) that contains the rotation, (3x3) rotation matrices or (3x1) rotation vectors, for all the transformations from gripper frame to robot base frame.t\_gripper2base - Translation part extracted from the homogeneous matrix that transforms a point expressed in the gripper frame to the robot base frame (\(\_{}^{b}\textrm{T}\_g\)). This is a vector (vector<Mat>) that contains the (3x1) translation vectors for all the transformations from gripper frame to robot base frame.R\_target2cam - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the target frame to the camera frame (\(\_{}^{c}\textrm{T}\_t\)). This is a vector (vector<Mat>) that contains the rotation, (3x3) rotation matrices or (3x1) rotation vectors, for all the transformations from calibration target frame to camera frame.t\_target2cam - Rotation part extracted from the homogeneous matrix that transforms a point expressed in the target frame to the camera frame (\(\_{}^{c}\textrm{T}\_t\)). This is a vector (vector<Mat>) that contains the (3x1) translation vectors for all the transformations from calibration target frame to camera frame.R\_cam2gripper - Estimated (3x3) rotation part extracted from the homogeneous matrix that transforms a point expressed in the camera frame to the gripper frame (\(\_{}^{g}\textrm{T}\_c\)).t\_cam2gripper - Estimated (3x1) translation part extracted from the homogeneous matrix that transforms a point expressed in the camera frame to the gripper frame (\(\_{}^{g}\textrm{T}\_c\)).method - One of the implemented Hand-Eye calibration method, see cv::HandEyeCalibrationMethod The function performs the Hand-Eye calibration using various methods. One approach consists in estimating the rotation then the translation (separable solutions) and the following methods are implemented:

* + - R. Tsai, R. Lenz A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/EyeCalibration \cite Tsai89
    - F. Park, B. Martin Robot Sensor Calibration: Solving AX = XB on the Euclidean Group \cite Park94
    - R. Horaud, F. Dornaika Hand-Eye Calibration \cite Horaud95

Another approach consists in estimating simultaneously the rotation and the translation (simultaneous solutions), with the following implemented method:

* + - N. Andreff, R. Horaud, B. Espiau On-line Hand-Eye Calibration \cite Andreff99
    - K. Daniilidis Hand-Eye Calibration Using Dual Quaternions \cite Daniilidis98

The following picture describes the Hand-Eye calibration problem where the transformation between a camera ("eye") mounted on a robot gripper ("hand") has to be estimated. ![](pics/hand-eye\_figure.png) The calibration procedure is the following:

* + - a static calibration pattern is used to estimate the transformation between the target frame and the camera frame
    - the robot gripper is moved in order to acquire several poses
    - for each pose, the homogeneous transformation between the gripper frame and the robot base frame is recorded using for instance the robot kinematics \( \begin{bmatrix} X\_b\\ Y\_b\\ Z\_b\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{b}\textrm{R}\_g & \_{}^{b}\textrm{t}\_g \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_g\\ Y\_g\\ Z\_g\\ 1 \end{bmatrix} \)
    - for each pose, the homogeneous transformation between the calibration target frame and the camera frame is recorded using for instance a pose estimation method (PnP) from 2D-3D point correspondences \( \begin{bmatrix} X\_c\\ Y\_c\\ Z\_c\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{c}\textrm{R}\_t & \_{}^{c}\textrm{t}\_t \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_t\\ Y\_t\\ Z\_t\\ 1 \end{bmatrix} \)

The Hand-Eye calibration procedure returns the following homogeneous transformation \( \begin{bmatrix} X\_g\\ Y\_g\\ Z\_g\\ 1 \end{bmatrix} = \begin{bmatrix} \_{}^{g}\textrm{R}\_c & \_{}^{g}\textrm{t}\_c \\ 0\_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} X\_c\\ Y\_c\\ Z\_c\\ 1 \end{bmatrix} \) This problem is also known as solving the \(\mathbf{A}\mathbf{X}=\mathbf{X}\mathbf{B}\) equation: \( \begin{align\*} ^{b}{\textrm{T}\_g}^{(1)} \hspace{0.2em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(1)} &= \hspace{0.1em} ^{b}{\textrm{T}\_g}^{(2)} \hspace{0.2em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(2)} \\ (^{b}{\textrm{T}\_g}^{(2)})^{-1} \hspace{0.2em} ^{b}{\textrm{T}\_g}^{(1)} \hspace{0.2em} ^{g}\textrm{T}\_c &= \hspace{0.1em} ^{g}\textrm{T}\_c \hspace{0.2em} ^{c}{\textrm{T}\_t}^{(2)} (^{c}{\textrm{T}\_t}^{(1)})^{-1} \\ \textrm{A}\_i \textrm{X} &= \textrm{X} \textrm{B}\_i \\ \end{align\*} \) \note Additional information can be found on this [website](http://campar.in.tum.de/Chair/HandEyeCalibration). \note A minimum of 2 motions with non parallel rotation axes are necessary to determine the hand-eye transformation. So at least 3 different poses are required, but it is strongly recommended to use many more poses.

#### calibrationMatrixValues public static void calibrationMatrixValues([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double apertureWidth, double apertureHeight, double[] fovx, double[] fovy, double[] focalLength, [Point](http://docs.google.com/org/opencv/core/Point.html) principalPoint, double[] aspectRatio) Computes useful camera characteristics from the camera intrinsic matrix.Parameters:cameraMatrix - Input camera intrinsic matrix that can be estimated by calibrateCamera or stereoCalibrate .imageSize - Input image size in pixels.apertureWidth - Physical width in mm of the sensor.apertureHeight - Physical height in mm of the sensor.fovx - Output field of view in degrees along the horizontal sensor axis.fovy - Output field of view in degrees along the vertical sensor axis.focalLength - Focal length of the lens in mm.principalPoint - Principal point in mm.aspectRatio - \(f\_y/f\_x\) The function computes various useful camera characteristics from the previously estimated camera matrix. **Note:** Do keep in mind that the unity measure 'mm' stands for whatever unit of measure one chooses for the chessboard pitch (it can thus be any value).

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition. The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2dr3dt2 - Optional output derivative of rvec3 with regard to tvec2 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2dr3dt2 - Optional output derivative of rvec3 with regard to tvec2dt3dr1 - Optional output derivative of tvec3 with regard to rvec1 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2dr3dt2 - Optional output derivative of rvec3 with regard to tvec2dt3dr1 - Optional output derivative of tvec3 with regard to rvec1dt3dt1 - Optional output derivative of tvec3 with regard to tvec1 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr2) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2dr3dt2 - Optional output derivative of rvec3 with regard to tvec2dt3dr1 - Optional output derivative of tvec3 with regard to rvec1dt3dt1 - Optional output derivative of tvec3 with regard to tvec1dt3dr2 - Optional output derivative of tvec3 with regard to rvec2 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### composeRT public static void composeRT([Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dr3dt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dt3dt2) Combines two rotation-and-shift transformations.Parameters:rvec1 - First rotation vector.tvec1 - First translation vector.rvec2 - Second rotation vector.tvec2 - Second translation vector.rvec3 - Output rotation vector of the superposition.tvec3 - Output translation vector of the superposition.dr3dr1 - Optional output derivative of rvec3 with regard to rvec1dr3dt1 - Optional output derivative of rvec3 with regard to tvec1dr3dr2 - Optional output derivative of rvec3 with regard to rvec2dr3dt2 - Optional output derivative of rvec3 with regard to tvec2dt3dr1 - Optional output derivative of tvec3 with regard to rvec1dt3dt1 - Optional output derivative of tvec3 with regard to tvec1dt3dr2 - Optional output derivative of tvec3 with regard to rvec2dt3dt2 - Optional output derivative of tvec3 with regard to tvec2 The functions compute: \(\begin{array}{l} \texttt{rvec3} = \mathrm{rodrigues} ^{-1} \left ( \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \mathrm{rodrigues} ( \texttt{rvec1} ) \right ) \\ \texttt{tvec3} = \mathrm{rodrigues} ( \texttt{rvec2} ) \cdot \texttt{tvec1} + \texttt{tvec2} \end{array} ,\) where \(\mathrm{rodrigues}\) denotes a rotation vector to a rotation matrix transformation, and \(\mathrm{rodrigues}^{-1}\) denotes the inverse transformation. See Rodrigues for details. Also, the functions can compute the derivatives of the output vectors with regards to the input vectors (see matMulDeriv ). The functions are used inside stereoCalibrate but can also be used in your own code where Levenberg-Marquardt or another gradient-based solver is used to optimize a function that contains a matrix multiplication.

#### computeCorrespondEpilines public static void computeCorrespondEpilines([Mat](http://docs.google.com/org/opencv/core/Mat.html) points, int whichImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) lines) For points in an image of a stereo pair, computes the corresponding epilines in the other image.Parameters:points - Input points. \(N \times 1\) or \(1 \times N\) matrix of type CV\_32FC2 or vector<Point2f> .whichImage - Index of the image (1 or 2) that contains the points .F - Fundamental matrix that can be estimated using findFundamentalMat or stereoRectify .lines - Output vector of the epipolar lines corresponding to the points in the other image. Each line \(ax + by + c=0\) is encoded by 3 numbers \((a, b, c)\) . For every point in one of the two images of a stereo pair, the function finds the equation of the corresponding epipolar line in the other image. From the fundamental matrix definition (see findFundamentalMat ), line \(l^{(2)}\_i\) in the second image for the point \(p^{(1)}\_i\) in the first image (when whichImage=1 ) is computed as: \(l^{(2)}\_i = F p^{(1)}\_i\) And vice versa, when whichImage=2, \(l^{(1)}\_i\) is computed from \(p^{(2)}\_i\) as: \(l^{(1)}\_i = F^T p^{(2)}\_i\) Line coefficients are defined up to a scale. They are normalized so that \(a\_i^2+b\_i^2=1\) .

#### convertPointsFromHomogeneous public static void convertPointsFromHomogeneous([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts points from homogeneous to Euclidean space.Parameters:src - Input vector of N-dimensional points.dst - Output vector of N-1-dimensional points. The function converts points homogeneous to Euclidean space using perspective projection. That is, each point (x1, x2, ... x(n-1), xn) is converted to (x1/xn, x2/xn, ..., x(n-1)/xn). When xn=0, the output point coordinates will be (0,0,0,...).

#### convertPointsToHomogeneous public static void convertPointsToHomogeneous([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts points from Euclidean to homogeneous space.Parameters:src - Input vector of N-dimensional points.dst - Output vector of N+1-dimensional points. The function converts points from Euclidean to homogeneous space by appending 1's to the tuple of point coordinates. That is, each point (x1, x2, ..., xn) is converted to (x1, x2, ..., xn, 1).

#### correctMatches public static void correctMatches([Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) newPoints1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) newPoints2) Refines coordinates of corresponding points.Parameters:F - 3x3 fundamental matrix.points1 - 1xN array containing the first set of points.points2 - 1xN array containing the second set of points.newPoints1 - The optimized points1.newPoints2 - The optimized points2. The function implements the Optimal Triangulation Method (see Multiple View Geometry for details). For each given point correspondence points1[i] <-> points2[i], and a fundamental matrix F, it computes the corrected correspondences newPoints1[i] <-> newPoints2[i] that minimize the geometric error \(d(points1[i], newPoints1[i])^2 + d(points2[i],newPoints2[i])^2\) (where \(d(a,b)\) is the geometric distance between points \(a\) and \(b\) ) subject to the epipolar constraint \(newPoints2^T \* F \* newPoints1 = 0\) .

#### decomposeEssentialMat public static void decomposeEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t) Decompose an essential matrix to possible rotations and translation.Parameters:E - The input essential matrix.R1 - One possible rotation matrix.R2 - Another possible rotation matrix.t - One possible translation. This function decomposes the essential matrix E using svd decomposition CITE: HartleyZ00. In general, four possible poses exist for the decomposition of E. They are \([R\_1, t]\), \([R\_1, -t]\), \([R\_2, t]\), \([R\_2, -t]\). If E gives the epipolar constraint \([p\_2; 1]^T A^{-T} E A^{-1} [p\_1; 1] = 0\) between the image points \(p\_1\) in the first image and \(p\_2\) in second image, then any of the tuples \([R\_1, t]\), \([R\_1, -t]\), \([R\_2, t]\), \([R\_2, -t]\) is a change of basis from the first camera's coordinate system to the second camera's coordinate system. However, by decomposing E, one can only get the direction of the translation. For this reason, the translation t is returned with unit length.

#### decomposeHomographyMat public static int decomposeHomographyMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) H, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> translations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals) Decompose a homography matrix to rotation(s), translation(s) and plane normal(s).Parameters:H - The input homography matrix between two images.K - The input camera intrinsic matrix.rotations - Array of rotation matrices.translations - Array of translation matrices.normals - Array of plane normal matrices. This function extracts relative camera motion between two views of a planar object and returns up to four mathematical solution tuples of rotation, translation, and plane normal. The decomposition of the homography matrix H is described in detail in CITE: Malis. If the homography H, induced by the plane, gives the constraint \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) on the source image points \(p\_i\) and the destination image points \(p'\_i\), then the tuple of rotations[k] and translations[k] is a change of basis from the source camera's coordinate system to the destination camera's coordinate system. However, by decomposing H, one can only get the translation normalized by the (typically unknown) depth of the scene, i.e. its direction but with normalized length. If point correspondences are available, at least two solutions may further be invalidated, by applying positive depth constraint, i.e. all points must be in front of the camera. Returns:automatically generated

#### decomposeProjectionMatrix public static void decomposeProjectionMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.Parameters:projMatrix - 3x4 input projection matrix P.cameraMatrix - Output 3x3 camera intrinsic matrix \(\cameramatrix{A}\).rotMatrix - Output 3x3 external rotation matrix R.transVect - Output 4x1 translation vector T. degrees. The function computes a decomposition of a projection matrix into a calibration and a rotation matrix and the position of a camera. It optionally returns three rotation matrices, one for each axis, and three Euler angles that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. The function is based on RQDecomp3x3 .

#### decomposeProjectionMatrix public static void decomposeProjectionMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.Parameters:projMatrix - 3x4 input projection matrix P.cameraMatrix - Output 3x3 camera intrinsic matrix \(\cameramatrix{A}\).rotMatrix - Output 3x3 external rotation matrix R.transVect - Output 4x1 translation vector T.rotMatrixX - Optional 3x3 rotation matrix around x-axis. degrees. The function computes a decomposition of a projection matrix into a calibration and a rotation matrix and the position of a camera. It optionally returns three rotation matrices, one for each axis, and three Euler angles that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. The function is based on RQDecomp3x3 .

#### decomposeProjectionMatrix public static void decomposeProjectionMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.Parameters:projMatrix - 3x4 input projection matrix P.cameraMatrix - Output 3x3 camera intrinsic matrix \(\cameramatrix{A}\).rotMatrix - Output 3x3 external rotation matrix R.transVect - Output 4x1 translation vector T.rotMatrixX - Optional 3x3 rotation matrix around x-axis.rotMatrixY - Optional 3x3 rotation matrix around y-axis. degrees. The function computes a decomposition of a projection matrix into a calibration and a rotation matrix and the position of a camera. It optionally returns three rotation matrices, one for each axis, and three Euler angles that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. The function is based on RQDecomp3x3 .

#### decomposeProjectionMatrix public static void decomposeProjectionMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixZ) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.Parameters:projMatrix - 3x4 input projection matrix P.cameraMatrix - Output 3x3 camera intrinsic matrix \(\cameramatrix{A}\).rotMatrix - Output 3x3 external rotation matrix R.transVect - Output 4x1 translation vector T.rotMatrixX - Optional 3x3 rotation matrix around x-axis.rotMatrixY - Optional 3x3 rotation matrix around y-axis.rotMatrixZ - Optional 3x3 rotation matrix around z-axis. degrees. The function computes a decomposition of a projection matrix into a calibration and a rotation matrix and the position of a camera. It optionally returns three rotation matrices, one for each axis, and three Euler angles that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. The function is based on RQDecomp3x3 .

#### decomposeProjectionMatrix public static void decomposeProjectionMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) transVect, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixX, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixY, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rotMatrixZ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) eulerAngles) Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.Parameters:projMatrix - 3x4 input projection matrix P.cameraMatrix - Output 3x3 camera intrinsic matrix \(\cameramatrix{A}\).rotMatrix - Output 3x3 external rotation matrix R.transVect - Output 4x1 translation vector T.rotMatrixX - Optional 3x3 rotation matrix around x-axis.rotMatrixY - Optional 3x3 rotation matrix around y-axis.rotMatrixZ - Optional 3x3 rotation matrix around z-axis.eulerAngles - Optional three-element vector containing three Euler angles of rotation in degrees. The function computes a decomposition of a projection matrix into a calibration and a rotation matrix and the position of a camera. It optionally returns three rotation matrices, one for each axis, and three Euler angles that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. The function is based on RQDecomp3x3 .

#### drawChessboardCorners public static void drawChessboardCorners([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners, boolean patternWasFound) Renders the detected chessboard corners.Parameters:image - Destination image. It must be an 8-bit color image.patternSize - Number of inner corners per a chessboard row and column (patternSize = cv::Size(points\_per\_row,points\_per\_column)).corners - Array of detected corners, the output of findChessboardCorners.patternWasFound - Parameter indicating whether the complete board was found or not. The return value of findChessboardCorners should be passed here. The function draws individual chessboard corners detected either as red circles if the board was not found, or as colored corners connected with lines if the board was found.

#### drawFrameAxes public static void drawFrameAxes([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, float length) Draw axes of the world/object coordinate system from pose estimation. SEE: solvePnPParameters:image - Input/output image. It must have 1 or 3 channels. The number of channels is not altered.cameraMatrix - Input 3x3 floating-point matrix of camera intrinsic parameters. \(\cameramatrix{A}\)distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is empty, the zero distortion coefficients are assumed.rvec - Rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Translation vector.length - Length of the painted axes in the same unit than tvec (usually in meters). This function draws the axes of the world/object coordinate system w.r.t. to the camera frame. OX is drawn in red, OY in green and OZ in blue.

#### drawFrameAxes public static void drawFrameAxes([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, float length, int thickness) Draw axes of the world/object coordinate system from pose estimation. SEE: solvePnPParameters:image - Input/output image. It must have 1 or 3 channels. The number of channels is not altered.cameraMatrix - Input 3x3 floating-point matrix of camera intrinsic parameters. \(\cameramatrix{A}\)distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is empty, the zero distortion coefficients are assumed.rvec - Rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Translation vector.length - Length of the painted axes in the same unit than tvec (usually in meters).thickness - Line thickness of the painted axes. This function draws the axes of the world/object coordinate system w.r.t. to the camera frame. OX is drawn in red, OY in green and OZ in blue.

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).

* + - REF: RANSAC - RANSAC-based robust method
    - REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).

* + - REF: RANSAC - RANSAC-based robust method
    - REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).method - Robust method used to compute transformation. The following methods are possible:

* + - REF: RANSAC - RANSAC-based robust method
    - REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).method - Robust method used to compute transformation. The following methods are possible:

* + - REF: RANSAC - RANSAC-based robust method
    - REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations.confidence - Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffine2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence, long refineIters) Computes an optimal affine transformation between two 2D point sets. It computes \( \begin{bmatrix} x\\ y\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12}\\ a\_{21} & a\_{22}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ \end{bmatrix} \)Parameters:from - First input 2D point set containing \((X,Y)\).to - Second input 2D point set containing \((x,y)\).inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations.confidence - Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation.refineIters - Maximum number of iterations of refining algorithm (Levenberg-Marquardt). Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The returned matrix has the following form: \( \begin{bmatrix} a\_{11} & a\_{12} & b\_1\\ a\_{21} & a\_{22} & b\_2\\ \end{bmatrix} \) The function estimates an optimal 2D affine transformation between two 2D point sets using the selected robust algorithm. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. **Note:** The RANSAC method can handle practically any ratio of outliers but needs a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffinePartial2D, getAffineTransform

#### estimateAffine3D public static int estimateAffine3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal affine transformation between two 3D point sets. It computes \( \begin{bmatrix} x\\ y\\ z\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12} & a\_{13}\\ a\_{21} & a\_{22} & a\_{23}\\ a\_{31} & a\_{32} & a\_{33}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ Z\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ b\_3\\ \end{bmatrix} \)Parameters:src - First input 3D point set containing \((X,Y,Z)\).dst - Second input 3D point set containing \((x,y,z)\).out - Output 3D affine transformation matrix \(3 \times 4\) of the form \( \begin{bmatrix} a\_{11} & a\_{12} & a\_{13} & b\_1\\ a\_{21} & a\_{22} & a\_{23} & b\_2\\ a\_{31} & a\_{32} & a\_{33} & b\_3\\ \end{bmatrix} \)inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier). an inlier. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. The function estimates an optimal 3D affine transformation between two 3D point sets using the RANSAC algorithm. Returns:automatically generated

#### estimateAffine3D public static int estimateAffine3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, double ransacThreshold) Computes an optimal affine transformation between two 3D point sets. It computes \( \begin{bmatrix} x\\ y\\ z\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12} & a\_{13}\\ a\_{21} & a\_{22} & a\_{23}\\ a\_{31} & a\_{32} & a\_{33}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ Z\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ b\_3\\ \end{bmatrix} \)Parameters:src - First input 3D point set containing \((X,Y,Z)\).dst - Second input 3D point set containing \((x,y,z)\).out - Output 3D affine transformation matrix \(3 \times 4\) of the form \( \begin{bmatrix} a\_{11} & a\_{12} & a\_{13} & b\_1\\ a\_{21} & a\_{22} & a\_{23} & b\_2\\ a\_{31} & a\_{32} & a\_{33} & b\_3\\ \end{bmatrix} \)inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).ransacThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. The function estimates an optimal 3D affine transformation between two 3D point sets using the RANSAC algorithm. Returns:automatically generated

#### estimateAffine3D public static int estimateAffine3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) out, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, double ransacThreshold, double confidence) Computes an optimal affine transformation between two 3D point sets. It computes \( \begin{bmatrix} x\\ y\\ z\\ \end{bmatrix} = \begin{bmatrix} a\_{11} & a\_{12} & a\_{13}\\ a\_{21} & a\_{22} & a\_{23}\\ a\_{31} & a\_{32} & a\_{33}\\ \end{bmatrix} \begin{bmatrix} X\\ Y\\ Z\\ \end{bmatrix} + \begin{bmatrix} b\_1\\ b\_2\\ b\_3\\ \end{bmatrix} \)Parameters:src - First input 3D point set containing \((X,Y,Z)\).dst - Second input 3D point set containing \((x,y,z)\).out - Output 3D affine transformation matrix \(3 \times 4\) of the form \( \begin{bmatrix} a\_{11} & a\_{12} & a\_{13} & b\_1\\ a\_{21} & a\_{22} & a\_{23} & b\_2\\ a\_{31} & a\_{32} & a\_{33} & b\_3\\ \end{bmatrix} \)inliers - Output vector indicating which points are inliers (1-inlier, 0-outlier).ransacThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier.confidence - Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. The function estimates an optimal 3D affine transformation between two 3D point sets using the RANSAC algorithm. Returns:automatically generated

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method. a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.

Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations. between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations.confidence - Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation. Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### estimateAffinePartial2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) estimateAffinePartial2D([Mat](http://docs.google.com/org/opencv/core/Mat.html) from, [Mat](http://docs.google.com/org/opencv/core/Mat.html) to, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int method, double ransacReprojThreshold, long maxIters, double confidence, long refineIters) Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.Parameters:from - First input 2D point set.to - Second input 2D point set.inliers - Output vector indicating which points are inliers.method - Robust method used to compute transformation. The following methods are possible:

* + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method RANSAC is the default method.

ransacReprojThreshold - Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.maxIters - The maximum number of robust method iterations.confidence - Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation.refineIters - Maximum number of iterations of refining algorithm (Levenberg-Marquardt). Passing 0 will disable refining, so the output matrix will be output of robust method.Returns:Output 2D affine transformation (4 degrees of freedom) matrix \(2 \times 3\) or empty matrix if transformation could not be estimated. The function estimates an optimal 2D affine transformation with 4 degrees of freedom limited to combinations of translation, rotation, and uniform scaling. Uses the selected algorithm for robust estimation. The computed transformation is then refined further (using only inliers) with the Levenberg-Marquardt method to reduce the re-projection error even more. Estimated transformation matrix is: \( \begin{bmatrix} \cos(\theta) \cdot s & -\sin(\theta) \cdot s & t\_x \\ \sin(\theta) \cdot s & \cos(\theta) \cdot s & t\_y \end{bmatrix} \) Where \( \theta \) is the rotation angle, \( s \) the scaling factor and \( t\_x, t\_y \) are translations in \( x, y \) axes respectively. **Note:** The RANSAC method can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. SEE: estimateAffine2D, getAffineTransform

#### filterHomographyDecompByVisibleRefpoints public static void filterHomographyDecompByVisibleRefpoints(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals, [Mat](http://docs.google.com/org/opencv/core/Mat.html) beforePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) afterPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) possibleSolutions) Filters homography decompositions based on additional information.Parameters:rotations - Vector of rotation matrices.normals - Vector of plane normal matrices.beforePoints - Vector of (rectified) visible reference points before the homography is appliedafterPoints - Vector of (rectified) visible reference points after the homography is appliedpossibleSolutions - Vector of int indices representing the viable solution set after filtering This function is intended to filter the output of the decomposeHomographyMat based on additional information as described in CITE: Malis . The summary of the method: the decomposeHomographyMat function returns 2 unique solutions and their "opposites" for a total of 4 solutions. If we have access to the sets of points visible in the camera frame before and after the homography transformation is applied, we can determine which are the true potential solutions and which are the opposites by verifying which homographies are consistent with all visible reference points being in front of the camera. The inputs are left unchanged; the filtered solution set is returned as indices into the existing one.

#### filterHomographyDecompByVisibleRefpoints public static void filterHomographyDecompByVisibleRefpoints(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rotations, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> normals, [Mat](http://docs.google.com/org/opencv/core/Mat.html) beforePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) afterPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) possibleSolutions, [Mat](http://docs.google.com/org/opencv/core/Mat.html) pointsMask) Filters homography decompositions based on additional information.Parameters:rotations - Vector of rotation matrices.normals - Vector of plane normal matrices.beforePoints - Vector of (rectified) visible reference points before the homography is appliedafterPoints - Vector of (rectified) visible reference points after the homography is appliedpossibleSolutions - Vector of int indices representing the viable solution set after filteringpointsMask - optional Mat/Vector of 8u type representing the mask for the inliers as given by the findHomography function This function is intended to filter the output of the decomposeHomographyMat based on additional information as described in CITE: Malis . The summary of the method: the decomposeHomographyMat function returns 2 unique solutions and their "opposites" for a total of 4 solutions. If we have access to the sets of points visible in the camera frame before and after the homography transformation is applied, we can determine which are the true potential solutions and which are the opposites by verifying which homographies are consistent with all visible reference points being in front of the camera. The inputs are left unchanged; the filtered solution set is returned as indices into the existing one.

#### filterSpeckles public static void filterSpeckles([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, double newVal, int maxSpeckleSize, double maxDiff) Filters off small noise blobs (speckles) in the disparity mapParameters:img - The input 16-bit signed disparity imagenewVal - The disparity value used to paint-off the specklesmaxSpeckleSize - The maximum speckle size to consider it a speckle. Larger blobs are not affected by the algorithmmaxDiff - Maximum difference between neighbor disparity pixels to put them into the same blob. Note that since StereoBM, StereoSGBM and may be other algorithms return a fixed-point disparity map, where disparity values are multiplied by 16, this scale factor should be taken into account when specifying this parameter value.

#### filterSpeckles public static void filterSpeckles([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, double newVal, int maxSpeckleSize, double maxDiff, [Mat](http://docs.google.com/org/opencv/core/Mat.html) buf) Filters off small noise blobs (speckles) in the disparity mapParameters:img - The input 16-bit signed disparity imagenewVal - The disparity value used to paint-off the specklesmaxSpeckleSize - The maximum speckle size to consider it a speckle. Larger blobs are not affected by the algorithmmaxDiff - Maximum difference between neighbor disparity pixels to put them into the same blob. Note that since StereoBM, StereoSGBM and may be other algorithms return a fixed-point disparity map, where disparity values are multiplied by 16, this scale factor should be taken into account when specifying this parameter value.buf - The optional temporary buffer to avoid memory allocation within the function.

#### find4QuadCornerSubpix public static boolean find4QuadCornerSubpix([Mat](http://docs.google.com/org/opencv/core/Mat.html) img, [Mat](http://docs.google.com/org/opencv/core/Mat.html) corners, [Size](http://docs.google.com/org/opencv/core/Size.html) region\_size)

#### findChessboardCorners public static boolean findChessboardCorners([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners) Finds the positions of internal corners of the chessboard.Parameters:image - Source chessboard view. It must be an 8-bit grayscale or color image.patternSize - Number of inner corners per a chessboard row and column ( patternSize = cvSize(points\_per\_row,points\_per\_colum) = cvSize(columns,rows) ).corners - Output array of detected corners.

* + REF: CALIB\_CB\_ADAPTIVE\_THRESH Use adaptive thresholding to convert the image to black and white, rather than a fixed threshold level (computed from the average image brightness).
  + REF: CALIB\_CB\_NORMALIZE\_IMAGE Normalize the image gamma with equalizeHist before applying fixed or adaptive thresholding.
  + REF: CALIB\_CB\_FILTER\_QUADS Use additional criteria (like contour area, perimeter, square-like shape) to filter out false quads extracted at the contour retrieval stage.
  + REF: CALIB\_CB\_FAST\_CHECK Run a fast check on the image that looks for chessboard corners, and shortcut the call if none is found. This can drastically speed up the call in the degenerate condition when no chessboard is observed.

The function attempts to determine whether the input image is a view of the chessboard pattern and locate the internal chessboard corners. The function returns a non-zero value if all of the corners are found and they are placed in a certain order (row by row, left to right in every row). Otherwise, if the function fails to find all the corners or reorder them, it returns 0. For example, a regular chessboard has 8 x 8 squares and 7 x 7 internal corners, that is, points where the black squares touch each other. The detected coordinates are approximate, and to determine their positions more accurately, the function calls cornerSubPix. You also may use the function cornerSubPix with different parameters if returned coordinates are not accurate enough. Sample usage of detecting and drawing chessboard corners: : Size patternsize(8,6); //interior number of corners Mat gray = ....; //source image vector<Point2f> corners; //this will be filled by the detected corners //CALIB\_CB\_FAST\_CHECK saves a lot of time on images //that do not contain any chessboard corners bool patternfound = findChessboardCorners(gray, patternsize, corners, CALIB\_CB\_ADAPTIVE\_THRESH + CALIB\_CB\_NORMALIZE\_IMAGE + CALIB\_CB\_FAST\_CHECK); if(patternfound) cornerSubPix(gray, corners, Size(11, 11), Size(-1, -1), TermCriteria(CV\_TERMCRIT\_EPS + CV\_TERMCRIT\_ITER, 30, 0.1)); drawChessboardCorners(img, patternsize, Mat(corners), patternfound); **Note:** The function requires white space (like a square-thick border, the wider the better) around the board to make the detection more robust in various environments. Otherwise, if there is no border and the background is dark, the outer black squares cannot be segmented properly and so the square grouping and ordering algorithm fails. Returns:automatically generated

#### findChessboardCorners public static boolean findChessboardCorners([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) corners, int flags) Finds the positions of internal corners of the chessboard.Parameters:image - Source chessboard view. It must be an 8-bit grayscale or color image.patternSize - Number of inner corners per a chessboard row and column ( patternSize = cvSize(points\_per\_row,points\_per\_colum) = cvSize(columns,rows) ).corners - Output array of detected corners.flags - Various operation flags that can be zero or a combination of the following values:

* + REF: CALIB\_CB\_ADAPTIVE\_THRESH Use adaptive thresholding to convert the image to black and white, rather than a fixed threshold level (computed from the average image brightness).
  + REF: CALIB\_CB\_NORMALIZE\_IMAGE Normalize the image gamma with equalizeHist before applying fixed or adaptive thresholding.
  + REF: CALIB\_CB\_FILTER\_QUADS Use additional criteria (like contour area, perimeter, square-like shape) to filter out false quads extracted at the contour retrieval stage.
  + REF: CALIB\_CB\_FAST\_CHECK Run a fast check on the image that looks for chessboard corners, and shortcut the call if none is found. This can drastically speed up the call in the degenerate condition when no chessboard is observed.

The function attempts to determine whether the input image is a view of the chessboard pattern and locate the internal chessboard corners. The function returns a non-zero value if all of the corners are found and they are placed in a certain order (row by row, left to right in every row). Otherwise, if the function fails to find all the corners or reorder them, it returns 0. For example, a regular chessboard has 8 x 8 squares and 7 x 7 internal corners, that is, points where the black squares touch each other. The detected coordinates are approximate, and to determine their positions more accurately, the function calls cornerSubPix. You also may use the function cornerSubPix with different parameters if returned coordinates are not accurate enough. Sample usage of detecting and drawing chessboard corners: : Size patternsize(8,6); //interior number of corners Mat gray = ....; //source image vector<Point2f> corners; //this will be filled by the detected corners //CALIB\_CB\_FAST\_CHECK saves a lot of time on images //that do not contain any chessboard corners bool patternfound = findChessboardCorners(gray, patternsize, corners, CALIB\_CB\_ADAPTIVE\_THRESH + CALIB\_CB\_NORMALIZE\_IMAGE + CALIB\_CB\_FAST\_CHECK); if(patternfound) cornerSubPix(gray, corners, Size(11, 11), Size(-1, -1), TermCriteria(CV\_TERMCRIT\_EPS + CV\_TERMCRIT\_ITER, 30, 0.1)); drawChessboardCorners(img, patternsize, Mat(corners), patternfound); **Note:** The function requires white space (like a square-thick border, the wider the better) around the board to make the detection more robust in various environments. Otherwise, if there is no border and the background is dark, the outer black squares cannot be segmented properly and so the square grouping and ordering algorithm fails. Returns:automatically generated

#### findCirclesGrid public static boolean findCirclesGrid([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers)

#### findCirclesGrid public static boolean findCirclesGrid([Mat](http://docs.google.com/org/opencv/core/Mat.html) image, [Size](http://docs.google.com/org/opencv/core/Size.html) patternSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) centers, int flags)

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 . are feature points from cameras with same focal length and principal point.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.method - Method for computing a fundamental matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.method - Method for computing a fundamental matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.

prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob, double threshold)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.method - Method for computing a fundamental matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm.

threshold - Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct. for the other points. The array is computed only in the RANSAC and LMedS methods.This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, int method, double prob, double threshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .focal - focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.method - Method for computing a fundamental matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm.

threshold - Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.mask - Output array of N elements, every element of which is set to 0 for outliers and to 1 for the other points. The array is computed only in the RANSAC and LMedS methods.This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix) Calculates an essential matrix from the corresponding points in two images.Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use undistortPoints() with P = cv::NoArray() for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. confidence (probability) that the estimated matrix is correct. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function estimates essential matrix based on the five-point algorithm solver in CITE: Nister03 . CITE: SteweniusCFS is also a related. The epipolar geometry is described by the following equation: \([p\_2; 1]^T K^{-T} E K^{-1} [p\_1; 1] = 0\) where \(E\) is an essential matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The result of this function may be passed further to decomposeEssentialMat or recoverPose to recover the relative pose between cameras. Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method) Calculates an essential matrix from the corresponding points in two images.Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use undistortPoints() with P = cv::NoArray() for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.method - Method for computing an essential matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm. confidence (probability) that the estimated matrix is correct. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. for the other points. The array is computed only in the RANSAC and LMedS methods.

This function estimates essential matrix based on the five-point algorithm solver in CITE: Nister03 . CITE: SteweniusCFS is also a related. The epipolar geometry is described by the following equation: \([p\_2; 1]^T K^{-T} E K^{-1} [p\_1; 1] = 0\) where \(E\) is an essential matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The result of this function may be passed further to decomposeEssentialMat or recoverPose to recover the relative pose between cameras. Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob) Calculates an essential matrix from the corresponding points in two images.Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use undistortPoints() with P = cv::NoArray() for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.method - Method for computing an essential matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm.

prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct. line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. for the other points. The array is computed only in the RANSAC and LMedS methods.This function estimates essential matrix based on the five-point algorithm solver in CITE: Nister03 . CITE: SteweniusCFS is also a related. The epipolar geometry is described by the following equation: \([p\_2; 1]^T K^{-T} E K^{-1} [p\_1; 1] = 0\) where \(E\) is an essential matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The result of this function may be passed further to decomposeEssentialMat or recoverPose to recover the relative pose between cameras. Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob, double threshold) Calculates an essential matrix from the corresponding points in two images.Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use undistortPoints() with P = cv::NoArray() for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.method - Method for computing an essential matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm.

prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.threshold - Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise. for the other points. The array is computed only in the RANSAC and LMedS methods.This function estimates essential matrix based on the five-point algorithm solver in CITE: Nister03 . CITE: SteweniusCFS is also a related. The epipolar geometry is described by the following equation: \([p\_2; 1]^T K^{-T} E K^{-1} [p\_1; 1] = 0\) where \(E\) is an essential matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The result of this function may be passed further to decomposeEssentialMat or recoverPose to recover the relative pose between cameras. Returns:automatically generated

#### findEssentialMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findEssentialMat([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, int method, double prob, double threshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates an essential matrix from the corresponding points in two images.Parameters:points1 - Array of N (N >= 5) 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix. If this assumption does not hold for your use case, use undistortPoints() with P = cv::NoArray() for both cameras to transform image points to normalized image coordinates, which are valid for the identity camera intrinsic matrix. When passing these coordinates, pass the identity matrix for this parameter.method - Method for computing an essential matrix.

* + REF: RANSAC for the RANSAC algorithm.
  + REF: LMEDS for the LMedS algorithm.

prob - Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.threshold - Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.mask - Output array of N elements, every element of which is set to 0 for outliers and to 1 for the other points. The array is computed only in the RANSAC and LMedS methods.This function estimates essential matrix based on the five-point algorithm solver in CITE: Nister03 . CITE: SteweniusCFS is also a related. The epipolar geometry is described by the following equation: \([p\_2; 1]^T K^{-T} E K^{-1} [p\_1; 1] = 0\) where \(E\) is an essential matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The result of this function may be passed further to decomposeEssentialMat or recoverPose to recover the relative pose between cameras. Returns:automatically generated

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2)

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method)

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold)

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence)

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, int maxIters) Calculates a fundamental matrix from the corresponding points in two images.Parameters:points1 - Array of N points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .method - Method for computing a fundamental matrix.

* + REF: FM\_7POINT for a 7-point algorithm. \(N = 7\)
  + REF: FM\_8POINT for an 8-point algorithm. \(N \ge 8\)
  + REF: FM\_RANSAC for the RANSAC algorithm. \(N \ge 8\)
  + REF: FM\_LMEDS for the LMedS algorithm. \(N \ge 8\)

ransacReprojThreshold - Parameter used only for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.confidence - Parameter used for the RANSAC and LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.maxIters - The maximum number of robust method iterations.The epipolar geometry is described by the following equation: \([p\_2; 1]^T F [p\_1; 1] = 0\) where \(F\) is a fundamental matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The function calculates the fundamental matrix using one of four methods listed above and returns the found fundamental matrix. Normally just one matrix is found. But in case of the 7-point algorithm, the function may return up to 3 solutions ( \(9 \times 3\) matrix that stores all 3 matrices sequentially). The calculated fundamental matrix may be passed further to computeCorrespondEpilines that finds the epipolar lines corresponding to the specified points. It can also be passed to stereoRectifyUncalibrated to compute the rectification transformation. : // Example. Estimation of fundamental matrix using the RANSAC algorithm int point\_count = 100; vector<Point2f> points1(point\_count); vector<Point2f> points2(point\_count); // initialize the points here ... for( int i = 0; i < point\_count; i++ ) { points1[i] = ...; points2[i] = ...; } Mat fundamental\_matrix = findFundamentalMat(points1, points2, FM\_RANSAC, 3, 0.99); Returns:automatically generated

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, int maxIters, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Calculates a fundamental matrix from the corresponding points in two images.Parameters:points1 - Array of N points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .method - Method for computing a fundamental matrix.

* + REF: FM\_7POINT for a 7-point algorithm. \(N = 7\)
  + REF: FM\_8POINT for an 8-point algorithm. \(N \ge 8\)
  + REF: FM\_RANSAC for the RANSAC algorithm. \(N \ge 8\)
  + REF: FM\_LMEDS for the LMedS algorithm. \(N \ge 8\)

ransacReprojThreshold - Parameter used only for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.confidence - Parameter used for the RANSAC and LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.mask - optional output maskmaxIters - The maximum number of robust method iterations.The epipolar geometry is described by the following equation: \([p\_2; 1]^T F [p\_1; 1] = 0\) where \(F\) is a fundamental matrix, \(p\_1\) and \(p\_2\) are corresponding points in the first and the second images, respectively. The function calculates the fundamental matrix using one of four methods listed above and returns the found fundamental matrix. Normally just one matrix is found. But in case of the 7-point algorithm, the function may return up to 3 solutions ( \(9 \times 3\) matrix that stores all 3 matrices sequentially). The calculated fundamental matrix may be passed further to computeCorrespondEpilines that finds the epipolar lines corresponding to the specified points. It can also be passed to stereoRectifyUncalibrated to compute the rectification transformation. : // Example. Estimation of fundamental matrix using the RANSAC algorithm int point\_count = 100; vector<Point2f> points1(point\_count); vector<Point2f> points2(point\_count); // initialize the points here ... for( int i = 0; i < point\_count; i++ ) { points1[i] = ...; points2[i] = ...; } Mat fundamental\_matrix = findFundamentalMat(points1, points2, FM\_RANSAC, 3, 0.99); Returns:automatically generated

#### findFundamentalMat public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findFundamentalMat([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points1, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) points2, int method, double ransacReprojThreshold, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10. mask values are ignored.

The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .method - Method used to compute a homography matrix. The following methods are possible:

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10. mask values are ignored.

The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .method - Method used to compute a homography matrix. The following methods are possible:

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method

ransacReprojThreshold - Maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10. mask values are ignored.The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .method - Method used to compute a homography matrix. The following methods are possible:

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method

ransacReprojThreshold - Maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10.mask - Optional output mask set by a robust method ( RANSAC or LMeDS ). Note that the input mask values are ignored.The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int maxIters) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .method - Method used to compute a homography matrix. The following methods are possible:

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method

ransacReprojThreshold - Maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10.mask - Optional output mask set by a robust method ( RANSAC or LMeDS ). Note that the input mask values are ignored.maxIters - The maximum number of RANSAC iterations.The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### findHomography public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) findHomography([MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) srcPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) dstPoints, int method, double ransacReprojThreshold, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, int maxIters, double confidence) Finds a perspective transformation between two planes.Parameters:srcPoints - Coordinates of the points in the original plane, a matrix of the type CV\_32FC2 or vector<Point2f> .dstPoints - Coordinates of the points in the target plane, a matrix of the type CV\_32FC2 or a vector<Point2f> .method - Method used to compute a homography matrix. The following methods are possible:

* + **0** - a regular method using all the points, i.e., the least squares method
  + REF: RANSAC - RANSAC-based robust method
  + REF: LMEDS - Least-Median robust method
  + REF: RHO - PROSAC-based robust method

ransacReprojThreshold - Maximum allowed reprojection error to treat a point pair as an inlier (used in the RANSAC and RHO methods only). That is, if \(\| \texttt{dstPoints} \_i - \texttt{convertPointsHomogeneous} ( \texttt{H} \* \texttt{srcPoints} \_i) \|\_2 > \texttt{ransacReprojThreshold}\) then the point \(i\) is considered as an outlier. If srcPoints and dstPoints are measured in pixels, it usually makes sense to set this parameter somewhere in the range of 1 to 10.mask - Optional output mask set by a robust method ( RANSAC or LMeDS ). Note that the input mask values are ignored.maxIters - The maximum number of RANSAC iterations.confidence - Confidence level, between 0 and 1.The function finds and returns the perspective transformation \(H\) between the source and the destination planes: \(s\_i \vecthree{x'\_i}{y'\_i}{1} \sim H \vecthree{x\_i}{y\_i}{1}\) so that the back-projection error \(\sum \_i \left ( x'\_i- \frac{h\_{11} x\_i + h\_{12} y\_i + h\_{13}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2+ \left ( y'\_i- \frac{h\_{21} x\_i + h\_{22} y\_i + h\_{23}}{h\_{31} x\_i + h\_{32} y\_i + h\_{33}} \right )^2\) is minimized. If the parameter method is set to the default value 0, the function uses all the point pairs to compute an initial homography estimate with a simple least-squares scheme. However, if not all of the point pairs ( \(srcPoints\_i\), \(dstPoints\_i\) ) fit the rigid perspective transformation (that is, there are some outliers), this initial estimate will be poor. In this case, you can use one of the three robust methods. The methods RANSAC, LMeDS and RHO try many different random subsets of the corresponding point pairs (of four pairs each, collinear pairs are discarded), estimate the homography matrix using this subset and a simple least-squares algorithm, and then compute the quality/goodness of the computed homography (which is the number of inliers for RANSAC or the least median re-projection error for LMeDS). The best subset is then used to produce the initial estimate of the homography matrix and the mask of inliers/outliers. Regardless of the method, robust or not, the computed homography matrix is refined further (using inliers only in case of a robust method) with the Levenberg-Marquardt method to reduce the re-projection error even more. The methods RANSAC and RHO can handle practically any ratio of outliers but need a threshold to distinguish inliers from outliers. The method LMeDS does not need any threshold but it works correctly only when there are more than 50% of inliers. Finally, if there are no outliers and the noise is rather small, use the default method (method=0). The function is used to find initial intrinsic and extrinsic matrices. Homography matrix is determined up to a scale. Thus, it is normalized so that \(h\_{33}=1\). Note that whenever an \(H\) matrix cannot be estimated, an empty one will be returned. SEE: getAffineTransform, estimateAffine2D, estimateAffinePartial2D, getPerspectiveTransform, warpPerspective, perspectiveTransform Returns:automatically generated

#### fisheye\_calibrate public static double fisheye\_calibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs) Performs camera calibarationParameters:objectPoints - vector of vectors of calibration pattern points in the calibration pattern coordinate space.imagePoints - vector of vectors of the projections of calibration pattern points. imagePoints.size() and objectPoints.size() and imagePoints[i].size() must be equal to objectPoints[i].size() for each i.image\_size - Size of the image used only to initialize the camera intrinsic matrix.K - Output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is specified, some or all of fx, fy, cx, cy must be initialized before calling the function.D - Output vector of distortion coefficients \(\distcoeffsfisheye\).rvecs - Output vector of rotation vectors (see Rodrigues ) estimated for each pattern view. That is, each k-th rotation vector together with the corresponding k-th translation vector (see the next output parameter description) brings the calibration pattern from the model coordinate space (in which object points are specified) to the world coordinate space, that is, a real position of the calibration pattern in the k-th pattern view (k=0.. \*M\* -1).tvecs - Output vector of translation vectors estimated for each pattern view.

* + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.
  + REF: fisheye::CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is set too.

Returns:automatically generated

#### fisheye\_calibrate public static double fisheye\_calibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) Performs camera calibarationParameters:objectPoints - vector of vectors of calibration pattern points in the calibration pattern coordinate space.imagePoints - vector of vectors of the projections of calibration pattern points. imagePoints.size() and objectPoints.size() and imagePoints[i].size() must be equal to objectPoints[i].size() for each i.image\_size - Size of the image used only to initialize the camera intrinsic matrix.K - Output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is specified, some or all of fx, fy, cx, cy must be initialized before calling the function.D - Output vector of distortion coefficients \(\distcoeffsfisheye\).rvecs - Output vector of rotation vectors (see Rodrigues ) estimated for each pattern view. That is, each k-th rotation vector together with the corresponding k-th translation vector (see the next output parameter description) brings the calibration pattern from the model coordinate space (in which object points are specified) to the world coordinate space, that is, a real position of the calibration pattern in the k-th pattern view (k=0.. \*M\* -1).tvecs - Output vector of translation vectors estimated for each pattern view.flags - Different flags that may be zero or a combination of the following values:

* + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.
  + REF: fisheye::CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is set too.

Returns:automatically generated

#### fisheye\_calibrate public static double fisheye\_calibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Performs camera calibarationParameters:objectPoints - vector of vectors of calibration pattern points in the calibration pattern coordinate space.imagePoints - vector of vectors of the projections of calibration pattern points. imagePoints.size() and objectPoints.size() and imagePoints[i].size() must be equal to objectPoints[i].size() for each i.image\_size - Size of the image used only to initialize the camera intrinsic matrix.K - Output 3x3 floating-point camera intrinsic matrix \(\cameramatrix{A}\) . If REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is specified, some or all of fx, fy, cx, cy must be initialized before calling the function.D - Output vector of distortion coefficients \(\distcoeffsfisheye\).rvecs - Output vector of rotation vectors (see Rodrigues ) estimated for each pattern view. That is, each k-th rotation vector together with the corresponding k-th translation vector (see the next output parameter description) brings the calibration pattern from the model coordinate space (in which object points are specified) to the world coordinate space, that is, a real position of the calibration pattern in the k-th pattern view (k=0.. \*M\* -1).tvecs - Output vector of translation vectors estimated for each pattern view.flags - Different flags that may be zero or a combination of the following values:

* + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS cameraMatrix contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center ( imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.
  + REF: fisheye::CALIB\_FIX\_PRINCIPAL\_POINT The principal point is not changed during the global optimization. It stays at the center or at a different location specified when REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS is set too.

criteria - Termination criteria for the iterative optimization algorithm.Returns:automatically generated

#### fisheye\_distortPoints public static void fisheye\_distortPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Distorts 2D points using fisheye model.Parameters:undistorted - Array of object points, 1xN/Nx1 2-channel (or vector<Point2f> ), where N is the number of points in the view.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).distorted - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> . Note that the function assumes the camera intrinsic matrix of the undistorted points to be identity. This means if you want to transform back points undistorted with undistortPoints() you have to multiply them with \(P^{-1}\).

#### fisheye\_distortPoints public static void fisheye\_distortPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha) Distorts 2D points using fisheye model.Parameters:undistorted - Array of object points, 1xN/Nx1 2-channel (or vector<Point2f> ), where N is the number of points in the view.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).alpha - The skew coefficient.distorted - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> . Note that the function assumes the camera intrinsic matrix of the undistorted points to be identity. This means if you want to transform back points undistorted with undistortPoints() you have to multiply them with \(P^{-1}\).

#### fisheye\_estimateNewCameraMatrixForUndistortRectify public static void fisheye\_estimateNewCameraMatrixForUndistortRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P) Estimates new camera intrinsic matrix for undistortion or rectification.Parameters:K - Camera intrinsic matrix \(cameramatrix{K}\).image\_size - Size of the imageD - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4) length. Balance is in range of [0, 1].

#### fisheye\_estimateNewCameraMatrixForUndistortRectify public static void fisheye\_estimateNewCameraMatrixForUndistortRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance) Estimates new camera intrinsic matrix for undistortion or rectification.Parameters:K - Camera intrinsic matrix \(cameramatrix{K}\).image\_size - Size of the imageD - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4)balance - Sets the new focal length in range between the min focal length and the max focal length. Balance is in range of [0, 1].

#### fisheye\_estimateNewCameraMatrixForUndistortRectify public static void fisheye\_estimateNewCameraMatrixForUndistortRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size) Estimates new camera intrinsic matrix for undistortion or rectification.Parameters:K - Camera intrinsic matrix \(cameramatrix{K}\).image\_size - Size of the imageD - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4)balance - Sets the new focal length in range between the min focal length and the max focal length. Balance is in range of [0, 1].new\_size - the new size

#### fisheye\_estimateNewCameraMatrixForUndistortRectify public static void fisheye\_estimateNewCameraMatrixForUndistortRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Size](http://docs.google.com/org/opencv/core/Size.html) image\_size, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, double balance, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size, double fov\_scale) Estimates new camera intrinsic matrix for undistortion or rectification.Parameters:K - Camera intrinsic matrix \(cameramatrix{K}\).image\_size - Size of the imageD - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4)balance - Sets the new focal length in range between the min focal length and the max focal length. Balance is in range of [0, 1].new\_size - the new sizefov\_scale - Divisor for new focal length.

#### fisheye\_initUndistortRectifyMap public static void fisheye\_initUndistortRectifyMap([Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P, [Size](http://docs.google.com/org/opencv/core/Size.html) size, int m1type, [Mat](http://docs.google.com/org/opencv/core/Mat.html) map1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) map2) Computes undistortion and rectification maps for image transform by cv::remap(). If D is empty zero distortion is used, if R or P is empty identity matrixes are used.Parameters:K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4)size - Undistorted image size.m1type - Type of the first output map that can be CV\_32FC1 or CV\_16SC2 . See convertMaps() for details.map1 - The first output map.map2 - The second output map.

#### fisheye\_projectPoints public static void fisheye\_projectPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D)

#### fisheye\_projectPoints public static void fisheye\_projectPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha)

#### fisheye\_projectPoints public static void fisheye\_projectPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, double alpha, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian)

#### fisheye\_stereoCalibrate public static double fisheye\_stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T) Performs stereo calibrationParameters:objectPoints - Vector of vectors of the calibration pattern points.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera.K1 - Input/output first camera intrinsic matrix: \(\vecthreethree{f\_x^{(j)}}{0}{c\_x^{(j)}}{0}{f\_y^{(j)}}{c\_y^{(j)}}{0}{0}{1}\) , \(j = 0,\, 1\) . If any of REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS , REF: fisheye::CALIB\_FIX\_INTRINSIC are specified, some or all of the matrix components must be initialized.D1 - Input/output vector of distortion coefficients \(\distcoeffsfisheye\) of 4 elements.K2 - Input/output second camera intrinsic matrix. The parameter is similar to K1 .D2 - Input/output lens distortion coefficients for the second camera. The parameter is similar to D1 .imageSize - Size of the image used only to initialize camera intrinsic matrix.R - Output rotation matrix between the 1st and the 2nd camera coordinate systems.T - Output translation vector between the coordinate systems of the cameras.

* + REF: fisheye::CALIB\_FIX\_INTRINSIC Fix K1, K2? and D1, D2? so that only R, T matrices are estimated.
  + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS K1, K2 contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center (imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.

Returns:automatically generated

#### fisheye\_stereoCalibrate public static double fisheye\_stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, int flags) Performs stereo calibrationParameters:objectPoints - Vector of vectors of the calibration pattern points.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera.K1 - Input/output first camera intrinsic matrix: \(\vecthreethree{f\_x^{(j)}}{0}{c\_x^{(j)}}{0}{f\_y^{(j)}}{c\_y^{(j)}}{0}{0}{1}\) , \(j = 0,\, 1\) . If any of REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS , REF: fisheye::CALIB\_FIX\_INTRINSIC are specified, some or all of the matrix components must be initialized.D1 - Input/output vector of distortion coefficients \(\distcoeffsfisheye\) of 4 elements.K2 - Input/output second camera intrinsic matrix. The parameter is similar to K1 .D2 - Input/output lens distortion coefficients for the second camera. The parameter is similar to D1 .imageSize - Size of the image used only to initialize camera intrinsic matrix.R - Output rotation matrix between the 1st and the 2nd camera coordinate systems.T - Output translation vector between the coordinate systems of the cameras.flags - Different flags that may be zero or a combination of the following values:

* + REF: fisheye::CALIB\_FIX\_INTRINSIC Fix K1, K2? and D1, D2? so that only R, T matrices are estimated.
  + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS K1, K2 contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center (imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.

Returns:automatically generated

#### fisheye\_stereoCalibrate public static double fisheye\_stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Performs stereo calibrationParameters:objectPoints - Vector of vectors of the calibration pattern points.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera.K1 - Input/output first camera intrinsic matrix: \(\vecthreethree{f\_x^{(j)}}{0}{c\_x^{(j)}}{0}{f\_y^{(j)}}{c\_y^{(j)}}{0}{0}{1}\) , \(j = 0,\, 1\) . If any of REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS , REF: fisheye::CALIB\_FIX\_INTRINSIC are specified, some or all of the matrix components must be initialized.D1 - Input/output vector of distortion coefficients \(\distcoeffsfisheye\) of 4 elements.K2 - Input/output second camera intrinsic matrix. The parameter is similar to K1 .D2 - Input/output lens distortion coefficients for the second camera. The parameter is similar to D1 .imageSize - Size of the image used only to initialize camera intrinsic matrix.R - Output rotation matrix between the 1st and the 2nd camera coordinate systems.T - Output translation vector between the coordinate systems of the cameras.flags - Different flags that may be zero or a combination of the following values:

* + REF: fisheye::CALIB\_FIX\_INTRINSIC Fix K1, K2? and D1, D2? so that only R, T matrices are estimated.
  + REF: fisheye::CALIB\_USE\_INTRINSIC\_GUESS K1, K2 contains valid initial values of fx, fy, cx, cy that are optimized further. Otherwise, (cx, cy) is initially set to the image center (imageSize is used), and focal distances are computed in a least-squares fashion.
  + REF: fisheye::CALIB\_RECOMPUTE\_EXTRINSIC Extrinsic will be recomputed after each iteration of intrinsic optimization.
  + REF: fisheye::CALIB\_CHECK\_COND The functions will check validity of condition number.
  + REF: fisheye::CALIB\_FIX\_SKEW Skew coefficient (alpha) is set to zero and stay zero.
  + REF: fisheye::CALIB\_FIX\_K1,..., REF: fisheye::CALIB\_FIX\_K4 Selected distortion coefficients are set to zeros and stay zero.

criteria - Termination criteria for the iterative optimization algorithm.Returns:automatically generated

#### fisheye\_stereoRectify public static void fisheye\_stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags) Stereo rectification for fisheye camera modelParameters:K1 - First camera intrinsic matrix.D1 - First camera distortion parameters.K2 - Second camera intrinsic matrix.D2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix between the coordinate systems of the first and the second cameras.tvec - Translation vector between coordinate systems of the cameras.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see reprojectImageTo3D ).flags - Operation flags that may be zero or REF: fisheye::CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area. initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to larger value can help you preserve details in the original image, especially when there is a big radial distortion. length. Balance is in range of [0, 1].

#### fisheye\_stereoRectify public static void fisheye\_stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize) Stereo rectification for fisheye camera modelParameters:K1 - First camera intrinsic matrix.D1 - First camera distortion parameters.K2 - Second camera intrinsic matrix.D2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix between the coordinate systems of the first and the second cameras.tvec - Translation vector between coordinate systems of the cameras.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see reprojectImageTo3D ).flags - Operation flags that may be zero or REF: fisheye::CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to larger value can help you preserve details in the original image, especially when there is a big radial distortion. length. Balance is in range of [0, 1].

#### fisheye\_stereoRectify public static void fisheye\_stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, double balance) Stereo rectification for fisheye camera modelParameters:K1 - First camera intrinsic matrix.D1 - First camera distortion parameters.K2 - Second camera intrinsic matrix.D2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix between the coordinate systems of the first and the second cameras.tvec - Translation vector between coordinate systems of the cameras.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see reprojectImageTo3D ).flags - Operation flags that may be zero or REF: fisheye::CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to larger value can help you preserve details in the original image, especially when there is a big radial distortion.balance - Sets the new focal length in range between the min focal length and the max focal length. Balance is in range of [0, 1].

#### fisheye\_stereoRectify public static void fisheye\_stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) K1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, double balance, double fov\_scale) Stereo rectification for fisheye camera modelParameters:K1 - First camera intrinsic matrix.D1 - First camera distortion parameters.K2 - Second camera intrinsic matrix.D2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix between the coordinate systems of the first and the second cameras.tvec - Translation vector between coordinate systems of the cameras.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see reprojectImageTo3D ).flags - Operation flags that may be zero or REF: fisheye::CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to larger value can help you preserve details in the original image, especially when there is a big radial distortion.balance - Sets the new focal length in range between the min focal length and the max focal length. Balance is in range of [0, 1].fov\_scale - Divisor for new focal length.

#### fisheye\_undistortImage public static void fisheye\_undistortImage([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Transforms an image to compensate for fisheye lens distortion.Parameters:distorted - image with fisheye lens distortion.undistorted - Output image with compensated fisheye lens distortion.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\). may additionally scale and shift the result by using a different matrix. The function transforms an image to compensate radial and tangential lens distortion. The function is simply a combination of fisheye::initUndistortRectifyMap (with unity R ) and remap (with bilinear interpolation). See the former function for details of the transformation being performed. See below the results of undistortImage.

* + a\) result of undistort of perspective camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4, k\_5, k\_6) of distortion were optimized under calibration)
    - b\) result of fisheye::undistortImage of fisheye camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4) of fisheye distortion were optimized under calibration)
    - c\) original image was captured with fisheye lens

Pictures a) and b) almost the same. But if we consider points of image located far from the center of image, we can notice that on image a) these points are distorted.![image](pics/fisheye\_undistorted.jpg)

#### fisheye\_undistortImage public static void fisheye\_undistortImage([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Knew) Transforms an image to compensate for fisheye lens distortion.Parameters:distorted - image with fisheye lens distortion.undistorted - Output image with compensated fisheye lens distortion.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).Knew - Camera intrinsic matrix of the distorted image. By default, it is the identity matrix but you may additionally scale and shift the result by using a different matrix. The function transforms an image to compensate radial and tangential lens distortion. The function is simply a combination of fisheye::initUndistortRectifyMap (with unity R ) and remap (with bilinear interpolation). See the former function for details of the transformation being performed. See below the results of undistortImage.

* + a\) result of undistort of perspective camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4, k\_5, k\_6) of distortion were optimized under calibration)
    - b\) result of fisheye::undistortImage of fisheye camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4) of fisheye distortion were optimized under calibration)
    - c\) original image was captured with fisheye lens

Pictures a) and b) almost the same. But if we consider points of image located far from the center of image, we can notice that on image a) these points are distorted.![image](pics/fisheye\_undistorted.jpg)

#### fisheye\_undistortImage public static void fisheye\_undistortImage([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Knew, [Size](http://docs.google.com/org/opencv/core/Size.html) new\_size) Transforms an image to compensate for fisheye lens distortion.Parameters:distorted - image with fisheye lens distortion.undistorted - Output image with compensated fisheye lens distortion.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).Knew - Camera intrinsic matrix of the distorted image. By default, it is the identity matrix but you may additionally scale and shift the result by using a different matrix.new\_size - the new size The function transforms an image to compensate radial and tangential lens distortion. The function is simply a combination of fisheye::initUndistortRectifyMap (with unity R ) and remap (with bilinear interpolation). See the former function for details of the transformation being performed. See below the results of undistortImage.

* + a\) result of undistort of perspective camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4, k\_5, k\_6) of distortion were optimized under calibration)
    - b\) result of fisheye::undistortImage of fisheye camera model (all possible coefficients (k\_1, k\_2, k\_3, k\_4) of fisheye distortion were optimized under calibration)
    - c\) original image was captured with fisheye lens

Pictures a) and b) almost the same. But if we consider points of image located far from the center of image, we can notice that on image a) these points are distorted.![image](pics/fisheye\_undistorted.jpg)

#### fisheye\_undistortPoints public static void fisheye\_undistortPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D) Undistorts 2D points using fisheye modelParameters:distorted - Array of object points, 1xN/Nx1 2-channel (or vector<Point2f> ), where N is the number of points in the view.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\). 1-channel or 1x1 3-channelundistorted - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .

#### fisheye\_undistortPoints public static void fisheye\_undistortPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R) Undistorts 2D points using fisheye modelParameters:distorted - Array of object points, 1xN/Nx1 2-channel (or vector<Point2f> ), where N is the number of points in the view.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelundistorted - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .

#### fisheye\_undistortPoints public static void fisheye\_undistortPoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) distorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) undistorted, [Mat](http://docs.google.com/org/opencv/core/Mat.html) K, [Mat](http://docs.google.com/org/opencv/core/Mat.html) D, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P) Undistorts 2D points using fisheye modelParameters:distorted - Array of object points, 1xN/Nx1 2-channel (or vector<Point2f> ), where N is the number of points in the view.K - Camera intrinsic matrix \(cameramatrix{K}\).D - Input vector of distortion coefficients \(\distcoeffsfisheye\).R - Rectification transformation in the object space: 3x3 1-channel, or vector: 3x1/1x3 1-channel or 1x1 3-channelP - New camera intrinsic matrix (3x3) or new projection matrix (3x4)undistorted - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .

#### getOptimalNewCameraMatrix public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) getOptimalNewCameraMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha) Returns the new camera intrinsic matrix based on the free scaling parameter.Parameters:cameraMatrix - Input camera intrinsic matrix.distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.imageSize - Original image size.alpha - Free scaling parameter between 0 (when all the pixels in the undistorted image are valid) and 1 (when all the source image pixels are retained in the undistorted image). See stereoRectify for details. undistorted image. See roi1, roi2 description in stereoRectify . principal point should be at the image center or not. By default, the principal point is chosen to best fit a subset of the source image (determined by alpha) to the corrected image. Returns:new\_camera\_matrix Output new camera intrinsic matrix. The function computes and returns the optimal new camera intrinsic matrix based on the free scaling parameter. By varying this parameter, you may retrieve only sensible pixels alpha=0 , keep all the original image pixels if there is valuable information in the corners alpha=1 , or get something in between. When alpha>0 , the undistorted result is likely to have some black pixels corresponding to "virtual" pixels outside of the captured distorted image. The original camera intrinsic matrix, distortion coefficients, the computed new camera intrinsic matrix, and newImageSize should be passed to initUndistortRectifyMap to produce the maps for remap .

#### getOptimalNewCameraMatrix public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) getOptimalNewCameraMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize) Returns the new camera intrinsic matrix based on the free scaling parameter.Parameters:cameraMatrix - Input camera intrinsic matrix.distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.imageSize - Original image size.alpha - Free scaling parameter between 0 (when all the pixels in the undistorted image are valid) and 1 (when all the source image pixels are retained in the undistorted image). See stereoRectify for details.newImgSize - Image size after rectification. By default, it is set to imageSize . undistorted image. See roi1, roi2 description in stereoRectify . principal point should be at the image center or not. By default, the principal point is chosen to best fit a subset of the source image (determined by alpha) to the corrected image. Returns:new\_camera\_matrix Output new camera intrinsic matrix. The function computes and returns the optimal new camera intrinsic matrix based on the free scaling parameter. By varying this parameter, you may retrieve only sensible pixels alpha=0 , keep all the original image pixels if there is valuable information in the corners alpha=1 , or get something in between. When alpha>0 , the undistorted result is likely to have some black pixels corresponding to "virtual" pixels outside of the captured distorted image. The original camera intrinsic matrix, distortion coefficients, the computed new camera intrinsic matrix, and newImageSize should be passed to initUndistortRectifyMap to produce the maps for remap .

#### getOptimalNewCameraMatrix public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) getOptimalNewCameraMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI) Returns the new camera intrinsic matrix based on the free scaling parameter.Parameters:cameraMatrix - Input camera intrinsic matrix.distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.imageSize - Original image size.alpha - Free scaling parameter between 0 (when all the pixels in the undistorted image are valid) and 1 (when all the source image pixels are retained in the undistorted image). See stereoRectify for details.newImgSize - Image size after rectification. By default, it is set to imageSize .validPixROI - Optional output rectangle that outlines all-good-pixels region in the undistorted image. See roi1, roi2 description in stereoRectify . principal point should be at the image center or not. By default, the principal point is chosen to best fit a subset of the source image (determined by alpha) to the corrected image. Returns:new\_camera\_matrix Output new camera intrinsic matrix. The function computes and returns the optimal new camera intrinsic matrix based on the free scaling parameter. By varying this parameter, you may retrieve only sensible pixels alpha=0 , keep all the original image pixels if there is valuable information in the corners alpha=1 , or get something in between. When alpha>0 , the undistorted result is likely to have some black pixels corresponding to "virtual" pixels outside of the captured distorted image. The original camera intrinsic matrix, distortion coefficients, the computed new camera intrinsic matrix, and newImageSize should be passed to initUndistortRectifyMap to produce the maps for remap .

#### getOptimalNewCameraMatrix public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) getOptimalNewCameraMatrix([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI, boolean centerPrincipalPoint) Returns the new camera intrinsic matrix based on the free scaling parameter.Parameters:cameraMatrix - Input camera intrinsic matrix.distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.imageSize - Original image size.alpha - Free scaling parameter between 0 (when all the pixels in the undistorted image are valid) and 1 (when all the source image pixels are retained in the undistorted image). See stereoRectify for details.newImgSize - Image size after rectification. By default, it is set to imageSize .validPixROI - Optional output rectangle that outlines all-good-pixels region in the undistorted image. See roi1, roi2 description in stereoRectify .centerPrincipalPoint - Optional flag that indicates whether in the new camera intrinsic matrix the principal point should be at the image center or not. By default, the principal point is chosen to best fit a subset of the source image (determined by alpha) to the corrected image. Returns:new\_camera\_matrix Output new camera intrinsic matrix. The function computes and returns the optimal new camera intrinsic matrix based on the free scaling parameter. By varying this parameter, you may retrieve only sensible pixels alpha=0 , keep all the original image pixels if there is valuable information in the corners alpha=1 , or get something in between. When alpha>0 , the undistorted result is likely to have some black pixels corresponding to "virtual" pixels outside of the captured distorted image. The original camera intrinsic matrix, distortion coefficients, the computed new camera intrinsic matrix, and newImageSize should be passed to initUndistortRectifyMap to produce the maps for remap .

#### getValidDisparityROI public static [Rect](http://docs.google.com/org/opencv/core/Rect.html) getValidDisparityROI([Rect](http://docs.google.com/org/opencv/core/Rect.html) roi1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi2, int minDisparity, int numberOfDisparities, int blockSize)

#### initCameraMatrix2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) initCameraMatrix2D(java.util.List<[MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html)> objectPoints, java.util.List<[MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize) Finds an initial camera intrinsic matrix from 3D-2D point correspondences.Parameters:objectPoints - Vector of vectors of the calibration pattern points in the calibration pattern coordinate space. In the old interface all the per-view vectors are concatenated. See calibrateCamera for details.imagePoints - Vector of vectors of the projections of the calibration pattern points. In the old interface all the per-view vectors are concatenated.imageSize - Image size in pixels used to initialize the principal point. Otherwise, \(f\_x = f\_y \* \texttt{aspectRatio}\) . The function estimates and returns an initial camera intrinsic matrix for the camera calibration process. Currently, the function only supports planar calibration patterns, which are patterns where each object point has z-coordinate =0. Returns:automatically generated

#### initCameraMatrix2D public static [Mat](http://docs.google.com/org/opencv/core/Mat.html) initCameraMatrix2D(java.util.List<[MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html)> objectPoints, java.util.List<[MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html)> imagePoints, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, double aspectRatio) Finds an initial camera intrinsic matrix from 3D-2D point correspondences.Parameters:objectPoints - Vector of vectors of the calibration pattern points in the calibration pattern coordinate space. In the old interface all the per-view vectors are concatenated. See calibrateCamera for details.imagePoints - Vector of vectors of the projections of the calibration pattern points. In the old interface all the per-view vectors are concatenated.imageSize - Image size in pixels used to initialize the principal point.aspectRatio - If it is zero or negative, both \(f\_x\) and \(f\_y\) are estimated independently. Otherwise, \(f\_x = f\_y \* \texttt{aspectRatio}\) . The function estimates and returns an initial camera intrinsic matrix for the camera calibration process. Currently, the function only supports planar calibration patterns, which are patterns where each object point has z-coordinate =0. Returns:automatically generated

#### matMulDeriv public static void matMulDeriv([Mat](http://docs.google.com/org/opencv/core/Mat.html) A, [Mat](http://docs.google.com/org/opencv/core/Mat.html) B, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dABdA, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dABdB) Computes partial derivatives of the matrix product for each multiplied matrix.Parameters:A - First multiplied matrix.B - Second multiplied matrix.dABdA - First output derivative matrix d(A\\*B)/dA of size \(\texttt{A.rows\*B.cols} \times {A.rows\*A.cols}\) .dABdB - Second output derivative matrix d(A\\*B)/dB of size \(\texttt{A.rows\*B.cols} \times {B.rows\*B.cols}\) . The function computes partial derivatives of the elements of the matrix product \(A\*B\) with regard to the elements of each of the two input matrices. The function is used to compute the Jacobian matrices in stereoCalibrate but can also be used in any other similar optimization function.

#### projectPoints public static void projectPoints([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints) Projects 3D points to an image plane.Parameters:objectPoints - Array of object points expressed wrt. the world coordinate frame. A 3xN/Nx3 1-channel or 1xN/Nx1 3-channel (or vector<Point3f> ), where N is the number of points in the view.rvec - The rotation vector (REF: Rodrigues) that, together with tvec, performs a change of basis from world to camera coordinate system, see REF: calibrateCamera for details.tvec - The translation vector, see parameter description above.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\) . If the vector is empty, the zero distortion coefficients are assumed.imagePoints - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> . points with respect to components of the rotation vector, translation vector, focal lengths, coordinates of the principal point and the distortion coefficients. In the old interface different components of the jacobian are returned via different output parameters. function assumes that the aspect ratio (\(f\_x / f\_y\)) is fixed and correspondingly adjusts the jacobian matrix. The function computes the 2D projections of 3D points to the image plane, given intrinsic and extrinsic camera parameters. Optionally, the function computes Jacobians -matrices of partial derivatives of image points coordinates (as functions of all the input parameters) with respect to the particular parameters, intrinsic and/or extrinsic. The Jacobians are used during the global optimization in REF: calibrateCamera, REF: solvePnP, and REF: stereoCalibrate. The function itself can also be used to compute a re-projection error, given the current intrinsic and extrinsic parameters. **Note:** By setting rvec = tvec = \([0, 0, 0]\), or by setting cameraMatrix to a 3x3 identity matrix, or by passing zero distortion coefficients, one can get various useful partial cases of the function. This means, one can compute the distorted coordinates for a sparse set of points or apply a perspective transformation (and also compute the derivatives) in the ideal zero-distortion setup.

#### projectPoints public static void projectPoints([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian) Projects 3D points to an image plane.Parameters:objectPoints - Array of object points expressed wrt. the world coordinate frame. A 3xN/Nx3 1-channel or 1xN/Nx1 3-channel (or vector<Point3f> ), where N is the number of points in the view.rvec - The rotation vector (REF: Rodrigues) that, together with tvec, performs a change of basis from world to camera coordinate system, see REF: calibrateCamera for details.tvec - The translation vector, see parameter description above.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\) . If the vector is empty, the zero distortion coefficients are assumed.imagePoints - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .jacobian - Optional output 2Nx(10+<numDistCoeffs>) jacobian matrix of derivatives of image points with respect to components of the rotation vector, translation vector, focal lengths, coordinates of the principal point and the distortion coefficients. In the old interface different components of the jacobian are returned via different output parameters. function assumes that the aspect ratio (\(f\_x / f\_y\)) is fixed and correspondingly adjusts the jacobian matrix. The function computes the 2D projections of 3D points to the image plane, given intrinsic and extrinsic camera parameters. Optionally, the function computes Jacobians -matrices of partial derivatives of image points coordinates (as functions of all the input parameters) with respect to the particular parameters, intrinsic and/or extrinsic. The Jacobians are used during the global optimization in REF: calibrateCamera, REF: solvePnP, and REF: stereoCalibrate. The function itself can also be used to compute a re-projection error, given the current intrinsic and extrinsic parameters. **Note:** By setting rvec = tvec = \([0, 0, 0]\), or by setting cameraMatrix to a 3x3 identity matrix, or by passing zero distortion coefficients, one can get various useful partial cases of the function. This means, one can compute the distorted coordinates for a sparse set of points or apply a perspective transformation (and also compute the derivatives) in the ideal zero-distortion setup.

#### projectPoints public static void projectPoints([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian, double aspectRatio) Projects 3D points to an image plane.Parameters:objectPoints - Array of object points expressed wrt. the world coordinate frame. A 3xN/Nx3 1-channel or 1xN/Nx1 3-channel (or vector<Point3f> ), where N is the number of points in the view.rvec - The rotation vector (REF: Rodrigues) that, together with tvec, performs a change of basis from world to camera coordinate system, see REF: calibrateCamera for details.tvec - The translation vector, see parameter description above.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\) . If the vector is empty, the zero distortion coefficients are assumed.imagePoints - Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .jacobian - Optional output 2Nx(10+<numDistCoeffs>) jacobian matrix of derivatives of image points with respect to components of the rotation vector, translation vector, focal lengths, coordinates of the principal point and the distortion coefficients. In the old interface different components of the jacobian are returned via different output parameters.aspectRatio - Optional "fixed aspect ratio" parameter. If the parameter is not 0, the function assumes that the aspect ratio (\(f\_x / f\_y\)) is fixed and correspondingly adjusts the jacobian matrix. The function computes the 2D projections of 3D points to the image plane, given intrinsic and extrinsic camera parameters. Optionally, the function computes Jacobians -matrices of partial derivatives of image points coordinates (as functions of all the input parameters) with respect to the particular parameters, intrinsic and/or extrinsic. The Jacobians are used during the global optimization in REF: calibrateCamera, REF: solvePnP, and REF: stereoCalibrate. The function itself can also be used to compute a re-projection error, given the current intrinsic and extrinsic parameters. **Note:** By setting rvec = tvec = \([0, 0, 0]\), or by setting cameraMatrix to a 3x3 identity matrix, or by passing zero distortion coefficients, one can get various useful partial cases of the function. This means, one can compute the distorted coordinates for a sparse set of points or apply a perspective transformation (and also compute the derivatives) in the ideal zero-distortion setup.

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length. are feature points from cameras with same focal length and principal point. inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.focal - Focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point. inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.focal - Focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera. inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double focal, [Point](http://docs.google.com/org/opencv/core/Point.html) pp, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.focal - Focal length of the camera. Note that this function assumes that points1 and points2 are feature points from cameras with same focal length and principal point.pp - principal point of the camera.mask - Input/output mask for inliers in points1 and points2. If it is not empty, then it marks inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it computes camera intrinsic matrix from focal length and principal point: \(A = \begin{bmatrix} f & 0 & x\_{pp} \\ 0 & f & y\_{pp} \\ 0 & 0 & 1 \end{bmatrix}\) Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t) Recovers the relative camera rotation and the translation from an estimated essential matrix and the corresponding points in two images, using cheirality check. Returns the number of inliers that pass the check.Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter described below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length. inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function decomposes an essential matrix using REF: decomposeEssentialMat and then verifies possible pose hypotheses by doing cheirality check. The cheirality check means that the triangulated 3D points should have positive depth. Some details can be found in CITE: Nister03. This function can be used to process the output E and mask from REF: findEssentialMat. In this scenario, points1 and points2 are the same input for findEssentialMat.: // Example. Estimation of fundamental matrix using the RANSAC algorithm int point\_count = 100; vector<Point2f> points1(point\_count); vector<Point2f> points2(point\_count); // initialize the points here ... for( int i = 0; i < point\_count; i++ ) { points1[i] = ...; points2[i] = ...; } // cametra matrix with both focal lengths = 1, and principal point = (0, 0) Mat cameraMatrix = Mat::eye(3, 3, CV\_64F); Mat E, R, t, mask; E = findEssentialMat(points1, points2, cameraMatrix, RANSAC, 0.999, 1.0, mask); recoverPose(E, points1, points2, cameraMatrix, R, t, mask); Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.distanceThresh - threshold distance which is used to filter out far away points (i.e. infinite points). inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it outputs the triangulated 3D point that are used for the cheirality check. Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.distanceThresh - threshold distance which is used to filter out far away points (i.e. infinite points).mask - Input/output mask for inliers in points1 and points2. If it is not empty, then it marks inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function differs from the one above that it outputs the triangulated 3D point that are used for the cheirality check. Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, double distanceThresh, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask, [Mat](http://docs.google.com/org/opencv/core/Mat.html) triangulatedPoints)Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1.cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter description below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.distanceThresh - threshold distance which is used to filter out far away points (i.e. infinite points).mask - Input/output mask for inliers in points1 and points2. If it is not empty, then it marks inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check.triangulatedPoints - 3D points which were reconstructed by triangulation. This function differs from the one above that it outputs the triangulated 3D point that are used for the cheirality check. Returns:automatically generated

#### recoverPose public static int recoverPose([Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) t, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mask) Recovers the relative camera rotation and the translation from an estimated essential matrix and the corresponding points in two images, using cheirality check. Returns the number of inliers that pass the check.Parameters:E - The input essential matrix.points1 - Array of N 2D points from the first image. The point coordinates should be floating-point (single or double precision).points2 - Array of the second image points of the same size and format as points1 .cameraMatrix - Camera intrinsic matrix \(\cameramatrix{A}\) . Note that this function assumes that points1 and points2 are feature points from cameras with the same camera intrinsic matrix.R - Output rotation matrix. Together with the translation vector, this matrix makes up a tuple that performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Note that, in general, t can not be used for this tuple, see the parameter described below.t - Output translation vector. This vector is obtained by REF: decomposeEssentialMat and therefore is only known up to scale, i.e. t is the direction of the translation vector and has unit length.mask - Input/output mask for inliers in points1 and points2. If it is not empty, then it marks inliers in points1 and points2 for then given essential matrix E. Only these inliers will be used to recover pose. In the output mask only inliers which pass the cheirality check. This function decomposes an essential matrix using REF: decomposeEssentialMat and then verifies possible pose hypotheses by doing cheirality check. The cheirality check means that the triangulated 3D points should have positive depth. Some details can be found in CITE: Nister03. This function can be used to process the output E and mask from REF: findEssentialMat. In this scenario, points1 and points2 are the same input for findEssentialMat.: // Example. Estimation of fundamental matrix using the RANSAC algorithm int point\_count = 100; vector<Point2f> points1(point\_count); vector<Point2f> points2(point\_count); // initialize the points here ... for( int i = 0; i < point\_count; i++ ) { points1[i] = ...; points2[i] = ...; } // cametra matrix with both focal lengths = 1, and principal point = (0, 0) Mat cameraMatrix = Mat::eye(3, 3, CV\_64F); Mat E, R, t, mask; E = findEssentialMat(points1, points2, cameraMatrix, RANSAC, 0.999, 1.0, mask); recoverPose(E, points1, points2, cameraMatrix, R, t, mask); Returns:automatically generated

#### rectify3Collinear public static float rectify3Collinear([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs3, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imgpt1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imgpt3, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R12, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T12, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R13, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T13, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P3, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImgSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) roi2, int flags)

#### reprojectImageTo3D public static void reprojectImageTo3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q) Reprojects a disparity image to 3D space.Parameters:disparity - Input single-channel 8-bit unsigned, 16-bit signed, 32-bit signed or 32-bit floating-point disparity image. The values of 8-bit / 16-bit signed formats are assumed to have no fractional bits. If the disparity is 16-bit signed format, as computed by REF: StereoBM or REF: StereoSGBM and maybe other algorithms, it should be divided by 16 (and scaled to float) before being used here.\_3dImage - Output 3-channel floating-point image of the same size as disparity. Each element of \_3dImage(x,y) contains 3D coordinates of the point (x,y) computed from the disparity map. If one uses Q obtained by REF: stereoRectify, then the returned points are represented in the first camera's rectified coordinate system.Q - \(4 \times 4\) perspective transformation matrix that can be obtained with REF: stereoRectify. points where the disparity was not computed). If handleMissingValues=true, then pixels with the minimal disparity that corresponds to the outliers (see StereoMatcher::compute ) are transformed to 3D points with a very large Z value (currently set to 10000). depth. ddepth can also be set to CV\_16S, CV\_32S or CV\_32F. The function transforms a single-channel disparity map to a 3-channel image representing a 3D surface. That is, for each pixel (x,y) and the corresponding disparity d=disparity(x,y) , it computes: \(\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ \texttt{disparity} (x,y) \\ z \end{bmatrix}.\) SEE: To reproject a sparse set of points {(x,y,d),...} to 3D space, use perspectiveTransform.

#### reprojectImageTo3D public static void reprojectImageTo3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, boolean handleMissingValues) Reprojects a disparity image to 3D space.Parameters:disparity - Input single-channel 8-bit unsigned, 16-bit signed, 32-bit signed or 32-bit floating-point disparity image. The values of 8-bit / 16-bit signed formats are assumed to have no fractional bits. If the disparity is 16-bit signed format, as computed by REF: StereoBM or REF: StereoSGBM and maybe other algorithms, it should be divided by 16 (and scaled to float) before being used here.\_3dImage - Output 3-channel floating-point image of the same size as disparity. Each element of \_3dImage(x,y) contains 3D coordinates of the point (x,y) computed from the disparity map. If one uses Q obtained by REF: stereoRectify, then the returned points are represented in the first camera's rectified coordinate system.Q - \(4 \times 4\) perspective transformation matrix that can be obtained with REF: stereoRectify.handleMissingValues - Indicates, whether the function should handle missing values (i.e. points where the disparity was not computed). If handleMissingValues=true, then pixels with the minimal disparity that corresponds to the outliers (see StereoMatcher::compute ) are transformed to 3D points with a very large Z value (currently set to 10000). depth. ddepth can also be set to CV\_16S, CV\_32S or CV\_32F. The function transforms a single-channel disparity map to a 3-channel image representing a 3D surface. That is, for each pixel (x,y) and the corresponding disparity d=disparity(x,y) , it computes: \(\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ \texttt{disparity} (x,y) \\ z \end{bmatrix}.\) SEE: To reproject a sparse set of points {(x,y,d),...} to 3D space, use perspectiveTransform.

#### reprojectImageTo3D public static void reprojectImageTo3D([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) \_3dImage, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, boolean handleMissingValues, int ddepth) Reprojects a disparity image to 3D space.Parameters:disparity - Input single-channel 8-bit unsigned, 16-bit signed, 32-bit signed or 32-bit floating-point disparity image. The values of 8-bit / 16-bit signed formats are assumed to have no fractional bits. If the disparity is 16-bit signed format, as computed by REF: StereoBM or REF: StereoSGBM and maybe other algorithms, it should be divided by 16 (and scaled to float) before being used here.\_3dImage - Output 3-channel floating-point image of the same size as disparity. Each element of \_3dImage(x,y) contains 3D coordinates of the point (x,y) computed from the disparity map. If one uses Q obtained by REF: stereoRectify, then the returned points are represented in the first camera's rectified coordinate system.Q - \(4 \times 4\) perspective transformation matrix that can be obtained with REF: stereoRectify.handleMissingValues - Indicates, whether the function should handle missing values (i.e. points where the disparity was not computed). If handleMissingValues=true, then pixels with the minimal disparity that corresponds to the outliers (see StereoMatcher::compute ) are transformed to 3D points with a very large Z value (currently set to 10000).ddepth - The optional output array depth. If it is -1, the output image will have CV\_32F depth. ddepth can also be set to CV\_16S, CV\_32S or CV\_32F. The function transforms a single-channel disparity map to a 3-channel image representing a 3D surface. That is, for each pixel (x,y) and the corresponding disparity d=disparity(x,y) , it computes: \(\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ \texttt{disparity} (x,y) \\ z \end{bmatrix}.\) SEE: To reproject a sparse set of points {(x,y,d),...} to 3D space, use perspectiveTransform.

#### Rodrigues public static void Rodrigues([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst) Converts a rotation matrix to a rotation vector or vice versa.Parameters:src - Input rotation vector (3x1 or 1x3) or rotation matrix (3x3).dst - Output rotation matrix (3x3) or rotation vector (3x1 or 1x3), respectively. derivatives of the output array components with respect to the input array components. \(\begin{array}{l} \theta \leftarrow norm(r) \\ r \leftarrow r/ \theta \\ R = \cos(\theta) I + (1- \cos{\theta} ) r r^T + \sin(\theta) \vecthreethree{0}{-r\_z}{r\_y}{r\_z}{0}{-r\_x}{-r\_y}{r\_x}{0} \end{array}\) Inverse transformation can be also done easily, since \(\sin ( \theta ) \vecthreethree{0}{-r\_z}{r\_y}{r\_z}{0}{-r\_x}{-r\_y}{r\_x}{0} = \frac{R - R^T}{2}\) A rotation vector is a convenient and most compact representation of a rotation matrix (since any rotation matrix has just 3 degrees of freedom). The representation is used in the global 3D geometry optimization procedures like REF: calibrateCamera, REF: stereoCalibrate, or REF: solvePnP . **Note:** More information about the computation of the derivative of a 3D rotation matrix with respect to its exponential coordinate can be found in:

* + A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates, Guillermo Gallego, Anthony J. Yezzi CITE: Gallego2014ACF

**Note:** Useful information on SE(3) and Lie Groups can be found in:

* + A tutorial on SE(3) transformation parameterizations and on-manifold optimization, Jose-Luis Blanco CITE: blanco2010tutorial
  + Lie Groups for 2D and 3D Transformation, Ethan Eade CITE: Eade17
  + A micro Lie theory for state estimation in robotics, Joan Solà, Jérémie Deray, Dinesh Atchuthan CITE: Sol2018AML

#### Rodrigues public static void Rodrigues([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) dst, [Mat](http://docs.google.com/org/opencv/core/Mat.html) jacobian) Converts a rotation matrix to a rotation vector or vice versa.Parameters:src - Input rotation vector (3x1 or 1x3) or rotation matrix (3x3).dst - Output rotation matrix (3x3) or rotation vector (3x1 or 1x3), respectively.jacobian - Optional output Jacobian matrix, 3x9 or 9x3, which is a matrix of partial derivatives of the output array components with respect to the input array components. \(\begin{array}{l} \theta \leftarrow norm(r) \\ r \leftarrow r/ \theta \\ R = \cos(\theta) I + (1- \cos{\theta} ) r r^T + \sin(\theta) \vecthreethree{0}{-r\_z}{r\_y}{r\_z}{0}{-r\_x}{-r\_y}{r\_x}{0} \end{array}\) Inverse transformation can be also done easily, since \(\sin ( \theta ) \vecthreethree{0}{-r\_z}{r\_y}{r\_z}{0}{-r\_x}{-r\_y}{r\_x}{0} = \frac{R - R^T}{2}\) A rotation vector is a convenient and most compact representation of a rotation matrix (since any rotation matrix has just 3 degrees of freedom). The representation is used in the global 3D geometry optimization procedures like REF: calibrateCamera, REF: stereoCalibrate, or REF: solvePnP . **Note:** More information about the computation of the derivative of a 3D rotation matrix with respect to its exponential coordinate can be found in:

* + A Compact Formula for the Derivative of a 3-D Rotation in Exponential Coordinates, Guillermo Gallego, Anthony J. Yezzi CITE: Gallego2014ACF

**Note:** Useful information on SE(3) and Lie Groups can be found in:

* + A tutorial on SE(3) transformation parameterizations and on-manifold optimization, Jose-Luis Blanco CITE: blanco2010tutorial
  + Lie Groups for 2D and 3D Transformation, Ethan Eade CITE: Eade17
  + A micro Lie theory for state estimation in robotics, Joan Solà, Jérémie Deray, Dinesh Atchuthan CITE: Sol2018AML

#### RQDecomp3x3 public static double[] RQDecomp3x3([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ) Computes an RQ decomposition of 3x3 matrices.Parameters:src - 3x3 input matrix.mtxR - Output 3x3 upper-triangular matrix.mtxQ - Output 3x3 orthogonal matrix. The function computes a RQ decomposition using the given rotations. This function is used in decomposeProjectionMatrix to decompose the left 3x3 submatrix of a projection matrix into a camera and a rotation matrix. It optionally returns three rotation matrices, one for each axis, and the three Euler angles in degrees (as the return value) that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. Returns:automatically generated

#### RQDecomp3x3 public static double[] RQDecomp3x3([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx) Computes an RQ decomposition of 3x3 matrices.Parameters:src - 3x3 input matrix.mtxR - Output 3x3 upper-triangular matrix.mtxQ - Output 3x3 orthogonal matrix.Qx - Optional output 3x3 rotation matrix around x-axis. The function computes a RQ decomposition using the given rotations. This function is used in decomposeProjectionMatrix to decompose the left 3x3 submatrix of a projection matrix into a camera and a rotation matrix. It optionally returns three rotation matrices, one for each axis, and the three Euler angles in degrees (as the return value) that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. Returns:automatically generated

#### RQDecomp3x3 public static double[] RQDecomp3x3([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qy) Computes an RQ decomposition of 3x3 matrices.Parameters:src - 3x3 input matrix.mtxR - Output 3x3 upper-triangular matrix.mtxQ - Output 3x3 orthogonal matrix.Qx - Optional output 3x3 rotation matrix around x-axis.Qy - Optional output 3x3 rotation matrix around y-axis. The function computes a RQ decomposition using the given rotations. This function is used in decomposeProjectionMatrix to decompose the left 3x3 submatrix of a projection matrix into a camera and a rotation matrix. It optionally returns three rotation matrices, one for each axis, and the three Euler angles in degrees (as the return value) that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. Returns:automatically generated

#### RQDecomp3x3 public static double[] RQDecomp3x3([Mat](http://docs.google.com/org/opencv/core/Mat.html) src, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxR, [Mat](http://docs.google.com/org/opencv/core/Mat.html) mtxQ, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qx, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qy, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Qz) Computes an RQ decomposition of 3x3 matrices.Parameters:src - 3x3 input matrix.mtxR - Output 3x3 upper-triangular matrix.mtxQ - Output 3x3 orthogonal matrix.Qx - Optional output 3x3 rotation matrix around x-axis.Qy - Optional output 3x3 rotation matrix around y-axis.Qz - Optional output 3x3 rotation matrix around z-axis. The function computes a RQ decomposition using the given rotations. This function is used in decomposeProjectionMatrix to decompose the left 3x3 submatrix of a projection matrix into a camera and a rotation matrix. It optionally returns three rotation matrices, one for each axis, and the three Euler angles in degrees (as the return value) that could be used in OpenGL. Note, there is always more than one sequence of rotations about the three principal axes that results in the same orientation of an object, e.g. see CITE: Slabaugh . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions. Returns:automatically generated

#### sampsonDistance public static double sampsonDistance([Mat](http://docs.google.com/org/opencv/core/Mat.html) pt1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) pt2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F) Calculates the Sampson Distance between two points. The function cv::sampsonDistance calculates and returns the first order approximation of the geometric error as: \( sd( \texttt{pt1} , \texttt{pt2} )= \frac{(\texttt{pt2}^t \cdot \texttt{F} \cdot \texttt{pt1})^2} {((\texttt{F} \cdot \texttt{pt1})(0))^2 + ((\texttt{F} \cdot \texttt{pt1})(1))^2 + ((\texttt{F}^t \cdot \texttt{pt2})(0))^2 + ((\texttt{F}^t \cdot \texttt{pt2})(1))^2} \) The fundamental matrix may be calculated using the cv::findFundamentalMat function. See CITE: HartleyZ00 11.4.3 for details.Parameters:pt1 - first homogeneous 2d pointpt2 - second homogeneous 2d pointF - fundamental matrix Returns:The computed Sampson distance.

#### solveP3P public static int solveP3P([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, int flags) Finds an object pose from 3 3D-2D point correspondences.Parameters:objectPoints - Array of object points in the object coordinate space, 3x3 1-channel or 1x3/3x1 3-channel. vector<Point3f> can be also passed here.imagePoints - Array of corresponding image points, 3x2 1-channel or 1x3/3x1 2-channel. vector<Point2f> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system. A P3P problem has up to 4 solutions.tvecs - Output translation vectors.flags - Method for solving a P3P problem:

* + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete).
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke and S. Roumeliotis. "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17).

The function estimates the object pose given 3 object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. **Note:** The solutions are sorted by reprojection errors (lowest to highest). Returns:automatically generated

#### solvePnP public static boolean solvePnP([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec)

Finds an object pose from 3D-2D point correspondences. This function returns the rotation and the translation vectors that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame, using different methods:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): need 4 input points to return a unique solution.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector. the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F. Moreno-Noguer, V. Lepetit and P. Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of J. Hesch and S. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A. Penate-Sanchez, J. Andrade-Cetto, F. Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + REF: SOLVEPNP\_SQPNP Method is based on the paper "A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem" by G. Terzakis and M.Lourakis (CITE: Terzakis20). It requires 3 or more points.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
    - With REF: SOLVEPNP\_SQPNP input points must be >= 3

Returns:automatically generated

#### solvePnP public static boolean solvePnP([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess)

Finds an object pose from 3D-2D point correspondences. This function returns the rotation and the translation vectors that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame, using different methods:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): need 4 input points to return a unique solution.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F. Moreno-Noguer, V. Lepetit and P. Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of J. Hesch and S. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A. Penate-Sanchez, J. Andrade-Cetto, F. Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + REF: SOLVEPNP\_SQPNP Method is based on the paper "A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem" by G. Terzakis and M.Lourakis (CITE: Terzakis20). It requires 3 or more points.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
    - With REF: SOLVEPNP\_SQPNP input points must be >= 3

Returns:automatically generated

#### solvePnP public static boolean solvePnP([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int flags)

Finds an object pose from 3D-2D point correspondences. This function returns the rotation and the translation vectors that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame, using different methods:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): need 4 input points to return a unique solution.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.flags - Method for solving a PnP problem:

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F. Moreno-Noguer, V. Lepetit and P. Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of J. Hesch and S. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A. Penate-Sanchez, J. Andrade-Cetto, F. Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + REF: SOLVEPNP\_SQPNP Method is based on the paper "A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem" by G. Terzakis and M.Lourakis (CITE: Terzakis20). It requires 3 or more points.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
    - With REF: SOLVEPNP\_SQPNP input points must be >= 3

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors. the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0] and useExtrinsicGuess is set to true. and useExtrinsicGuess is set to true. (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward).![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0] and useExtrinsicGuess is set to true. and useExtrinsicGuess is set to true. (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward).![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.flags - Method for solving a PnP problem:

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0] and useExtrinsicGuess is set to true. and useExtrinsicGuess is set to true. (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.

The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward).![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* + An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
  + If you are using Python:
    - Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
    - The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
    - Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
  + The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
  + The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
  + With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
  + With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
  + With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.flags - Method for solving a PnP problem:

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

rvec - Rotation vector used to initialize an iterative PnP refinement algorithm, when flag is REF: SOLVEPNP\_ITERATIVE and useExtrinsicGuess is set to true. and useExtrinsicGuess is set to true. (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
* If you are using Python:
  + Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
  + The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
  + Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
* The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
* The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
* With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
* With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
* With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
  + point 0: [-squareLength / 2, squareLength / 2, 0]
  + point 1: [ squareLength / 2, squareLength / 2, 0]
  + point 2: [ squareLength / 2, -squareLength / 2, 0]
  + point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.flags - Method for solving a PnP problem:

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

rvec - Rotation vector used to initialize an iterative PnP refinement algorithm, when flag is REF: SOLVEPNP\_ITERATIVE and useExtrinsicGuess is set to true.tvec - Translation vector used to initialize an iterative PnP refinement algorithm, when flag is REF: SOLVEPNP\_ITERATIVE and useExtrinsicGuess is set to true. (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
* If you are using Python:
  + Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
  + The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
  + Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
* The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
* The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
* With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
* With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
* With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
  + point 0: [-squareLength / 2, squareLength / 2, 0]
  + point 1: [ squareLength / 2, squareLength / 2, 0]
  + point 2: [ squareLength / 2, -squareLength / 2, 0]
  + point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPGeneric public static int solvePnPGeneric([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> rvecs, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> tvecs, boolean useExtrinsicGuess, int flags, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) reprojectionError)

Finds an object pose from 3D-2D point correspondences. This function returns a list of all the possible solutions (a solution is a <rotation vector, translation vector> couple), depending on the number of input points and the chosen method:

* + P3P methods (REF: SOLVEPNP\_P3P, REF: SOLVEPNP\_AP3P): 3 or 4 input points. Number of returned solutions can be between 0 and 4 with 3 input points.
  + REF: SOLVEPNP\_IPPE Input points must be >= 4 and object points must be coplanar. Returns 2 solutions.
  + REF: SOLVEPNP\_IPPE\_SQUARE Special case suitable for marker pose estimation. Number of input points must be 4 and 2 solutions are returned. Object points must be defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]
  + for all the other flags, number of input points must be >= 4 and object points can be in any configuration. Only 1 solution is returned.

Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvecs - Vector of output rotation vectors (see REF: Rodrigues ) that, together with tvecs, brings points from the model coordinate system to the camera coordinate system.tvecs - Vector of output translation vectors.useExtrinsicGuess - Parameter used for #SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.flags - Method for solving a PnP problem:

* + REF: SOLVEPNP\_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection error, that is the sum of squared distances between the observed projections imagePoints and the projected (using projectPoints ) objectPoints .
  + REF: SOLVEPNP\_P3P Method is based on the paper of X.S. Gao, X.-R. Hou, J. Tang, H.-F. Chang "Complete Solution Classification for the Perspective-Three-Point Problem" (CITE: gao2003complete). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_AP3P Method is based on the paper of T. Ke, S. Roumeliotis "An Efficient Algebraic Solution to the Perspective-Three-Point Problem" (CITE: Ke17). In this case the function requires exactly four object and image points.
  + REF: SOLVEPNP\_EPNP Method has been introduced by F.Moreno-Noguer, V.Lepetit and P.Fua in the paper "EPnP: Efficient Perspective-n-Point Camera Pose Estimation" (CITE: lepetit2009epnp).
  + REF: SOLVEPNP\_DLS **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of Joel A. Hesch and Stergios I. Roumeliotis. "A Direct Least-Squares (DLS) Method for PnP" (CITE: hesch2011direct).
  + REF: SOLVEPNP\_UPNP **Broken implementation. Using this flag will fallback to EPnP.** \n Method is based on the paper of A.Penate-Sanchez, J.Andrade-Cetto, F.Moreno-Noguer. "Exhaustive Linearization for Robust Camera Pose and Focal Length Estimation" (CITE: penate2013exhaustive). In this case the function also estimates the parameters \(f\_x\) and \(f\_y\) assuming that both have the same value. Then the cameraMatrix is updated with the estimated focal length.
  + REF: SOLVEPNP\_IPPE Method is based on the paper of T. Collins and A. Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method requires coplanar object points.
  + REF: SOLVEPNP\_IPPE\_SQUARE Method is based on the paper of Toby Collins and Adrien Bartoli. "Infinitesimal Plane-Based Pose Estimation" (CITE: Collins14). This method is suitable for marker pose estimation. It requires 4 coplanar object points defined in the following order:
    - point 0: [-squareLength / 2, squareLength / 2, 0]
    - point 1: [ squareLength / 2, squareLength / 2, 0]
    - point 2: [ squareLength / 2, -squareLength / 2, 0]
    - point 3: [-squareLength / 2, -squareLength / 2, 0]

rvec - Rotation vector used to initialize an iterative PnP refinement algorithm, when flag is REF: SOLVEPNP\_ITERATIVE and useExtrinsicGuess is set to true.tvec - Translation vector used to initialize an iterative PnP refinement algorithm, when flag is REF: SOLVEPNP\_ITERATIVE and useExtrinsicGuess is set to true.reprojectionError - Optional vector of reprojection error, that is the RMS error (\( \text{RMSE} = \sqrt{\frac{\sum\_{i}^{N} \left ( \hat{y\_i} - y\_i \right )^2}{N}} \)) between the input image points and the 3D object points projected with the estimated pose.The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients, see the figure below (more precisely, the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward). ![](pnp.jpg) Points expressed in the world frame \( \bf{X}\_w \) are projected into the image plane \( \left[ u, v \right] \) using the perspective projection model \( \Pi \) and the camera intrinsic parameters matrix \( \bf{A} \): \( \begin{align\*} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \bf{A} \hspace{0.1em} \Pi \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f\_x & 0 & c\_x \\ 0 & f\_y & c\_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) The estimated pose is thus the rotation (rvec) and the translation (tvec) vectors that allow transforming a 3D point expressed in the world frame into the camera frame: \( \begin{align\*} \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \hspace{0.2em} ^{c}\bf{T}\_w \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \\ \begin{bmatrix} X\_c \\ Y\_c \\ Z\_c \\ 1 \end{bmatrix} &= \begin{bmatrix} r\_{11} & r\_{12} & r\_{13} & t\_x \\ r\_{21} & r\_{22} & r\_{23} & t\_y \\ r\_{31} & r\_{32} & r\_{33} & t\_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X\_{w} \\ Y\_{w} \\ Z\_{w} \\ 1 \end{bmatrix} \end{align\*} \) **Note:**

* An example of how to use solvePnP for planar augmented reality can be found at opencv\_source\_code/samples/python/plane\_ar.py
* If you are using Python:
  + Numpy array slices won't work as input because solvePnP requires contiguous arrays (enforced by the assertion using cv::Mat::checkVector() around line 55 of modules/calib3d/src/solvepnp.cpp version 2.4.9)
  + The P3P algorithm requires image points to be in an array of shape (N,1,2) due to its calling of cv::undistortPoints (around line 75 of modules/calib3d/src/solvepnp.cpp version 2.4.9) which requires 2-channel information.
  + Thus, given some data D = np.array(...) where D.shape = (N,M), in order to use a subset of it as, e.g., imagePoints, one must effectively copy it into a new array: imagePoints = np.ascontiguousarray(D[:,:2]).reshape((N,1,2))
* The methods REF: SOLVEPNP\_DLS and REF: SOLVEPNP\_UPNP cannot be used as the current implementations are unstable and sometimes give completely wrong results. If you pass one of these two flags, REF: SOLVEPNP\_EPNP method will be used instead.
* The minimum number of points is 4 in the general case. In the case of REF: SOLVEPNP\_P3P and REF: SOLVEPNP\_AP3P methods, it is required to use exactly 4 points (the first 3 points are used to estimate all the solutions of the P3P problem, the last one is used to retain the best solution that minimizes the reprojection error).
* With REF: SOLVEPNP\_ITERATIVE method and useExtrinsicGuess=true, the minimum number of points is 3 (3 points are sufficient to compute a pose but there are up to 4 solutions). The initial solution should be close to the global solution to converge.
* With REF: SOLVEPNP\_IPPE input points must be >= 4 and object points must be coplanar.
* With REF: SOLVEPNP\_IPPE\_SQUARE this is a special case suitable for marker pose estimation. Number of input points must be 4. Object points must be defined in the following order:
  + point 0: [-squareLength / 2, squareLength / 2, 0]
  + point 1: [ squareLength / 2, squareLength / 2, 0]
  + point 2: [ squareLength / 2, -squareLength / 2, 0]
  + point 3: [-squareLength / 2, -squareLength / 2, 0]

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector. the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them. is the maximum allowed distance between the observed and computed point projections to consider it an inlier. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them. is the maximum allowed distance between the observed and computed point projections to consider it an inlier. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.iterationsCount - Number of iterations. is the maximum allowed distance between the observed and computed point projections to consider it an inlier. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.iterationsCount - Number of iterations.reprojectionError - Inlier threshold value used by the RANSAC procedure. The parameter value is the maximum allowed distance between the observed and computed point projections to consider it an inlier. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.iterationsCount - Number of iterations.reprojectionError - Inlier threshold value used by the RANSAC procedure. The parameter value is the maximum allowed distance between the observed and computed point projections to consider it an inlier.confidence - The probability that the algorithm produces a useful result. The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.iterationsCount - Number of iterations.reprojectionError - Inlier threshold value used by the RANSAC procedure. The parameter value is the maximum allowed distance between the observed and computed point projections to consider it an inlier.confidence - The probability that the algorithm produces a useful result.inliers - Output vector that contains indices of inliers in objectPoints and imagePoints . The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRansac public static boolean solvePnPRansac([MatOfPoint3f](http://docs.google.com/org/opencv/core/MatOfPoint3f.html) objectPoints, [MatOfPoint2f](http://docs.google.com/org/opencv/core/MatOfPoint2f.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [MatOfDouble](http://docs.google.com/org/opencv/core/MatOfDouble.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, boolean useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, [Mat](http://docs.google.com/org/opencv/core/Mat.html) inliers, int flags) Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can be also passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can be also passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system.tvec - Output translation vector.useExtrinsicGuess - Parameter used for REF: SOLVEPNP\_ITERATIVE. If true (1), the function uses the provided rvec and tvec values as initial approximations of the rotation and translation vectors, respectively, and further optimizes them.iterationsCount - Number of iterations.reprojectionError - Inlier threshold value used by the RANSAC procedure. The parameter value is the maximum allowed distance between the observed and computed point projections to consider it an inlier.confidence - The probability that the algorithm produces a useful result.inliers - Output vector that contains indices of inliers in objectPoints and imagePoints .flags - Method for solving a PnP problem (see REF: solvePnP ). The function estimates an object pose given a set of object points, their corresponding image projections, as well as the camera intrinsic matrix and the distortion coefficients. This function finds such a pose that minimizes reprojection error, that is, the sum of squared distances between the observed projections imagePoints and the projected (using REF: projectPoints ) objectPoints. The use of RANSAC makes the function resistant to outliers. **Note:**

* + An example of how to use solvePNPRansac for object detection can be found at opencv\_source\_code/samples/cpp/tutorial\_code/calib3d/real\_time\_pose\_estimation/
  + The default method used to estimate the camera pose for the Minimal Sample Sets step is #SOLVEPNP\_EPNP. Exceptions are:
    - if you choose #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P, these methods will be used.
    - if the number of input points is equal to 4, #SOLVEPNP\_P3P is used.
  + The method used to estimate the camera pose using all the inliers is defined by the flags parameters unless it is equal to #SOLVEPNP\_P3P or #SOLVEPNP\_AP3P. In this case, the method #SOLVEPNP\_EPNP will be used instead.

Returns:automatically generated

#### solvePnPRefineLM public static void solvePnPRefineLM([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can also be passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can also be passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Input/Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system. Input values are used as an initial solution.tvec - Input/Output translation vector. Input values are used as an initial solution. The function refines the object pose given at least 3 object points, their corresponding image projections, an initial solution for the rotation and translation vector, as well as the camera intrinsic matrix and the distortion coefficients. The function minimizes the projection error with respect to the rotation and the translation vectors, according to a Levenberg-Marquardt iterative minimization CITE: Madsen04 CITE: Eade13 process.

#### solvePnPRefineLM public static void solvePnPRefineLM([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can also be passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can also be passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Input/Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system. Input values are used as an initial solution.tvec - Input/Output translation vector. Input values are used as an initial solution.criteria - Criteria when to stop the Levenberg-Marquard iterative algorithm. The function refines the object pose given at least 3 object points, their corresponding image projections, an initial solution for the rotation and translation vector, as well as the camera intrinsic matrix and the distortion coefficients. The function minimizes the projection error with respect to the rotation and the translation vectors, according to a Levenberg-Marquardt iterative minimization CITE: Madsen04 CITE: Eade13 process.

#### solvePnPRefineVVS public static void solvePnPRefineVVS([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can also be passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can also be passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Input/Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system. Input values are used as an initial solution.tvec - Input/Output translation vector. Input values are used as an initial solution. gain in the Damped Gauss-Newton formulation. The function refines the object pose given at least 3 object points, their corresponding image projections, an initial solution for the rotation and translation vector, as well as the camera intrinsic matrix and the distortion coefficients. The function minimizes the projection error with respect to the rotation and the translation vectors, using a virtual visual servoing (VVS) CITE: Chaumette06 CITE: Marchand16 scheme.

#### solvePnPRefineVVS public static void solvePnPRefineVVS([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can also be passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can also be passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Input/Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system. Input values are used as an initial solution.tvec - Input/Output translation vector. Input values are used as an initial solution.criteria - Criteria when to stop the Levenberg-Marquard iterative algorithm. gain in the Damped Gauss-Newton formulation. The function refines the object pose given at least 3 object points, their corresponding image projections, an initial solution for the rotation and translation vector, as well as the camera intrinsic matrix and the distortion coefficients. The function minimizes the projection error with respect to the rotation and the translation vectors, using a virtual visual servoing (VVS) CITE: Chaumette06 CITE: Marchand16 scheme.

#### solvePnPRefineVVS public static void solvePnPRefineVVS([Mat](http://docs.google.com/org/opencv/core/Mat.html) objectPoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) imagePoints, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs, [Mat](http://docs.google.com/org/opencv/core/Mat.html) rvec, [Mat](http://docs.google.com/org/opencv/core/Mat.html) tvec, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria, double VVSlambda) Refine a pose (the translation and the rotation that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame) from a 3D-2D point correspondences and starting from an initial solution.Parameters:objectPoints - Array of object points in the object coordinate space, Nx3 1-channel or 1xN/Nx1 3-channel, where N is the number of points. vector<Point3d> can also be passed here.imagePoints - Array of corresponding image points, Nx2 1-channel or 1xN/Nx1 2-channel, where N is the number of points. vector<Point2d> can also be passed here.cameraMatrix - Input camera intrinsic matrix \(\cameramatrix{A}\) .distCoeffs - Input vector of distortion coefficients \(\distcoeffs\). If the vector is NULL/empty, the zero distortion coefficients are assumed.rvec - Input/Output rotation vector (see REF: Rodrigues ) that, together with tvec, brings points from the model coordinate system to the camera coordinate system. Input values are used as an initial solution.tvec - Input/Output translation vector. Input values are used as an initial solution.criteria - Criteria when to stop the Levenberg-Marquard iterative algorithm.VVSlambda - Gain for the virtual visual servoing control law, equivalent to the \(\alpha\) gain in the Damped Gauss-Newton formulation. The function refines the object pose given at least 3 object points, their corresponding image projections, an initial solution for the rotation and translation vector, as well as the camera intrinsic matrix and the distortion coefficients. The function minimizes the projection error with respect to the rotation and the translation vectors, using a virtual visual servoing (VVS) CITE: Chaumette06 CITE: Marchand16 scheme.

#### stereoCalibrate public static double stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F)

#### stereoCalibrate public static double stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, int flags)

#### stereoCalibrate public static double stereoCalibrate(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria)

#### stereoCalibrateExtended public static double stereoCalibrateExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors) Calibrates a stereo camera set up. This function finds the intrinsic parameters for each of the two cameras and the extrinsic parameters between the two cameras.Parameters:objectPoints - Vector of vectors of the calibration pattern points. The same structure as in REF: calibrateCamera. For each pattern view, both cameras need to see the same object points. Therefore, objectPoints.size(), imagePoints1.size(), and imagePoints2.size() need to be equal as well as objectPoints[i].size(), imagePoints1[i].size(), and imagePoints2[i].size() need to be equal for each i.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera. The same structure as in REF: calibrateCamera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera. The same structure as in REF: calibrateCamera.cameraMatrix1 - Input/output camera intrinsic matrix for the first camera, the same as in REF: calibrateCamera. Furthermore, for the stereo case, additional flags may be used, see below.distCoeffs1 - Input/output vector of distortion coefficients, the same as in REF: calibrateCamera.cameraMatrix2 - Input/output second camera intrinsic matrix for the second camera. See description for cameraMatrix1.distCoeffs2 - Input/output lens distortion coefficients for the second camera. See description for distCoeffs1.imageSize - Size of the image used only to initialize the camera intrinsic matrices.R - Output rotation matrix. Together with the translation vector T, this matrix brings points given in the first camera's coordinate system to points in the second camera's coordinate system. In more technical terms, the tuple of R and T performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Due to its duality, this tuple is equivalent to the position of the first camera with respect to the second camera coordinate system.T - Output translation vector, see description above.E - Output essential matrix.F - Output fundamental matrix.perViewErrors - Output vector of the RMS re-projection error estimated for each pattern view.

* + REF: CALIB\_FIX\_INTRINSIC Fix cameraMatrix? and distCoeffs? so that only R, T, E, and F matrices are estimated.
  + REF: CALIB\_USE\_INTRINSIC\_GUESS Optimize some or all of the intrinsic parameters according to the specified flags. Initial values are provided by the user.
  + REF: CALIB\_USE\_EXTRINSIC\_GUESS R and T contain valid initial values that are optimized further. Otherwise R and T are initialized to the median value of the pattern views (each dimension separately).
  + REF: CALIB\_FIX\_PRINCIPAL\_POINT Fix the principal points during the optimization.
  + REF: CALIB\_FIX\_FOCAL\_LENGTH Fix \(f^{(j)}\_x\) and \(f^{(j)}\_y\) .
  + REF: CALIB\_FIX\_ASPECT\_RATIO Optimize \(f^{(j)}\_y\) . Fix the ratio \(f^{(j)}\_x/f^{(j)}\_y\) .
  + REF: CALIB\_SAME\_FOCAL\_LENGTH Enforce \(f^{(0)}\_x=f^{(1)}\_x\) and \(f^{(0)}\_y=f^{(1)}\_y\) .
  + REF: CALIB\_ZERO\_TANGENT\_DIST Set tangential distortion coefficients for each camera to zeros and fix there.
  + REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 Do not change the corresponding radial distortion coefficient during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_RATIONAL\_MODEL Enable coefficients k4, k5, and k6. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.

The function estimates the transformation between two cameras making a stereo pair. If one computes the poses of an object relative to the first camera and to the second camera, ( \(R\_1\),\(T\_1\) ) and (\(R\_2\),\(T\_2\)), respectively, for a stereo camera where the relative position and orientation between the two cameras are fixed, then those poses definitely relate to each other. This means, if the relative position and orientation (\(R\),\(T\)) of the two cameras is known, it is possible to compute (\(R\_2\),\(T\_2\)) when (\(R\_1\),\(T\_1\)) is given. This is what the described function does. It computes (\(R\),\(T\)) such that: \(R\_2=R R\_1\) \(T\_2=R T\_1 + T.\) Therefore, one can compute the coordinate representation of a 3D point for the second camera's coordinate system when given the point's coordinate representation in the first camera's coordinate system: \(\begin{bmatrix} X\_2 \\ Y\_2 \\ Z\_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X\_1 \\ Y\_1 \\ Z\_1 \\ 1 \end{bmatrix}.\) Optionally, it computes the essential matrix E: \(E= \vecthreethree{0}{-T\_2}{T\_1}{T\_2}{0}{-T\_0}{-T\_1}{T\_0}{0} R\) where \(T\_i\) are components of the translation vector \(T\) : \(T=[T\_0, T\_1, T\_2]^T\) . And the function can also compute the fundamental matrix F: \(F = cameraMatrix2^{-T}\cdot E \cdot cameraMatrix1^{-1}\) Besides the stereo-related information, the function can also perform a full calibration of each of the two cameras. However, due to the high dimensionality of the parameter space and noise in the input data, the function can diverge from the correct solution. If the intrinsic parameters can be estimated with high accuracy for each of the cameras individually (for example, using calibrateCamera ), you are recommended to do so and then pass REF: CALIB\_FIX\_INTRINSIC flag to the function along with the computed intrinsic parameters. Otherwise, if all the parameters are estimated at once, it makes sense to restrict some parameters, for example, pass REF: CALIB\_SAME\_FOCAL\_LENGTH and REF: CALIB\_ZERO\_TANGENT\_DIST flags, which is usually a reasonable assumption. Similarly to calibrateCamera, the function minimizes the total re-projection error for all the points in all the available views from both cameras. The function returns the final value of the re-projection error. Returns:automatically generated

#### stereoCalibrateExtended public static double stereoCalibrateExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags) Calibrates a stereo camera set up. This function finds the intrinsic parameters for each of the two cameras and the extrinsic parameters between the two cameras.Parameters:objectPoints - Vector of vectors of the calibration pattern points. The same structure as in REF: calibrateCamera. For each pattern view, both cameras need to see the same object points. Therefore, objectPoints.size(), imagePoints1.size(), and imagePoints2.size() need to be equal as well as objectPoints[i].size(), imagePoints1[i].size(), and imagePoints2[i].size() need to be equal for each i.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera. The same structure as in REF: calibrateCamera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera. The same structure as in REF: calibrateCamera.cameraMatrix1 - Input/output camera intrinsic matrix for the first camera, the same as in REF: calibrateCamera. Furthermore, for the stereo case, additional flags may be used, see below.distCoeffs1 - Input/output vector of distortion coefficients, the same as in REF: calibrateCamera.cameraMatrix2 - Input/output second camera intrinsic matrix for the second camera. See description for cameraMatrix1.distCoeffs2 - Input/output lens distortion coefficients for the second camera. See description for distCoeffs1.imageSize - Size of the image used only to initialize the camera intrinsic matrices.R - Output rotation matrix. Together with the translation vector T, this matrix brings points given in the first camera's coordinate system to points in the second camera's coordinate system. In more technical terms, the tuple of R and T performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Due to its duality, this tuple is equivalent to the position of the first camera with respect to the second camera coordinate system.T - Output translation vector, see description above.E - Output essential matrix.F - Output fundamental matrix.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:

* + REF: CALIB\_FIX\_INTRINSIC Fix cameraMatrix? and distCoeffs? so that only R, T, E, and F matrices are estimated.
  + REF: CALIB\_USE\_INTRINSIC\_GUESS Optimize some or all of the intrinsic parameters according to the specified flags. Initial values are provided by the user.
  + REF: CALIB\_USE\_EXTRINSIC\_GUESS R and T contain valid initial values that are optimized further. Otherwise R and T are initialized to the median value of the pattern views (each dimension separately).
  + REF: CALIB\_FIX\_PRINCIPAL\_POINT Fix the principal points during the optimization.
  + REF: CALIB\_FIX\_FOCAL\_LENGTH Fix \(f^{(j)}\_x\) and \(f^{(j)}\_y\) .
  + REF: CALIB\_FIX\_ASPECT\_RATIO Optimize \(f^{(j)}\_y\) . Fix the ratio \(f^{(j)}\_x/f^{(j)}\_y\) .
  + REF: CALIB\_SAME\_FOCAL\_LENGTH Enforce \(f^{(0)}\_x=f^{(1)}\_x\) and \(f^{(0)}\_y=f^{(1)}\_y\) .
  + REF: CALIB\_ZERO\_TANGENT\_DIST Set tangential distortion coefficients for each camera to zeros and fix there.
  + REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 Do not change the corresponding radial distortion coefficient during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_RATIONAL\_MODEL Enable coefficients k4, k5, and k6. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.

The function estimates the transformation between two cameras making a stereo pair. If one computes the poses of an object relative to the first camera and to the second camera, ( \(R\_1\),\(T\_1\) ) and (\(R\_2\),\(T\_2\)), respectively, for a stereo camera where the relative position and orientation between the two cameras are fixed, then those poses definitely relate to each other. This means, if the relative position and orientation (\(R\),\(T\)) of the two cameras is known, it is possible to compute (\(R\_2\),\(T\_2\)) when (\(R\_1\),\(T\_1\)) is given. This is what the described function does. It computes (\(R\),\(T\)) such that: \(R\_2=R R\_1\) \(T\_2=R T\_1 + T.\) Therefore, one can compute the coordinate representation of a 3D point for the second camera's coordinate system when given the point's coordinate representation in the first camera's coordinate system: \(\begin{bmatrix} X\_2 \\ Y\_2 \\ Z\_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X\_1 \\ Y\_1 \\ Z\_1 \\ 1 \end{bmatrix}.\) Optionally, it computes the essential matrix E: \(E= \vecthreethree{0}{-T\_2}{T\_1}{T\_2}{0}{-T\_0}{-T\_1}{T\_0}{0} R\) where \(T\_i\) are components of the translation vector \(T\) : \(T=[T\_0, T\_1, T\_2]^T\) . And the function can also compute the fundamental matrix F: \(F = cameraMatrix2^{-T}\cdot E \cdot cameraMatrix1^{-1}\) Besides the stereo-related information, the function can also perform a full calibration of each of the two cameras. However, due to the high dimensionality of the parameter space and noise in the input data, the function can diverge from the correct solution. If the intrinsic parameters can be estimated with high accuracy for each of the cameras individually (for example, using calibrateCamera ), you are recommended to do so and then pass REF: CALIB\_FIX\_INTRINSIC flag to the function along with the computed intrinsic parameters. Otherwise, if all the parameters are estimated at once, it makes sense to restrict some parameters, for example, pass REF: CALIB\_SAME\_FOCAL\_LENGTH and REF: CALIB\_ZERO\_TANGENT\_DIST flags, which is usually a reasonable assumption. Similarly to calibrateCamera, the function minimizes the total re-projection error for all the points in all the available views from both cameras. The function returns the final value of the re-projection error. Returns:automatically generated

#### stereoCalibrateExtended public static double stereoCalibrateExtended(java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> objectPoints, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints1, java.util.List<[Mat](http://docs.google.com/org/opencv/core/Mat.html)> imagePoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) E, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Mat](http://docs.google.com/org/opencv/core/Mat.html) perViewErrors, int flags, [TermCriteria](http://docs.google.com/org/opencv/core/TermCriteria.html) criteria) Calibrates a stereo camera set up. This function finds the intrinsic parameters for each of the two cameras and the extrinsic parameters between the two cameras.Parameters:objectPoints - Vector of vectors of the calibration pattern points. The same structure as in REF: calibrateCamera. For each pattern view, both cameras need to see the same object points. Therefore, objectPoints.size(), imagePoints1.size(), and imagePoints2.size() need to be equal as well as objectPoints[i].size(), imagePoints1[i].size(), and imagePoints2[i].size() need to be equal for each i.imagePoints1 - Vector of vectors of the projections of the calibration pattern points, observed by the first camera. The same structure as in REF: calibrateCamera.imagePoints2 - Vector of vectors of the projections of the calibration pattern points, observed by the second camera. The same structure as in REF: calibrateCamera.cameraMatrix1 - Input/output camera intrinsic matrix for the first camera, the same as in REF: calibrateCamera. Furthermore, for the stereo case, additional flags may be used, see below.distCoeffs1 - Input/output vector of distortion coefficients, the same as in REF: calibrateCamera.cameraMatrix2 - Input/output second camera intrinsic matrix for the second camera. See description for cameraMatrix1.distCoeffs2 - Input/output lens distortion coefficients for the second camera. See description for distCoeffs1.imageSize - Size of the image used only to initialize the camera intrinsic matrices.R - Output rotation matrix. Together with the translation vector T, this matrix brings points given in the first camera's coordinate system to points in the second camera's coordinate system. In more technical terms, the tuple of R and T performs a change of basis from the first camera's coordinate system to the second camera's coordinate system. Due to its duality, this tuple is equivalent to the position of the first camera with respect to the second camera coordinate system.T - Output translation vector, see description above.E - Output essential matrix.F - Output fundamental matrix.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:

* + REF: CALIB\_FIX\_INTRINSIC Fix cameraMatrix? and distCoeffs? so that only R, T, E, and F matrices are estimated.
  + REF: CALIB\_USE\_INTRINSIC\_GUESS Optimize some or all of the intrinsic parameters according to the specified flags. Initial values are provided by the user.
  + REF: CALIB\_USE\_EXTRINSIC\_GUESS R and T contain valid initial values that are optimized further. Otherwise R and T are initialized to the median value of the pattern views (each dimension separately).
  + REF: CALIB\_FIX\_PRINCIPAL\_POINT Fix the principal points during the optimization.
  + REF: CALIB\_FIX\_FOCAL\_LENGTH Fix \(f^{(j)}\_x\) and \(f^{(j)}\_y\) .
  + REF: CALIB\_FIX\_ASPECT\_RATIO Optimize \(f^{(j)}\_y\) . Fix the ratio \(f^{(j)}\_x/f^{(j)}\_y\) .
  + REF: CALIB\_SAME\_FOCAL\_LENGTH Enforce \(f^{(0)}\_x=f^{(1)}\_x\) and \(f^{(0)}\_y=f^{(1)}\_y\) .
  + REF: CALIB\_ZERO\_TANGENT\_DIST Set tangential distortion coefficients for each camera to zeros and fix there.
  + REF: CALIB\_FIX\_K1,..., REF: CALIB\_FIX\_K6 Do not change the corresponding radial distortion coefficient during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_RATIONAL\_MODEL Enable coefficients k4, k5, and k6. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the rational model and return 8 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_THIN\_PRISM\_MODEL Coefficients s1, s2, s3 and s4 are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the thin prism model and return 12 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_S1\_S2\_S3\_S4 The thin prism distortion coefficients are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.
  + REF: CALIB\_TILTED\_MODEL Coefficients tauX and tauY are enabled. To provide the backward compatibility, this extra flag should be explicitly specified to make the calibration function use the tilted sensor model and return 14 coefficients. If the flag is not set, the function computes and returns only 5 distortion coefficients.
  + REF: CALIB\_FIX\_TAUX\_TAUY The coefficients of the tilted sensor model are not changed during the optimization. If REF: CALIB\_USE\_INTRINSIC\_GUESS is set, the coefficient from the supplied distCoeffs matrix is used. Otherwise, it is set to 0.

criteria - Termination criteria for the iterative optimization algorithm.The function estimates the transformation between two cameras making a stereo pair. If one computes the poses of an object relative to the first camera and to the second camera, ( \(R\_1\),\(T\_1\) ) and (\(R\_2\),\(T\_2\)), respectively, for a stereo camera where the relative position and orientation between the two cameras are fixed, then those poses definitely relate to each other. This means, if the relative position and orientation (\(R\),\(T\)) of the two cameras is known, it is possible to compute (\(R\_2\),\(T\_2\)) when (\(R\_1\),\(T\_1\)) is given. This is what the described function does. It computes (\(R\),\(T\)) such that: \(R\_2=R R\_1\) \(T\_2=R T\_1 + T.\) Therefore, one can compute the coordinate representation of a 3D point for the second camera's coordinate system when given the point's coordinate representation in the first camera's coordinate system: \(\begin{bmatrix} X\_2 \\ Y\_2 \\ Z\_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X\_1 \\ Y\_1 \\ Z\_1 \\ 1 \end{bmatrix}.\) Optionally, it computes the essential matrix E: \(E= \vecthreethree{0}{-T\_2}{T\_1}{T\_2}{0}{-T\_0}{-T\_1}{T\_0}{0} R\) where \(T\_i\) are components of the translation vector \(T\) : \(T=[T\_0, T\_1, T\_2]^T\) . And the function can also compute the fundamental matrix F: \(F = cameraMatrix2^{-T}\cdot E \cdot cameraMatrix1^{-1}\) Besides the stereo-related information, the function can also perform a full calibration of each of the two cameras. However, due to the high dimensionality of the parameter space and noise in the input data, the function can diverge from the correct solution. If the intrinsic parameters can be estimated with high accuracy for each of the cameras individually (for example, using calibrateCamera ), you are recommended to do so and then pass REF: CALIB\_FIX\_INTRINSIC flag to the function along with the computed intrinsic parameters. Otherwise, if all the parameters are estimated at once, it makes sense to restrict some parameters, for example, pass REF: CALIB\_SAME\_FOCAL\_LENGTH and REF: CALIB\_ZERO\_TANGENT\_DIST flags, which is usually a reasonable assumption. Similarly to calibrateCamera, the function minimizes the total re-projection error for all the points in all the available views from both cameras. The function returns the final value of the re-projection error. Returns:automatically generated

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D). the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area. scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases. initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion. are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D).flags - Operation flags that may be zero or REF: CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area. scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases. initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion. are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D).flags - Operation flags that may be zero or REF: CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.alpha - Free scaling parameter. If it is -1 or absent, the function performs the default scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases. initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion. are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D).flags - Operation flags that may be zero or REF: CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.alpha - Free scaling parameter. If it is -1 or absent, the function performs the default scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion. are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI1) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D).flags - Operation flags that may be zero or REF: CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.alpha - Free scaling parameter. If it is -1 or absent, the function performs the default scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion.validPixROI1 - Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectify public static void stereoRectify([Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cameraMatrix2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) distCoeffs2, [Size](http://docs.google.com/org/opencv/core/Size.html) imageSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R, [Mat](http://docs.google.com/org/opencv/core/Mat.html) T, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) R2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) P2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) Q, int flags, double alpha, [Size](http://docs.google.com/org/opencv/core/Size.html) newImageSize, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI1, [Rect](http://docs.google.com/org/opencv/core/Rect.html) validPixROI2) Computes rectification transforms for each head of a calibrated stereo camera.Parameters:cameraMatrix1 - First camera intrinsic matrix.distCoeffs1 - First camera distortion parameters.cameraMatrix2 - Second camera intrinsic matrix.distCoeffs2 - Second camera distortion parameters.imageSize - Size of the image used for stereo calibration.R - Rotation matrix from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.T - Translation vector from the coordinate system of the first camera to the second camera, see REF: stereoCalibrate.R1 - Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.R2 - Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.P1 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.P2 - Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.Q - Output \(4 \times 4\) disparity-to-depth mapping matrix (see REF: reprojectImageTo3D).flags - Operation flags that may be zero or REF: CALIB\_ZERO\_DISPARITY . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.alpha - Free scaling parameter. If it is -1 or absent, the function performs the default scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases.newImageSize - New image resolution after rectification. The same size should be passed to initUndistortRectifyMap (see the stereo\_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion.validPixROI1 - Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below).validPixROI2 - Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below). The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by stereoCalibrate as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

* + **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx\_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx\_2 & T\_x\*f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,\) where \(T\_x\) is a horizontal shift between the cameras and \(cx\_1=cx\_2\) if REF: CALIB\_ZERO\_DISPARITY is set.

* + **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

\(\texttt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\) \(\texttt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy\_2 & T\_y\*f \\ 0 & 0 & 1 & 0 \end{bmatrix},\) where \(T\_y\) is a vertical shift between the cameras and \(cy\_1=cy\_2\) if REF: CALIB\_ZERO\_DISPARITY is set. As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to initUndistortRectifyMap to initialize the rectification map for each camera. See below the screenshot from the stereo\_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels. ![image](pics/stereo\_undistort.jpg)

#### stereoRectifyUncalibrated public static boolean stereoRectifyUncalibrated([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Size](http://docs.google.com/org/opencv/core/Size.html) imgSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H2) Computes a rectification transform for an uncalibrated stereo camera.Parameters:points1 - Array of feature points in the first image.points2 - The corresponding points in the second image. The same formats as in findFundamentalMat are supported.F - Input fundamental matrix. It can be computed from the same set of point pairs using findFundamentalMat .imgSize - Size of the image.H1 - Output rectification homography matrix for the first image.H2 - Output rectification homography matrix for the second image. than zero, all the point pairs that do not comply with the epipolar geometry (that is, the points for which \(|\texttt{points2[i]}^T\*\texttt{F}\*\texttt{points1[i]}|>\texttt{threshold}\) ) are rejected prior to computing the homographies. Otherwise, all the points are considered inliers. The function computes the rectification transformations without knowing intrinsic parameters of the cameras and their relative position in the space, which explains the suffix "uncalibrated". Another related difference from stereoRectify is that the function outputs not the rectification transformations in the object (3D) space, but the planar perspective transformations encoded by the homography matrices H1 and H2 . The function implements the algorithm CITE: Hartley99 . **Note:** While the algorithm does not need to know the intrinsic parameters of the cameras, it heavily depends on the epipolar geometry. Therefore, if the camera lenses have a significant distortion, it would be better to correct it before computing the fundamental matrix and calling this function. For example, distortion coefficients can be estimated for each head of stereo camera separately by using calibrateCamera . Then, the images can be corrected using undistort , or just the point coordinates can be corrected with undistortPoints . Returns:automatically generated

#### stereoRectifyUncalibrated public static boolean stereoRectifyUncalibrated([Mat](http://docs.google.com/org/opencv/core/Mat.html) points1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) F, [Size](http://docs.google.com/org/opencv/core/Size.html) imgSize, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) H2, double threshold) Computes a rectification transform for an uncalibrated stereo camera.Parameters:points1 - Array of feature points in the first image.points2 - The corresponding points in the second image. The same formats as in findFundamentalMat are supported.F - Input fundamental matrix. It can be computed from the same set of point pairs using findFundamentalMat .imgSize - Size of the image.H1 - Output rectification homography matrix for the first image.H2 - Output rectification homography matrix for the second image.threshold - Optional threshold used to filter out the outliers. If the parameter is greater than zero, all the point pairs that do not comply with the epipolar geometry (that is, the points for which \(|\texttt{points2[i]}^T\*\texttt{F}\*\texttt{points1[i]}|>\texttt{threshold}\) ) are rejected prior to computing the homographies. Otherwise, all the points are considered inliers. The function computes the rectification transformations without knowing intrinsic parameters of the cameras and their relative position in the space, which explains the suffix "uncalibrated". Another related difference from stereoRectify is that the function outputs not the rectification transformations in the object (3D) space, but the planar perspective transformations encoded by the homography matrices H1 and H2 . The function implements the algorithm CITE: Hartley99 . **Note:** While the algorithm does not need to know the intrinsic parameters of the cameras, it heavily depends on the epipolar geometry. Therefore, if the camera lenses have a significant distortion, it would be better to correct it before computing the fundamental matrix and calling this function. For example, distortion coefficients can be estimated for each head of stereo camera separately by using calibrateCamera . Then, the images can be corrected using undistort , or just the point coordinates can be corrected with undistortPoints . Returns:automatically generated

#### triangulatePoints public static void triangulatePoints([Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatr1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projMatr2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projPoints1, [Mat](http://docs.google.com/org/opencv/core/Mat.html) projPoints2, [Mat](http://docs.google.com/org/opencv/core/Mat.html) points4D) This function reconstructs 3-dimensional points (in homogeneous coordinates) by using their observations with a stereo camera.Parameters:projMatr1 - 3x4 projection matrix of the first camera, i.e. this matrix projects 3D points given in the world's coordinate system into the first image.projMatr2 - 3x4 projection matrix of the second camera, i.e. this matrix projects 3D points given in the world's coordinate system into the second image.projPoints1 - 2xN array of feature points in the first image. In the case of the c++ version, it can be also a vector of feature points or two-channel matrix of size 1xN or Nx1.projPoints2 - 2xN array of corresponding points in the second image. In the case of the c++ version, it can be also a vector of feature points or two-channel matrix of size 1xN or Nx1.points4D - 4xN array of reconstructed points in homogeneous coordinates. These points are returned in the world's coordinate system. **Note:** Keep in mind that all input data should be of float type in order for this function to work. **Note:** If the projection matrices from REF: stereoRectify are used, then the returned points are represented in the first camera's rectified coordinate system. SEE: reprojectImageTo3D

#### validateDisparity public static void validateDisparity([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cost, int minDisparity, int numberOfDisparities)

#### validateDisparity public static void validateDisparity([Mat](http://docs.google.com/org/opencv/core/Mat.html) disparity, [Mat](http://docs.google.com/org/opencv/core/Mat.html) cost, int minDisparity, int numberOfDisparities, int disp12MaxDisp)

* [Overview](http://docs.google.com/overview-summary.html)
* [Package](http://docs.google.com/package-summary.html)
* Class
* [Tree](http://docs.google.com/package-tree.html)
* [Index](http://docs.google.com/index-all.html)
* [Help](http://docs.google.com/help-doc.html)
* Prev Class
* [Next Class](http://docs.google.com/org/opencv/calib3d/StereoBM.html)
* [Frames](http://docs.google.com/index.html?org/opencv/calib3d/Calib3d.html)
* [No Frames](http://docs.google.com/Calib3d.html)
* [All Classes](http://docs.google.com/allclasses-noframe.html)
* Summary:
* Nested |
* [Field](#3znysh7) |
* [Constr](#2et92p0) |
* [Method](#tyjcwt)
* Detail:
* [Field](#1t3h5sf) |
* [Constr](#pkwqa1) |
* [Method](#1opuj5n)

Generated on 2021-04-02 03:15:03 / OpenCV 3.4.14