HSP拡合令／関数	OpenCVSharp側関数	OpenCV側Namepase/関数／クラス／変数等	HTMLファイル名	説明(英語)	引数説明(英語)	詳細説明(英語)	引数の数	C++	特殊	変数getter	変数setter	new	delete	ptrget	memcpy
aruco_DetectorParameters_create	CVAPI(ExceptionStatus) aruco_DetectorParameters_create(aruco_DetectorParameters **returnValue) { BEGIN_WRAP const auto p = cv::aruco::DetectorParameters::create(); *returnValue = c(p); END_WRAP }	cv::aruco::DetectorParameters::create()	structcv_1_1aruco_1_1DetectorParameters.html				1								
aruco_detectMarkers	CVAPI(ExceptionStatus) aruco_detectMarkers(cv::_InputArray *image, cv::Ptr<cv::aruco::Dictionary> *dictionary, std::vector< std::vector<cv::Point2f> > *corners, std::vector<int> *ids, aruco_DetectorParameters *parameters, std::vector< std::vector<cv::Point2f> > *rejectedImgPoints) { BEGIN_WRAP const auto p = cpp(*parameters); cv::aruco::detectMarkers(*image, *dictionary, *corners, *ids, p, *rejectedImgPoints); END_WRAP }	cv::aruco::detectMarkers()	group__aruco.html	Basic marker detection.	image : input image dictionary : indicates the type of markers that will be searched corners : vector of detected marker corners. For each marker, its four corners are provided, (e.g std::vector<std::vector<cv::Point2f> >). For N detected markers, the dimensions of this array is Nx4. The order of the corners is clockwise. ids : vector of identifiers of the detected markers. The identifier is of type int (e.g. std::vector<int>). For N detected markers, the size of ids is also N. The identifiers have the same order than the markers in the imgPoints array. parameters : marker detection parameters rejectedImgPoints : contains the imgPoints of those squares whose inner code has not a correct codification. Useful for debugging purposes. cameraMatrix : optional input 3x3 floating-point camera matrix ¥(A = ¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) distCoeff : optional vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6],[s_1, s_2, s_3, s_4]]¥) of 4, 5, 8 or 12 elements	Performs marker detection in the input image. Only markers included in the specific dictionary are searched. For each detected marker, it returns the 2D position of its corner in the image and its corresponding identifier. Note that this function does not perform pose estimation.See alsoestimatePoseSingleMarkers, estimatePoseBoard	6								
aruco_drawDetectedMarkers	CVAPI(ExceptionStatus) aruco_drawDetectedMarkers(cv::_InputOutputArray *image, cv::Point2f **corners, int cornerSize1, int *cornerSize2, int *ids, int idxCount, MyCvScalar borderColor) { BEGIN_WRAP std::vector< std::vector<cv::Point2f> > cornerVec(cornerSize1); std::vector<int> idxVec; for (int i = 0; i < cornerSize1; i++) cornerVec[i] = std::vector<cv::Point2f>(corners[i], corners[i] + cornerSize2[i]); if (idx != nullptr) idxVec = std::vector<int>(idx, idx + idxCount);	cv::aruco::drawDetectedMarkers()	group__aruco.html	Draw detected markers in image.	image : input/output image. It must have 1 or 3 channels. The number of channels is not altered. corners : positions of marker corners on input image. (e.g std::vector<std::vector<cv::Point2f> >). For N detected markers, the dimensions of this array should be Nx4. The order of the corners should be clockwise. ids : vector of identifiers for markers in markersCorners . Optional, if not provided, ids are not painted. borderColor : color of marker borders. Rest of colors (text color and first corner color) are calculated based on this one to improve visualization.	Given an array of detected marker corners and its corresponding ids, this functions draws the markers in the image. The marker borders are painted and the markers identifiers if provided. Useful for debugging purposes.	4								
aruco_drawMarker	CVAPI(ExceptionStatus) aruco_drawMarker(cv::Ptr<cv::aruco::Dictionary> *dictionary, int id, int sidePixels, cv::_OutputArray *img, int borderBits) { BEGIN_WRAP cv::aruco::drawMarker(*dictionary, id, sidePixels, *img, borderBits); END_WRAP }	cv::aruco::drawMarker()	group__aruco.html	Draw a canonical marker image.	dictionary : dictionary of markers indicating the type of markers id : identifier of the marker that will be returned. It has to be a valid id in the specified dictionary. sidePixels : size of the image in pixels img : output image with the marker borderBits : width of the marker border.	This function returns a marker image in its canonical form (i.e. ready to be printed)	5								
aruco_estimatePoseSingleMarkers	CVAPI(ExceptionStatus) aruco_estimatePoseSingleMarkers(cv::Point2f **corners, int cornersLength1, int *cornersLengths2, float markerLength, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_OutputArray *rvecs, cv::_OutputArray *tvecs, cv::_OutputArray *objPoints) { BEGIN_WRAP std::vector<std::vector<cv::Point2f> > cornersVec(cornersLength1); for (int i = 0; i < cornersLength1; i++) cornersVec[i] = std::vector<cv::Point2f>(corners[i], corners[i] + cornersLengths2[i]); cv::aruco::estimatePoseSingleMarkers(cornersVec, markerLength, *cameraMatrix, *distCoeffs, *rvecs, *tvecs, *objPoints); END_WRAP }	cv::aruco::estimatePoseSingleMarkers()	group__aruco.html	Pose estimation for single markers.	corners : vector of already detected markers corners. For each marker, its four corners are provided, (e.g std::vector<std::vector<cv::Point2f> >). For N detected markers, the dimensions of this array should be Nx4. The order of the corners should be clockwise. markerLength : the length of the markers' side. The returning translation vectors will be in the same unit. Normally, unit is meters. cameraMatrix : input 3x3 floating-point camera matrix ¥(A = ¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) distCoeffs : vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6],[s_1, s_2, s_3, s_4]]¥) of 4, 5, 8 or 12 elements rvecs : array of output rotation vectors (tvecs : array of output translation vectors (e.g. std::vector<cv::Vec3d>). Each element in tvecs corresponds to the specific marker in imgPoints. _objPoints : array of object points of all the marker corners	See alsodetectMarkersSee alsoRodrigues) (e.g. std::vector<cv::Vec3d>). Each element in rvecs corresponds to the specific marker in imgPoints.This function receives the detected markers and returns their pose estimation respect to the camera individually. So for each marker, one rotation and translation vector is returned. The returned transformation is the one that transforms points from each marker coordinate system to the camera coordinate system. The marker corrdinate system is centered on the middle of the marker, with the Z axis perpendicular to the marker plane. The coordinates of the four corners of the marker in its own coordinate system are: (-markerLength/2, markerLength/2, 0), (markerLength/2, markerLength/2, 0), (markerLength/2, -markerLength/2, 0), (-markerLength/2, -markerLength/2, 0)	7								
aruco_drawAxis	CVAPI(ExceptionStatus) aruco_drawAxis(cv::_InputOutputArray *image, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *rvec, cv::_InputArray *tvec, float length) { BEGIN_WRAP cv::aruco::drawAxis(*image, *cameraMatrix, *distCoeffs, *rvec, *tvec, length); END_WRAP }	cv::aruco::drawAxis()	group__aruco.html	Draw coordinate system axis from pose estimation.	image : input/output image. It must have 1 or 3 channels. The number of channels is not altered. cameraMatrix : input 3x3 floating-point camera matrix ¥(A = ¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) distCoeffs : vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6],[s_1, s_2, s_3, s_4]]¥) of 4, 5, 8 or 12 elements rvec : rotation vector of the coordinate system that will be drawn. (tvec : translation vector of the coordinate system that will be drawn. length : length of the painted axis in the same unit than tvec (usually in meters)	See alsoRodrigues).Given the pose estimation of a marker or board, this function draws the axis of the world coordinate system, i.e. the system centered on the marker/board. Useful for debugging purposes.Deprecated:use cv::drawFrameAxesExamples: samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp.	6								
aruco_getPredefinedDictionary	CVAPI(ExceptionStatus) aruco_getPredefinedDictionary(int name, cv::Ptr<cv::aruco::Dictionary>** returnValue) { BEGIN_WRAP const auto dictionary = cv::aruco::getPredefinedDictionary(static_cast<cv::aruco::PREDEFINED_DICTIONARY_NAME>(name)); *returnValue = new cv::Ptr<cv::aruco::Dictionary>(dictionary); END_WRAP }	cv::aruco::getPredefinedDictionary()	group__aruco.html	Returns one of the predefined dictionaries defined in PREDEFINED_DICTIONARY_NAME.			1								
aruco_detectCharucoDiamond	CVAPI(ExceptionStatus) aruco_detectCharucoDiamond(cv::_InputArray *image, cv::Point2f **markerCorners, int markerCornersSize1, int *markerCornersSize2, std::vector<int> *markerIds, float squareMarkerLengthRate, std::vector< std::vector<cv::Point2f> > *diamondCorners, std::vector<cv::Vec4i> *diamondIds, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs) { BEGIN_WRAP std::vector< std::vector<cv::Point2f> > markerCornersVec(markerCornersSize1); for (int i = 0; i <	cv::aruco::detectCharucoDiamond()	group__aruco.html	Detect ChArUco Diamond markers.	image : input image necessary for corner subpixel. markerCorners : list of detected marker corners from detectMarkers function. markerIds : list of marker ids in markerCorners. squareMarkerLengthRate : rate between square and marker length: squareMarkerLengthRate = squareLength/markerLength. The real units are not necessary. diamondCorners : output list of detected diamond corners (4 corners per diamond). The order is the same than in marker corners: top left, top right, bottom right and bottom left. Similar format than the corners returned by detectMarkers (e.g std::vector<std::vector<cv::Point2f> >). diamondIds : ids of the diamonds in diamondCorners. The id of each diamond is in fact of type Vec4, so each diamond has 4 ids, which are the ids of the aruco markers composing the diamond. cameraMatrix : Optional camera calibration matrix. distCoeffs : Optional camera distortion coefficients.	This function detects Diamond markers from the previous detected ArUco markers. The diamonds are returned in the diamondCorners and diamondIds parameters. If camera calibration parameters are provided, the diamond search is based on reprojection. If not, diamond search is based on homography. Homography is faster than reprojection but can slightly reduce the detection rate.	8								
aruco_drawDetectedDiamonds	CVAPI(ExceptionStatus) aruco_drawDetectedDiamond s(cv::_InputOutputArray *image, cv::Point2f **corners, int cornerSize1, int *cornerSize2, std::vector<cv::Vec4i> *ids, MyCvScalar borderColor) { BEGIN_WRAP std::vector< std::vector<cv::Point2f> > cornerVec(cornerSize1); for (int i = 0; i < cornerSize1; i++) cornerVec[i] = std::vector<cv::Point2f>(corners[i], corners[i] + cornerSize2[i]); const cv::_InputArray idsArray = (ids != nullptr) ? *ids :	cv::aruco::drawDetectedDiamonds()	group__aruco.html	Draw a set of detected ChArUco Diamond markers.	image : input/output image. It must have 1 or 3 channels. The number of channels is not altered. diamondCorners : positions of diamond corners in the same format returned by detectChanucoDiamond(). (e.g std::vector<std::vector<cv::Point2f> >). For N detected markers, the dimensions of this array should be Nx4. The order of the corners should be clockwise. diamondIds : vector of identifiers for diamonds in diamondCorners, in the same format returned by detectChanucoDiamond() (e.g. std::vector<Vec4i>). Optional, if not provided, ids are not painted. borderColor : color of marker borders. Rest of colors (text color and first corner color) are calculated based on this one.	Given an array of detected diamonds, this functions draws them in the image. The marker borders are painted and the markers identifiers if provided. Useful for debugging purposes.	4								

aruco_Ptr_Dictionary_delete	CVAPI(ExceptionStatus) aruco_Ptr_Dictionary_delete(cv::Ptr<cv::aruco::Dictionary> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::aruco::Dictionary	classcv_1_1aruco_1_1Dictionary.html	Dictionary/Set of markers. It contains the inner codification. bytesList contains the marker codewords where bytesList.rows is the dictionary size each marker is encoded using nbytes = ceil(markerSize*markerSize/8.) each row contains all 4 rotations of the marker, so its length is 4*nbytes bytesList.ptr(i)[k*nbytes + j] is then the j-th byte of i-th marker, in its k-th rotation.										•		
aruco_Ptr_Dictionary_get	CVAPI(ExceptionStatus) aruco_Ptr_Dictionary_get(cv::Ptr<cv::aruco::Dictionary> *ptr, cv::aruco::Dictionary **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::aruco::Dictionary	classcv_1_1aruco_1_1Dictionary.html	Dictionary/Set of markers. It contains the inner codification. bytesList contains the marker codewords where bytesList.rows is the dictionary size each marker is encoded using nbytes = ceil(markerSize*markerSize/8.) each row contains all 4 rotations of the marker, so its length is 4*nbytes bytesList.ptr(i)[k*nbytes + j] is then the j-th byte of i-th marker, in its k-th rotation.											•	
aruco_Dictionary_setMarkerSize	CVAPI(ExceptionStatus) aruco_Dictionary_setMarkerSize(cv::aruco::Dictionary *obj, int value) { BEGIN_WRAP obj->markerSize = value; END_WRAP }	cv::aruco::Dictionary::markerSize	classcv_1_1aruco_1_1Dictionary.html									•					
aruco_Dictionary_setMaxCorrectionBits	CVAPI(ExceptionStatus) aruco_Dictionary_setMaxCorrectionBits(cv::aruco::Dictionary *obj, int value) { BEGIN_WRAP obj->maxCorrectionBits = value; END_WRAP }	cv::aruco::Dictionary::maxCorrectionBits	classcv_1_1aruco_1_1Dictionary.html									•					
aruco_Dictionary_getBytesList	CVAPI(ExceptionStatus) aruco_Dictionary_getBytesList(cv::aruco::Dictionary *obj, cv::Mat** returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->bytesList); END_WRAP }	cv::aruco::Dictionary::bytesList	classcv_1_1aruco_1_1Dictionary.html								•						
aruco_Dictionary_getMarkerSize	CVAPI(ExceptionStatus) aruco_Dictionary_getMarkerSize(cv::aruco::Dictionary *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->markerSize; END_WRAP }	cv::aruco::Dictionary::markerSize	classcv_1_1aruco_1_1Dictionary.html								•						
aruco_Dictionary_getMaxCorrectionBits	CVAPI(ExceptionStatus) aruco_Dictionary_getMaxCorrectionBits(cv::aruco::Dictionary *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->maxCorrectionBits; END_WRAP }	cv::aruco::Dictionary::maxCorrectionBits	classcv_1_1aruco_1_1Dictionary.html								•						
bgsegm_createBackgroundSubtractorMOG	CVAPI(ExceptionStatus) bgsegm_createBackgroundSubtractorMOG(int history, int nmixtures, double backgroundRatio, double noiseSigma, cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> **returnValue) { BEGIN_WRAP const auto ptr = cv::bgsegm::createBackgroundSubtractorMOG(history, nmixtures, backgroundRatio, noiseSigma); *returnValue = new cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG>(ptr); END_WRAP }	cv::bgsegm::createBackgroundSubtractorMOG()	group__bgsegm.html	Creates mixture-of-gaussian background subtractor.	history : Length of the history. nmixtures : Number of Gaussian mixtures. backgroundRatio : Background ratio. noiseSigma : Noise strength (standard deviation of the brightness or each color channel). 0 means some automatic value.			4									
bgsegm_Ptr_BackgroundSubtractorMOG_delete	CVAPI(ExceptionStatus) bgsegm_Ptr_BackgroundSubtractorMOG_delete(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html	Gaussian Mixture-based Background/Foreground Segmentation Algorithm. The class implements the algorithm described in [122] .										•		
bgsegm_Ptr_BackgroundSubtractorMOG_get	CVAPI(ExceptionStatus) bgsegm_Ptr_BackgroundSubtractorMOG_get(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, cv::bgsegm::BackgroundSubtractorMOG **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html	Gaussian Mixture-based Background/Foreground Segmentation Algorithm. The class implements the algorithm described in [122] .											•	
bgsegm_BackgroundSubtractorMOG_getHistory	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_getHistory(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getHistory(); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::getHistory()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_setHistory	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_setHistory(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, int value) { BEGIN_WRAP (*ptr)->setHistory(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::setHistory()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_getNMixtu	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_getNMixtures(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getNMixtures(); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::getNMixtures()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_setNMixtu	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_setNMixtures(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, int value) { BEGIN_WRAP (*ptr)->setNMixtures(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::setNMixtures()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_getBackgr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_getBackgroundRatio(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getBackgroundRatio(); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::getBackgroundRatio()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_setBackgr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_setBackgroundRatio(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, double value) { BEGIN_WRAP (*ptr)->setBackgroundRatio(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::setBackgroundRatio()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									
bgsegm_BackgroundSubtractorMOG_getNoiseS	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_getNoiseSigma(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getNoiseSigma(); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::getNoiseSigma()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html					1									

bgsegm_BackgroundSubtractorMOG_setNoiseS	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorMOG_setNoiseSigma(cv::Ptr<cv::bgsegm::BackgroundSubtractorMOG> *ptr, double value) { BEGIN_WRAP (*ptr)->setNoiseSigma(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorMOG::setNoiseSigma()	classcv_1_1bgsegm_1_1BackgroundSubtractorMOG.html				1								
bgsegm_createBackgroundSubtractorGMG	CVAPI(ExceptionStatus) bgsegm_createBackgroundSubtractorGMG(int initializationFrames, double decisionThreshold, cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> **returnValue) { BEGIN_WRAP const auto ptr = cv::bgsegm::createBackgroundSubtractorGMG(initializationFrames, decisionThreshold); *returnValue = new cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG>(ptr); END_WRAP }	cv::bgsegm::createBackgroundSubtractorGMG()	group__bgsegm.html	Creates a GMG Background Subtractor.	initializationFrames : number of frames used to initialize the background models. decisionThreshold : Threshold value, above which it is marked foreground, else background.		2								
bgsegm_Ptr_BackgroundSubtractorGMG_delete	CVAPI(ExceptionStatus) bgsegm_Ptr_BackgroundSubtractorGMG_delete(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Background Subtractor module based on the algorithm given in [92] . Takes a series of images and returns a sequence of mask (8UC1) images of the same size, where 255 indicates Foreground and 0 represents Background. This class implements an algorithm described in "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation," A. Godbehere, A. Matsukawa, K. Goldberg, American Control Conference, Montreal, June 2012.							•		
bgsegm_Ptr_BackgroundSubtractorGMG_get	CVAPI(ExceptionStatus) bgsegm_Ptr_BackgroundSubtractorGMG_get(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, cv::bgsegm::BackgroundSubtractorGMG **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Background Subtractor module based on the algorithm given in [92] . Takes a series of images and returns a sequence of mask (8UC1) images of the same size, where 255 indicates Foreground and 0 represents Background. This class implements an algorithm described in "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation," A. Godbehere, A. Matsukawa, K. Goldberg, American Control Conference, Montreal, June 2012.								•	
bgsegm_BackgroundSubtractorGMG_getMaxFe	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getMaxFeatures(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getMaxFeatures(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getMaxFeatures()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns total number of distinct colors to maintain in histogram.			1								
bgsegm_BackgroundSubtractorGMG_setMaxFe	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_setMaxFeatures(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int value) { BEGIN_WRAP (*ptr)->setMaxFeatures(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setMaxFeatures()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Sets total number of distinct colors to maintain in histogram.			1								
bgsegm_BackgroundSubtractorGMG_getDefaul	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getDefaultLearningRate(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getDefaultLearningRate(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getDefaultLearningRate()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns the learning rate of the algorithm.		It lies between 0.0 and 1.0. It determines how quickly features are "forgotten" from histograms.	1								
bgsegm_BackgroundSubtractorGMG_setDefaul	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_setDefaultLearningRate(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, double value) { BEGIN_WRAP (*ptr)->setDefaultLearningRate(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setDefaultLearningRate()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Sets the learning rate of the algorithm.			1								
bgsegm_BackgroundSubtractorGMG_getNumFr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getNumFrames(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getNumFrames(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getNumFrames()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns the number of frames used to initialize background model.			1								
bgsegm_BackgroundSubtractorGMG_setNumFr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_setNumFrames(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int value) { BEGIN_WRAP (*ptr)->setNumFrames(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setNumFrames()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Sets the number of frames used to initialize background model.			1								
bgsegm_BackgroundSubtractorGMG_getQuanti	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getQuantizationLevels(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getQuantizationLevels(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getQuantizationLevels()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns the parameter used for quantization of color-space.		It is the number of discrete levels in each channel to be used in histograms.	1								
bgsegm_BackgroundSubtractorGMG_setQuanti	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_setQuantizationLevels(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int value) { BEGIN_WRAP (*ptr)->setQuantizationLevels(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setQuantizationLevels()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Sets the parameter used for quantization of color-space.			1								
bgsegm_BackgroundSubtractorGMG_getBackgr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getBackgroundPrior(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getBackgroundPrior(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getBackgroundPrior()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns the prior probability that each individual pixel is a background pixel.			1								
bgsegm_BackgroundSubtractorGMG_setBackgr	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_setBackgroundPrior(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, double value) { BEGIN_WRAP (*ptr)->setBackgroundPrior(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setBackgroundPrior()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Sets the prior probability that each individual pixel is a background pixel.			1								
bgsegm_BackgroundSubtractorGMG_getSmooth	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtractorGMG_getSmoothingRadius(cv::Ptr<cv::bgsegm::BackgroundSubtractorGMG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getSmoothingRadius(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getSmoothingRadius()	classcv_1_1bgsegm_1_1BackgroundSubtractorGMG.html	Returns the kernel radius used for morphological operations.			1								

bgsegm_BackgroundSubtractorGMG_setSmoot	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_setSmoothingRadius(cv ::Ptr<cv::bgsegm::Backgroun dSubtractorGMG> *ptr, int value) { BEGIN_WRAP (*ptr)- >setSmoothingRadius(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setSmoot hingRadius()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Sets the kernel radius used for morphological operations.			1									
bgsegm_BackgroundSubtractorGMG_getDecisi	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_getDecisionThreshold(c v::Ptr<cv::bgsegm::Backgrou ndSubtractorGMG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)- >getDecisionThreshold(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getDecisi onThreshold()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Returns the value of decision threshold.		Decision value is the value above which pixel is determined to be FG.	1									
bgsegm_BackgroundSubtractorGMG_setDecisi	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_setDecisionThreshold(c v::Ptr<cv::bgsegm::Backgrou ndSubtractorGMG> *ptr, double value) { BEGIN_WRAP (*ptr)- >setDecisionThreshold(value) ; END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setDecisi onThreshold()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Sets the value of decision threshold.			1									
bgsegm_BackgroundSubtractorGMG_getUpdate	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_getUpdateBackground Model(cv::Ptr<cv::bgsegm::B ackgroundSubtractorGMG> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)- >getUpdateBackgroundModel( ) ? 1 : 0; END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getUpdat eBackgroundModel()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Returns the status of background model update.			1									
bgsegm_BackgroundSubtractorGMG_setUpdate	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_setUpdateBackgroundM odel(cv::Ptr<cv::bgsegm::Ba ckgroundSubtractorGMG> *ptr, int value) { BEGIN_WRAP (*ptr)- >setUpdateBackgroundModel(value != 0); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setUpdat eBackgroundModel()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Sets the status of background model update.			1									
bgsegm_BackgroundSubtractorGMG_getMinVal	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_getMinVal(cv::Ptr<cv:: bgsegm::BackgroundSubtract orGMG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)- >getMinVal(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getMinVa l()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Returns the minimum value taken on by pixels in image sequence. Usually 0.			1									
bgsegm_BackgroundSubtractorGMG_setMinVal	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_setMinVal(cv::Ptr<cv:: bgsegm::BackgroundSubtract orGMG> *ptr, double value) { BEGIN_WRAP (*ptr)->setMinVal(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setMinVal ()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Sets the minimum value taken on by pixels in image sequence.			1									
bgsegm_BackgroundSubtractorGMG_getMaxVa	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_getMaxVal(cv::Ptr<cv:: bgsegm::BackgroundSubtrac torGMG> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)- >getMaxVal(); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::getMaxV al()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Returns the maximum value taken on by pixels in image sequence. e.g. 1.0 or 255.			1									
bgsegm_BackgroundSubtractorGMG_setMaxVa	CVAPI(ExceptionStatus) bgsegm_BackgroundSubtracto rGMG_setMaxVal(cv::Ptr<cv:: bgsegm::BackgroundSubtract orGMG> *ptr, double value) { BEGIN_WRAP (*ptr)->setMaxVal(value); END_WRAP }	cv::bgsegm::BackgroundSubtractorGMG::setMaxV al()	classcv_1_1bgsegm_1_1Backgroun dSubtractorGMG.html	Sets the maximum value taken on by pixels in image sequence.			1									
dnn_superres_DnnSuperResImpl_new1	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_new1(cv::dnn_superres::DnnSuperR esImpl** returnValue) { BEGIN_WRAP *returnValue = new cv::dnn_superres::DnnSuperR esImpl; END_WRAP }	cv::dnn_superres::DnnSuperResImpl	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	A class to upscale images via convolutional neural networks. The following four models are implemented: edsr espcn fsrcnn lapsrn							•					
dnn_superres_DnnSuperResImpl_new2	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_new2(const char* algo, int scale, cv::dnn_superres::DnnSuperR esImpl** returnValue) { BEGIN_WRAP *returnValue = new cv::dnn_superres::DnnSuperR esImpl(algo, scale); END_WRAP }	cv::dnn_superres::DnnSuperResImpl	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	A class to upscale images via convolutional neural networks. The following four models are implemented: edsr espcn fsrcnn lapsrn							•					
dnn_superres_DnnSuperResImpl_delete	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_delete(cv::dnn_superres: :DnnSuperResImpl* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::dnn_superres::DnnSuperResImpl	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	A class to upscale images via convolutional neural networks. The following four models are implemented: edsr espcn fsrcnn lapsrn								•				
dnn_superres_DnnSuperResImpl_readModel1	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_readModel1(cv::dnn_superres::DnnSuperR esImpl* obj, const char *path) { BEGIN_WRAP obj->readModel(path); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::readModel()	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	Read the model from the given path.	path : Path to the model file.		1									
dnn_superres_DnnSuperResImpl_readModel2	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_readModel2(cv::dnn_superres::DnnSuperR esImpl* obj, const char* weights, const char *definition) { BEGIN_WRAP obj->readModel(weights, definition); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::readModel()	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	Read the model from the given path.	weights : Path to the model weights file. definition : Path to the model definition file.		2									
dnn_superres_DnnSuperResImpl_setModel	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_setModel(cv::dnn_superres::DnnSuperR esImpl* obj, const char* algo, int scale) { BEGIN_WRAP obj->setModel(algo, scale); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::setModel()	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	Set desired model.	algo : String containing one of the desired models: edsr espcn fsrcnn lapsrn scale : Integer specifying the upscale factor		2									
dnn_superres_DnnSuperResImpl_setPreferable	CVAPI(ExceptionStatus) dnn_superres_DnnSuperRes1 mpl_setPreferableBackend(cv::dnn_superres::DnnSuperR esImpl* obj, int backendId) { BEGIN_WRAP obj- >setPreferableBackend(backe ndId); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::setPreferabl eBackend()	classcv_1_1dnn__superres_1_1Dnn SuperResImpl.html	Set computation backend.			1									

dnn_superres_DnnSuperResImpl_setPreferable	CVAPI(ExceptionStatus) dnn_superres_DnnSuperResImpl_setPreferableTarget(cv::dnn_superres::DnnSuperResImpl* obj, int targetId) { BEGIN_WRAP obj->setPreferableTarget(targetId); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::setPreferableTarget()	classcv_1_1dnn__superres_1_1DnnSuperResImpl.html	Set computation target.			1									
dnn_superres_DnnSuperResImpl_upsample	CVAPI(ExceptionStatus) dnn_superres_DnnSuperResImpl_upsample(cv::dnn_superres::DnnSuperResImpl* obj, cv::_InputArray *img, cv::_OutputArray *result) { BEGIN_WRAP obj->upsample(*img, *result); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::upsample()	classcv_1_1dnn__superres_1_1DnnSuperResImpl.html	Upsample via neural network.	img : Image to upscale result : Destination upscaled image		2									
dnn_superres_DnnSuperResImpl_upsampleMul	CVAPI(ExceptionStatus) dnn_superres_DnnSuperResImpl_upsampleMultioutput(cv::dnn_superres::DnnSuperResImpl* obj, cv::_InputArray *img, std::vector<cv::Mat> *imgs_new, const int* scale_factors, int scale_factors_size, const char **node_names, int node_names_size) { BEGIN_WRAP const std::vector<int> scale_factors_vec(scale_factors, scale_factors + scale_factors_size); std::vector<cv::String> node_names_vec(node_names_size); for (int i = 0; i < node_names_size; i++) { node_names_vec[i].assign(cv	cv::dnn_superres::DnnSuperResImpl::upsampleMultioutput()	classcv_1_1dnn__superres_1_1DnnSuperResImpl.html	Upsample via neural network of multiple outputs.	img : Image to upscale imgs_new : Destination upscaled images scale_factors : Scaling factors of the output nodes node_names : Names of the output nodes in the neural network		4								
dnn_superres_DnnSuperResImpl_getScale	CVAPI(ExceptionStatus) dnn_superres_DnnSuperResImpl_getScale(cv::dnn_superres::DnnSuperResImpl* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->getScale(); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::getScale()	classcv_1_1dnn__superres_1_1DnnSuperResImpl.html	Returns the scale factor of the model:			1									
dnn_superres_DnnSuperResImpl_getAlgorithm	CVAPI(ExceptionStatus) dnn_superres_DnnSuperResImpl_getAlgorithm(cv::dnn_superres::DnnSuperResImpl* obj, std::string* returnValue) { BEGIN_WRAP returnValue->assign(obj->getAlgorithm()); END_WRAP }	cv::dnn_superres::DnnSuperResImpl::getAlgorithm()	classcv_1_1dnn__superres_1_1DnnSuperResImpl.html	Returns the scale factor of the model:			1									
flann_Index_new	CVAPI(ExceptionStatus) flann_Index_new(cv::_InputArray *features, cv::flann::IndexParams* params, cvflann::flann_distance_t distType, cv::flann::Index **returnValue) { BEGIN_WRAP *returnValue = new cv::flann::Index(*features, *params, distType); END_WRAP }	cv::flann::GenericIndex		The FLANN nearest neighbor index class. This class is templated with the type of elements for which the index is built.								•				
flann_Index_delete	CVAPI(ExceptionStatus) flann_Index_delete(cv::flann::Index* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::GenericIndex											•			
flann_Index_knnSearch1	CVAPI(ExceptionStatus) flann_Index_knnSearch1(cv::flann::Index* obj, float* queries, int queries_length, int* indices, float* dists, int knn, cv::flann::SearchParams* params) { BEGIN_WRAP const std::vector<float> queries_vec(queries, queries + queries_length); std::vector<int> indices_vec(knn); std::vector<float> dists_vec(knn); obj->knnSearch(queries_vec, indices_vec, dists_vec, knn, *params); memcpy(indices, &indices_vec[0], sizeof(int) * knn); memcpy(dists, &dists_vec[0], sizeof(float) * knn);	cv::flann::GenericIndex::knnSearch()														
flann_Index_knnSearch2	CVAPI(ExceptionStatus) flann_Index_knnSearch2(cv::flann::Index* obj, cv::Mat* queries, cv::Mat* indices, cv::Mat* dists, int knn, cv::flann::SearchParams* params) { BEGIN_WRAP obj->knnSearch(*queries, *indices, *dists, knn, *params); END_WRAP }	cv::flann::GenericIndex::knnSearch()														
flann_Index_knnSearch3	CVAPI(ExceptionStatus) flann_Index_knnSearch3(cv::flann::Index* obj, cv::Mat* queries, int* indices, float* dists, int knn, cv::flann::SearchParams* params) { BEGIN_WRAP cv::Mat indices_mat(1, knn, CV_32SC1); cv::Mat dists_mat(1, knn, CV_32FC1); obj->knnSearch(*queries, indices_mat, dists_mat, knn, *params); memcpy(indices, indices_mat.ptr<int>(0), sizeof(int) * knn); memcpy(dists, dists_mat.ptr<float>(0), sizeof(float) * knn); END_WRAP }	cv::flann::GenericIndex::knnSearch()														
flann_Index_radiusSearch1	CVAPI(ExceptionStatus) flann_Index_radiusSearch1(cv::flann::Index* obj, float* queries, int queries_length, int* indices, int indices_length, float* dists, int dists_length, double radius, int maxResults, cv::flann::SearchParams* params) { BEGIN_WRAP const std::vector<float> queries_vec(queries, queries + queries_length); std::vector<int> indices_vec(indices_length); std::vector<float> dists_vec(dists_length); obj->radiusSearch(queries_vec, indices_vec, dists_vec, radius, maxResults, *params); memcpy(indices, &indices_vec[0], sizeof(int) * indices_length);	cv::flann::GenericIndex::radiusSearch()														

flann_Index_radiusSearch2	CVAPI(ExceptionStatus) flann_Index_radiusSearch2(cv::flann::Index* obj, cv::Mat* queries, cv::Mat* indices, cv::Mat* dists, double radius, int maxResults, cv::flann::SearchParams* params) { BEGIN_WRAP obj->radiusSearch(*queries, *indices, *dists, radius, maxResults, *params); END_WRAP }	cv::flann::GenericIndex::radiusSearch()															
flann_Index_radiusSearch3	CVAPI(ExceptionStatus) flann_Index_radiusSearch3(cv::flann::Index* obj, cv::Mat* queries, int* indices, int indices_length, float* dists, int dists_length, double radius, int maxResults, cv::flann::SearchParams* params) { BEGIN_WRAP cv::Mat indices_mat(1, indices_length, CV_32SC1); cv::Mat dists_mat(1, dists_length, CV_32FC1); obj->radiusSearch(*queries, indices_mat, dists_mat, radius, maxResults, *params); memcpy(indices, indices_mat.ptr<int>(0), sizeof(int) * indices_length); memcpy(dists, dists_mat.ptr<float>(0), sizeof(float) * dists_length); END_WRAP }	cv::flann::GenericIndex::radiusSearch()															
flann_Index_save	CVAPI(ExceptionStatus) flann_Index_save(cv::flann::Index* obj, const char* filename) { BEGIN_WRAP obj->save(filename); END_WRAP }	cv::flann::GenericIndex::save()															
flann_Ptr_IndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_IndexParams_new(cv::Ptr<cv::flann::IndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::IndexParams>(new cv::flann::IndexParams); END_WRAP }	cv::flann::IndexParams												•			
flann_Ptr_IndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_IndexParams_get(cv::Ptr<cv::flann::IndexParams> *ptr, cv::flann::IndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::IndexParams														•	
flann_Ptr_IndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_IndexParams_delete(cv::Ptr<cv::flann::IndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::IndexParams													•		
flann_IndexParams_getString	CVAPI(ExceptionStatus) flann_IndexParams_getString(cv::flann::IndexParams *obj, const char *key, const char *defaultVal, std::string *returnValue) { BEGIN_WRAP returnValue->assign(obj->getString(key, defaultVal)); END_WRAP }	cv::flann::IndexParams::getString()															
flann_IndexParams_getInt	CVAPI(ExceptionStatus) flann_IndexParams_getInt(cv::flann::IndexParams* obj, const char* key, int defaultVal, int *returnValue) { BEGIN_WRAP *returnValue = obj->getInt(key, defaultVal); END_WRAP }	cv::flann::IndexParams::getInt()															
flann_IndexParams_getDouble	CVAPI(ExceptionStatus) flann_IndexParams_getDouble(cv::flann::IndexParams* obj, const char* key, double defaultVal, double *returnValue) { BEGIN_WRAP *returnValue = obj->getDouble(key, defaultVal); END_WRAP }	cv::flann::IndexParams::getDouble()															
flann_IndexParams_setString	CVAPI(ExceptionStatus) flann_IndexParams_setString(cv::flann::IndexParams* obj, const char* key, const char* value) { BEGIN_WRAP obj->setString(key, value); END_WRAP }	cv::flann::IndexParams::setString()															
flann_IndexParams_setInt	CVAPI(ExceptionStatus) flann_IndexParams_setInt(cv::flann::IndexParams* obj, const char* key, int value) { BEGIN_WRAP obj->setInt(key, value); END_WRAP }	cv::flann::IndexParams::setInt()															
flann_IndexParams_setDouble	CVAPI(ExceptionStatus) flann_IndexParams_setDouble(cv::flann::IndexParams* obj, const char* key, double value) { BEGIN_WRAP obj->setDouble(key, value); END_WRAP }	cv::flann::IndexParams::setDouble()															
flann_IndexParams_setFloat	CVAPI(ExceptionStatus) flann_IndexParams_setFloat(cv::flann::IndexParams* obj, const char* key, float value) { BEGIN_WRAP obj->setFloat(key, value); END_WRAP }	cv::flann::IndexParams::setFloat()															
flann_IndexParams_setBool	CVAPI(ExceptionStatus) flann_IndexParams_setBool(cv::flann::IndexParams* obj, const char* key, int value) { BEGIN_WRAP obj->setBool(key, (value != 0)); END_WRAP }	cv::flann::IndexParams::setBool()															
flann_IndexParams_setAlgorithm	CVAPI(ExceptionStatus) flann_IndexParams_setAlgorithm(cv::flann::IndexParams* obj, int value) { BEGIN_WRAP obj->setAlgorithm(value); END_WRAP }	cv::flann::IndexParams::setAlgorithm()															
flann_Ptr_LinearIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_LinearIndexParams_new(cv::Ptr<cv::flann::LinearIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::LinearIndexParams>(new cv::flann::LinearIndexParams); END_WRAP }	cv::flann::LinearIndexParams		the index will perform a linear, brute-force search.										•			

flann_Ptr_LinearIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_LinearIndexParams_get(cv::Ptr<cv::flann::LinearIndexParams> *ptr, cv::flann::LinearIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::LinearIndexParams												•		
flann_Ptr_LinearIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_LinearIndexParams_delete(cv::Ptr<cv::flann::LinearIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::LinearIndexParams													•	
flann_Ptr_KDTreeIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_KDTreeIndexParams_new(int trees, cv::Ptr<cv::flann::KDTreeIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::KDTreeIndexParams>(new cv::flann::KDTreeIndexParams(trees)); END_WRAP }	cv::flann::KDTreeIndexParams		the index constructed will consist of a set of randomized kd-trees which will be searched in parallel.								•			
flann_Ptr_KDTreeIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_KDTreeIndexParams_get(cv::Ptr<cv::flann::KDTreeIndexParams> *ptr, cv::flann::KDTreeIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::KDTreeIndexParams												•		
flann_Ptr_KDTreeIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_KDTreeIndexParams_delete(cv::Ptr<cv::flann::KDTreeIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::KDTreeIndexParams													•	
flann_Ptr_KMeansIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_KMeansIndexParams_new(int branching, int iterations, cvflann::flann_centers_init_t centers_init, float cb_index, cv::Ptr<cv::flann::KMeansIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::KMeansIndexParams>(new cv::flann::KMeansIndexParams(branching, iterations, centers_init, cb_index)); END_WRAP }	cv::flann::KMeansIndexParams		the index constructed will be a hierarchical k-means tree (one tree by default), dividing each set of points into n clusters whose barycenters are refined iteratively. Note that this algorithm has been extended to the support of binary vectors as an alternative to LSH when knn search speed is the criterium. It will also outperform LSH when processing directly (i.e. without the use of MCA/PCA) datasets whose points share mostly the same values for most of the dimensions. It is recommended to set more than one tree with binary data.								•			
flann_Ptr_KMeansIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_KMeansIndexParams_get(cv::Ptr<cv::flann::KMeansIndexParams> *ptr, cv::flann::KMeansIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::KMeansIndexParams												•		
flann_Ptr_KMeansIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_KMeansIndexParams_delete(cv::Ptr<cv::flann::KMeansIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::KMeansIndexParams													•	
flann_Ptr_LshIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_LshIndexParams_new(int table_number, int key_size, int multi_probe_level, cv::Ptr<cv::flann::LshIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::LshIndexParams>(new cv::flann::LshIndexParams(table_number, key_size, multi_probe_level)); END_WRAP }	cv::flann::LshIndexParams		the index created uses multi-probe LSH (by Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search by Qin Lv, William Josephson, Zhe Wang, Moses Charikar, Kai Li., Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB). Vienna, Austria. September 2007). This algorithm is designed for binary vectors.								•			
flann_Ptr_LshIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_LshIndexParams_get(cv::Ptr<cv::flann::LshIndexParams> *ptr, cv::flann::LshIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::LshIndexParams													•	
flann_Ptr_LshIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_LshIndexParams_delete(cv::Ptr<cv::flann::LshIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::LshIndexParams													•	
flann_Ptr_CompositeIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_CompositeIndexParams_new(int trees, int branching, int iterations, cvflann::flann_centers_init_t centers_init, float cb_index, cv::Ptr<cv::flann::CompositeIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::CompositeIndexParams>(new cv::flann::CompositeIndexParams(trees, branching, iterations, centers_init, cb_index)); END_WRAP }	cv::flann::CompositeIndexParams		the index created combines the randomized kd-trees and the hierarchical k-means tree.								•			
flann_Ptr_CompositeIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_CompositeIndexParams_get(cv::Ptr<cv::flann::CompositeIndexParams> *ptr, cv::flann::CompositeIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::CompositeIndexParams													•	
flann_Ptr_CompositeIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_CompositeIndexParams_delete(cv::Ptr<cv::flann::CompositeIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::CompositeIndexParams													•	

flann_Ptr_AutotunedIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_AutotunedIndexParams_new(float target_precision, float build_weight, float memory_weight, float sample_fraction, cv::Ptr<cv::flann::AutotunedIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::AutotunedIndexParams>(new cv::flann::AutotunedIndexParams(target_precision, build_weight, memory_weight, sample_fraction)); END_WRAP }	cv::flann::AutotunedIndexParams		the index created is automatically tuned to offer the best performance, by choosing the optimal index type (randomized kd-trees, hierarchical kmeans, linear) and parameters for the dataset provided.								•			
flann_Ptr_AutotunedIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_AutotunedIndexParams_get(cv::Ptr<cv::flann::AutotunedIndexParams> *ptr, cv::flann::AutotunedIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::AutotunedIndexParams												•		
flann_Ptr_AutotunedIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_AutotunedIndexParams_delete(cv::Ptr<cv::flann::AutotunedIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::AutotunedIndexParams											•			
flann_Ptr_SavedIndexParams_new	CVAPI(ExceptionStatus) flann_Ptr_SavedIndexParams_new(const char* filename, cv::Ptr<cv::flann::SavedIndexParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::SavedIndexParams>(new cv::flann::SavedIndexParams(filename)); END_WRAP }	cv::flann::SavedIndexParams		loading a previously saved index from the disk.								•			
flann_Ptr_SavedIndexParams_get	CVAPI(ExceptionStatus) flann_Ptr_SavedIndexParams_get(cv::Ptr<cv::flann::SavedIndexParams> *ptr, cv::flann::SavedIndexParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::SavedIndexParams												•		
flann_Ptr_SavedIndexParams_delete	CVAPI(ExceptionStatus) flann_Ptr_SavedIndexParams_delete(cv::Ptr<cv::flann::SavedIndexParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::SavedIndexParams											•			
flann_Ptr_SearchParams_new	CVAPI(ExceptionStatus) flann_Ptr_SearchParams_new(int checks, float eps, int sorted, cv::Ptr<cv::flann::SearchParams> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::flann::SearchParams>(new cv::flann::SearchParams(checks, eps, (sorted != 0))); END_WRAP }	cv::flann::SearchParams										•				
flann_Ptr_SearchParams_get	CVAPI(ExceptionStatus) flann_Ptr_SearchParams_get(cv::Ptr<cv::flann::SearchParams> *ptr, cv::flann::SearchParams **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::flann::SearchParams												•		
flann_Ptr_SearchParams_delete	CVAPI(ExceptionStatus) flann_Ptr_SearchParams_delete(cv::Ptr<cv::flann::SearchParams> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::flann::SearchParams											•			
highgui_namedWindow	CVAPI(ExceptionStatus) highgui_namedWindow(const char *winname, int flags) { BEGIN_WRAP cv::namedWindow(winname, flags); END_WRAP }	cv::namedWindow()	group__highgui.html	Creates a window.	winname : Name of the window in the window caption that may be used as a window identifier. flags : Flags of the window. The supported flags are: (cv::WindowFlags)	The function namedWindow creates a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names.If a window with the same name already exists, the function does nothing.You can call cv::destroyWindow or cv::destroyAllWindows to close the window and de-allocate any associated memory usage. For a simple program, you do not really have to call these functions because all the resources and windows of the application are closed automatically by the operating system upon exit.Note(Qt backend supports additional flags:WINDOW_NORMAL or WINDOW_AUTOSIZE: WINDOW_NORMAL enables you to resize the window, whereas WINDOW_AUTOSIZE adjusts automatically the window size to fit the displayed image (see imshow), and you cannot change the window size manually. WINDOW_FREERATIO or WINDOW_KEEPRATIO: WINDOW_FREERATIO adjusts the image with no respect to its ratio, whereas WINDOW_KEEPRATIO keeps the image	2									
highgui_destroyWindow	CVAPI(ExceptionStatus) highgui_destroyWindow(const char *winName) { BEGIN_WRAP cv::destroyWindow(winName); END_WRAP }	cv::destroyWindow()	group__highgui.html	Destroys the specified window.	winname : Name of the window to be destroyed.	The function destroyWindow destroys the window with the given name.Examples: modules/shape/samples/shape_example.cpp, samples/cpp/camshiftdemo.cpp, samples/cpp/fitellipse.cpp, and samples/cpp/grabcut.cpp.	1									
highgui_destroyAllWindows	CVAPI(ExceptionStatus) highgui_destroyAllWindows() { BEGIN_WRAP cv::destroyAllWindows(); END_WRAP }	cv::destroyAllWindows()	group__highgui.html	Destroys all of the HighGUI windows.		The function destroyAllWindows destroys all of the opened HighGUI windows.	1									
highgui_startWindowThread	CVAPI(ExceptionStatus) highgui_startWindowThread(int *returnValue) { BEGIN_WRAP *returnValue = cv::startWindowThread(); END_WRAP }	cv::startWindowThread()	group__highgui.html				1									
highgui_imshow	CVAPI(ExceptionStatus) highgui_imshow(const char *winname, cv::Mat *mat) { BEGIN_WRAP cv::imshow(winname, *mat); END_WRAP }	cv::imshow()	group__highgui.html	Displays an image in the specified window.	winname : Name of the window. mat : Image to be shown.	The function imshow displays an image in the specified window. If the window was created with the cv::WINDOW_AUTOSIZE flag, the image is shown with its original size, however it is still limited by the screen resolution. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth:If the image is 8-bit unsigned, it is displayed as is. If the image is 16-bit unsigned or 32-bit integer, the pixels are divided by 256. That is, the value range [0,255*256] is mapped to [0,255]. If the image is 32-bit or 64-bit floating-point, the pixel values are multiplied by 255. That is, the value range [0,1] is mapped to [0,255].If window was created with OpenGL support, cv::imshow also support ogl::Buffer , ogl::Texture2D and cuda::GpuMat as input.If the window was not created before this function, it is assumed creating a window with cv::WINDOW_AUTOSIZE.If you need to show an image that is bigger than the screen resolution, you will need to call	2									

highgui_imshow_umat	CVAPI(ExceptionStatus) highgui_imshow_umat(const char* winname, cv::UMat* mat) { BEGIN_WRAP cv::imshow(winname, *mat); END_WRAP }	cv::imshow()	group__highgui.html	Displays an image in the specified window.	winname : Name of the window. mat : Image to be shown.	The function imshow displays an image in the specified window. If the window was created with the cv::WINDOW_AUTOSIZE flag, the image is shown with its original size, however it is still limited by the screen resolution. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth:If the image is 8-bit unsigned, it is displayed as is. If the image is 16-bit unsigned or 32-bit integer, the pixels are divided by 256. That is, the value range [0,255*256] is mapped to [0,255]. If the image is 32-bit or 64-bit floating-point, the pixel values are multiplied by 255. That is, the value range [0,1] is mapped to [0,255].If window was created with OpenGL support, cv::imshow also support ogl::Buffer , ogl::Texture2D and cuda::GpuMat as input.If the window was not created before this function, it is assumed creating a window with cv::WINDOW_AUTOSIZE.If you need to show an image that is bigger than the screen resolution, you will need to call namedWindow("", WINDOW_NORMAL)	2								
highgui_waitKey	CVAPI(ExceptionStatus) highgui_waitKeyEx(int delay, int *returnValue) { BEGIN_WRAP *returnValue = cv::waitKeyEx(delay); END_WRAP }	cv::waitKeyEx()	group__highgui.html	Similar to waitKey, but returns full key code.		NoteKey code is implementation specific and depends on used backend: QT/GTK/Win32/etc	1								
highgui_waitKeyEx	CVAPI(ExceptionStatus) highgui_waitKeyEx(int delay, int *returnValue) { BEGIN_WRAP *returnValue = cv::waitKeyEx(delay); END_WRAP }	cv::waitKeyEx()	group__highgui.html	Similar to waitKey, but returns full key code.		NoteKey code is implementation specific and depends on used backend: QT/GTK/Win32/etc	1								
highgui_resizeWindow	CVAPI(ExceptionStatus) highgui_resizeWindow(const char *winName, int width, int height) { BEGIN_WRAP cv::resizeWindow(winName, width, height); END_WRAP }	cv::resizeWindow()	group__highgui.html	Resizes the window to the specified size.	winname : Window name. width : The new window width. height : The new window height.	NoteThe specified window size is for the image area. Toolbars are not counted. Only windows created without cv::WINDOW_AUTOSIZE flag can be resized.	3								
highgui_moveWindow	CVAPI(ExceptionStatus) highgui_moveWindow(const char *winName, int x, int y) { BEGIN_WRAP cv::moveWindow(winName, x, y); END_WRAP }	cv::moveWindow()	group__highgui.html	Moves the window to the specified position.	winname : Name of the window. x : The new x-coordinate of the window. y : The new y-coordinate of the window.	Examples: modules/shape/samples/shape_example.cpp, samples/cpp/image_alignment.cpp, samples/cpp/polar_transforms.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, and samples/cpp/warpPerspective_demo.cpp.	3								
highgui_setWindowProperty	CVAPI(ExceptionStatus) highgui_setWindowProperty(const char *winName, int propId, double propValue) { BEGIN_WRAP cv::setWindowProperty(winName, propId, propValue); END_WRAP }	cv::setWindowProperty()	group__highgui.html	Changes parameters of a window dynamically.	winname : Name of the window. prop_id : Window property to edit. The supported operation flags are: (cv::WindowPropertyFlags) prop_value : New value of the window property. The supported flags are: (cv::WindowFlags)	The function setWindowProperty enables changing properties of a window.	3								
highgui_setWindowTitle	CVAPI(ExceptionStatus) highgui_setWindowTitle(const char *winname, const char *title) { BEGIN_WRAP // TODO Resolve: #ifndef _WINRT_DLL cv::setWindowTitle(winname, title); #endif END_WRAP }	cv::setWindowTitle()	group__highgui.html	Updates window title.	winname : Name of the window. title : New title.		2								
highgui_getWindowProperty	CVAPI(ExceptionStatus) highgui_getWindowProperty(const char *winName, int propId, double *returnValue) { BEGIN_WRAP *returnValue = cv::getWindowProperty(winName, propId); END_WRAP }	cv::getWindowProperty()	group__highgui.html	Provides parameters of a window.	winname : Name of the window. prop_id : Window property to retrieve. The following operation flags are available: (cv::WindowPropertyFlags)	The function getWindowProperty returns properties of a window.See alsosetWindowProperty	2								
highgui_getWindowImageRect	CVAPI(ExceptionStatus) highgui_getWindowImageRect(const char *winName, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c(cv::getWindowImageRect(winName)); END_WRAP }	cv::getWindowImageRect()	group__highgui.html	Provides rectangle of image in the window.	winname : Name of the window.	The function getWindowImageRect returns the client screen coordinates, width and height of the image rendering area.See alsoresizeWindow moveWindow	1								
highgui_setMouseCallback	CVAPI(ExceptionStatus) highgui_setMouseCallback(const char *winName, cv::MouseCallback onMouse, void* userData) { BEGIN_WRAP cv::setMouseCallback(winName, onMouse, userData); END_WRAP }	cv::setMouseCallback()	group__highgui.html	Sets mouse handler for the specified window.	winname : Name of the window. onMouse : Callback function for mouse events. See OpenCV samples on how to specify and use the callback. userdata : The optional parameter passed to the callback.	Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/create_mask.cpp, samples/cpp/ffilldemo.cpp, samples/cpp/grabcut.cpp, samples/cpp/lkdemo.cpp, samples/cpp/warpPerspective_demo.cpp, and samples/cpp/watershed.cpp.	3								
highgui_getMouseWheelDelta	CVAPI(ExceptionStatus) highgui_getMouseWheelDelta(int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::getMouseWheelDelta(flags); END_WRAP }	cv::getMouseWheelDelta()	group__highgui.html	Gets the mouse-wheel motion delta, when handling mouse-wheel events cv::EVENT_MOUSEWHEEL and cv::EVENT_MOUSEHWHEEL.	flags : The mouse callback flags parameter.	For regular mice with a scroll-wheel, delta will be a multiple of 120. The value 120 corresponds to a one notch rotation of the wheel or the threshold for action to be taken and one such action should occur for each delta. Some high-precision mice with higher-resolution freely-rotating wheels may generate smaller values.For cv::EVENT_MOUSEWHEEL positive and negative values mean forward and backward scrolling, respectively. For cv::EVENT_MOUSEHWHEEL, where available, positive and negative values mean right and left scrolling, respectively.NoteMouse-wheel events are currently supported only on Windows.	1								
highgui_selectROI1	CVAPI(ExceptionStatus) highgui_selectROI1(const char *windowName, cv::_InputArray *img, int showCrosshair, int fromCenter, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c(cv::selectROI(windowName, *img, showCrosshair != 0, fromCenter != 0)); END_WRAP }	cv::selectROI()	group__highgui.html	Allows users to select a ROI on the given image.	windowName : name of the window where selection process will be shown. img : image to select a ROI. showCrosshair : if true crosshair of selection rectangle will be shown. fromCenter : if true center of selection will match initial mouse position. In opposite case a corner of selection rectangle will correspont to the initial mouse position.	The function creates a window and allows users to select a ROI using the mouse. Controls: use space or enter to finish selection, use key c to cancel selection (function will return the zero cv::Rect).NoteThe function sets it's own mouse callback for specified window using cv::setMouseCallback(windowName, ...). After finish of work an empty callback will be set for the used window.	4								
highgui_selectROI2	CVAPI(ExceptionStatus) highgui_selectROI2(cv::_InputArray *img, int showCrosshair, int fromCenter, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c(cv::selectROI(*img, showCrosshair != 0, fromCenter != 0)); END_WRAP }	cv::selectROI()	group__highgui.html	Allows users to select a ROI on the given image.	windowName : name of the window where selection process will be shown. img : image to select a ROI. showCrosshair : if true crosshair of selection rectangle will be shown. fromCenter : if true center of selection will match initial mouse position. In opposite case a corner of selection rectangle will correspont to the initial mouse position.	The function creates a window and allows users to select a ROI using the mouse. Controls: use space or enter to finish selection, use key c to cancel selection (function will return the zero cv::Rect).NoteThe function sets it's own mouse callback for specified window using cv::setMouseCallback(windowName, ...). After finish of work an empty callback will be set for the used window.	3								
highgui_selectROIs	CVAPI(ExceptionStatus) highgui_selectROIs(const char * windowName, cv::_InputArray *img, std::vector<cv::Rect> *boundingBoxes, int showCrosshair, int fromCenter) { BEGIN_WRAP cv::selectROIs(windowName, *img, *boundingBoxes, showCrosshair != 0, fromCenter != 0); END_WRAP }	cv::selectROIs()	group__highgui.html	Allows users to select multiple ROIs on the given image.	windowName : name of the window where selection process will be shown. img : image to select a ROI. boundingBoxes : selected ROIs. showCrosshair : if true crosshair of selection rectangle will be shown. fromCenter : if true center of selection will match initial mouse position. In opposite case a corner of selection rectangle will correspont to the initial mouse position.	The function creates a window and allows users to select multiple ROIs using the mouse. Controls: use space or enter to finish current selection and start a new one, use esc to terminate multiple ROI selection process.NoteThe function sets it's own mouse callback for specified window using cv::setMouseCallback(windowName, ...). After finish of work an empty callback will be set for the used window.	5								

highgui_createTrackbar	CVAPI(ExceptionStatus) highgui_createTrackbar(const char *trackbarName, const char *winName, int* value, int count, cv::TrackbarCallback onChange, void* userData, int *returnValue) { BEGIN_WRAP *returnValue = cv::createTrackbar(trackbarName, winName, value, count, onChange, userData); END_WRAP }	cv::createTrackbar()	group__highgui.html	Creates a trackbar and attaches it to the specified window.	trackbarname : Name of the created trackbar. winname : Name of the window that will be used as a parent of the created trackbar. value : Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable. count : Maximal position of the slider. The minimal position is always 0. onChange : Pointer to the function to be called every time the slider changes position. This function should be prototyped as void Foo(int,void*); , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only value is updated. userdata : User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.	The function createTrackbar creates a trackbar (a slider or range control) with the specified name and range, assigns a variable value to be a position synchronized with the trackbar and specifies the callback function onChange to be called on the trackbar position change. The created trackbar is displayed in the specified window winname.Note[Qt Backend Only] winname can be empty if the trackbar should be attached to the control panel.Clicking the label of each trackbar enables editing the trackbar values manually.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/connected_components.cpp, samples/cpp/contours2.cpp, samples/cpp/demhist.cpp, samples/cpp/distrans.cpp, samples/cpp/edge.cpp, samples/cpp/falsecolor.cpp, samples/cpp/ffilldemo.cpp, samples/cpp/fitellipse.cpp, samples/cpp/laplace.cpp, samples/cpp/pca.cpp, samples/cpp/tutorial_code/HighGUI/AddingTrackbar.cpp	6								
highgui_getTrackbarPos	CVAPI(ExceptionStatus) highgui_getTrackbarPos(const char *trackbarName, const char *winName, int *returnValue) { BEGIN_WRAP *returnValue = cv::getTrackbarPos(trackbarName, winName); END_WRAP }	cv::getTrackbarPos()	group__highgui.html	Returns the trackbar position.	trackbarname : Name of the trackbar. winname : Name of the window that is the parent of the trackbar.	The function returns the current position of the specified trackbar.Note[Qt Backend Only] winname can be empty if the trackbar is attached to the control panel.	2								
highgui_setTrackbarPos	CVAPI(ExceptionStatus) highgui_setTrackbarPos(const char *trackbarName, const char *winName, int pos) { BEGIN_WRAP cv::setTrackbarPos(trackbarName, winName, pos); END_WRAP }	cv::setTrackbarPos()	group__highgui.html	Sets the trackbar position.	trackbarname : Name of the trackbar. winname : Name of the window that is the parent of trackbar. pos : New position.	The function sets the position of the specified trackbar in the specified window.Note[Qt Backend Only] winname can be empty if the trackbar is attached to the control panel.Examples: samples/cpp/falsecolor.cpp.	3								
highgui_setTrackbarMax	CVAPI(ExceptionStatus) highgui_setTrackbarMax(const char *trackbarName, const char *winName, int maxVal) { BEGIN_WRAP cv::setTrackbarMax(trackbarName, winName, maxVal); END_WRAP }	cv::setTrackbarMax()	group__highgui.html	Sets the trackbar maximum position.	trackbarname : Name of the trackbar. winname : Name of the window that is the parent of trackbar. maxval : New maximum position.	The function sets the maximum position of the specified trackbar in the specified window.Note[Qt Backend Only] winname can be empty if the trackbar is attached to the control panel.Examples: samples/cpp/falsecolor.cpp.	3								
highgui_setTrackbarMin	CVAPI(ExceptionStatus) highgui_setTrackbarMin(const char *trackbarName, const char *winName, int minVal) { BEGIN_WRAP cv::setTrackbarMin(trackbarName, winName, minVal); END_WRAP }	cv::setTrackbarMin()	group__highgui.html	Sets the trackbar minimum position.	trackbarname : Name of the trackbar. winname : Name of the window that is the parent of trackbar. minval : New minimum position.	The function sets the minimum position of the specified trackbar in the specified window.Note[Qt Backend Only] winname can be empty if the trackbar is attached to the control panel.Examples: samples/cpp/falsecolor.cpp.	3								
highgui_cvGetWindowHandle	CVAPI(ExceptionStatus) highgui_cvGetWindowHandle(const char* name, void **returnValue) { BEGIN_WRAP *returnValue = cvGetWindowHandle(name); END_WRAP }	cvGetWindowHandle()														
imgcodecs_imread_NotWindows	CVAPI(ExceptionStatus) imgcodecs_imread(const char *filename, int flags, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::imread(filename, flags); *returnValue = new cv::Mat(ret); END_WRAP }	cv::imread()	group__imgcodecs.html	Loads an image from a file.	filename : Name of file to be loaded. flags : Flag that can take values of cv::ImreadModes	The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL).Currently, the following file formats are supported:Windows bitmaps - *.bmp, *.dib (always supported) JPEG files - *.jpeg, *.jpg, *.jpe (see the Note section) JPEG 2000 files - *.jp2 (see the Note section) Portable Network Graphics - *.png (see the Note section) WebP - *.webp (see the Note section) Portable image format - *.pbm, *.pgm, *.ppm *.pam, *.pnm (always supported) PFM files - *.pfm (see the Note section) Sun rasters - *.sr, *.ras (always supported) TIFF files - *.tiff, *.tif (see the Note section) OpenEXR Image files - *.exr (see the Note section) Radiance HDR - *.hdr, *.pic (always supported)	2								
imgcodecs_imread_Windows	CVAPI(ExceptionStatus) imgcodecs_imread(const char *filename, int flags, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::imread(filename, flags); *returnValue = new cv::Mat(ret); END_WRAP }	cv::imread()	group__imgcodecs.html	Loads an image from a file.	filename : Name of file to be loaded. flags : Flag that can take values of cv::ImreadModes	The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL).Currently, the following file formats are supported:Windows bitmaps - *.bmp, *.dib (always supported) JPEG files - *.jpeg, *.jpg, *.jpe (see the Note section) JPEG 2000 files - *.jp2 (see the Note section) Portable Network Graphics - *.png (see the Note section) WebP - *.webp (see the Note section) Portable image format - *.pbm, *.pgm, *.ppm *.pam, *.pnm (always supported) PFM files - *.pfm (see the Note section) Sun rasters - *.sr, *.ras (always supported) TIFF files - *.tiff, *.tif (see the Note section) OpenEXR Image files - *.exr (see the Note section) Radiance HDR - *.hdr, *.pic (always supported)	2								
imgcodecs_imreadmulti_NotWindows	CVAPI(ExceptionStatus) imgcodecs_imreadmulti(const char *filename, std::vector<cv::Mat> *mats, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::imreadmulti(filename, *mats, flags) ? 1 : 0; END_WRAP }	cv::imreadmulti()	group__imgcodecs.html	Loads a multi-page image from a file.	filename : Name of file to be loaded. flags : Flag that can take values of cv::ImreadModes, default with cv::IMREAD_ANYCOLOR. mats : A vector of Mat objects holding each page, if more than one.	The function imreadmulti loads a multi-page image from the specified file into a vector of Mat objects.See alsocv::imread	3								
imgcodecs_imreadmulti_Windows	CVAPI(ExceptionStatus) imgcodecs_imreadmulti(const char *filename, std::vector<cv::Mat> *mats, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::imreadmulti(filename, *mats, flags) ? 1 : 0; END_WRAP }	cv::imreadmulti()	group__imgcodecs.html	Loads a multi-page image from a file.	filename : Name of file to be loaded. flags : Flag that can take values of cv::ImreadModes, default with cv::IMREAD_ANYCOLOR. mats : A vector of Mat objects holding each page, if more than one.	The function imreadmulti loads a multi-page image from the specified file into a vector of Mat objects.See alsocv::imread	3								
imgcodecs_imwrite_NotWindows	CVAPI(ExceptionStatus) imgcodecs_imwrite(const char *filename, cv::Mat *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PFM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be saved in TIFF format (see the code	3								

imgcodecs_imwrite_Windows	CVAPI(ExceptionStatus) imgcodecs_imwrite(const char *filename, cv::Mat *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PPM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be	3								
imgcodecs_imwrite_multi_NotWindows	CVAPI(ExceptionStatus) imgcodecs_imwrite_multi(const char *filename, std::vector<cv::Mat> *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PPM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be	3								
imgcodecs_imwrite_multi_Windows	CVAPI(ExceptionStatus) imgcodecs_imwrite_multi(const char *filename, std::vector<cv::Mat> *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PPM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be	3								
imgcodecs_imdecode_Mat	CVAPI(ExceptionStatus) imgcodecs_imdecode_Mat(cv::Mat *buf, int flags, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::imdecode(*buf, flags); *returnValue = new cv::Mat(ret); END_WRAP }	cv::imdecode()	group__imgcodecs.html	Reads an image from a buffer in memory.	buf : Input array or vector of bytes. flags : The same flags as in cv::imread, see cv::ImreadModes.	The function imdecode reads an image from the specified buffer in the memory. If the buffer is too short or contains invalid data, the function returns an empty matrix (Mat::data==NULL).See cv::imread for the list of supported formats and flags description.NoteIn the case of color images, the decoded images will have the channels stored in B G R order.	2								
imgcodecs_imdecode_vector	CVAPI(ExceptionStatus) imgcodecs_imdecode_vector(uchar *buf, int bufLength, int flags, cv::Mat **returnValue) { BEGIN_WRAP //const std::vector<uchar> bufVec(buf, buf + bufLength); const cv::Mat bufMat(1, bufLength, CV_8UC1, buf, cv::Mat::AUTO_STEP); const auto ret = cv::imdecode(bufMat, flags); *returnValue = new cv::Mat(ret); END_WRAP }	cv::imdecode()	group__imgcodecs.html	Reads an image from a buffer in memory.	buf : Input array or vector of bytes. flags : The same flags as in cv::imread, see cv::ImreadModes.	The function imdecode reads an image from the specified buffer in the memory. If the buffer is too short or contains invalid data, the function returns an empty matrix (Mat::data==NULL).See cv::imread for the list of supported formats and flags description.NoteIn the case of color images, the decoded images will have the channels stored in B G R order.	2								
imgcodecs_imdecode_InputArray	CVAPI(ExceptionStatus) imgcodecs_imdecode_InputArray(cv::_InputArray *buf, int flags, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::imdecode(*buf, flags); *returnValue = new cv::Mat(ret); END_WRAP }	cv::imdecode()	group__imgcodecs.html	Reads an image from a buffer in memory.	buf : Input array or vector of bytes. flags : The same flags as in cv::imread, see cv::ImreadModes.	The function imdecode reads an image from the specified buffer in the memory. If the buffer is too short or contains invalid data, the function returns an empty matrix (Mat::data==NULL).See cv::imread for the list of supported formats and flags description.NoteIn the case of color images, the decoded images will have the channels stored in B G R order.	2								
imgcodecs_imencode_vector	CVAPI(ExceptionStatus) imgcodecs_imencode_vector(const char *ext, cv::_InputArray *img, std::vector<uchar> *buf, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; if (params != nullptr) paramsVec = std::vector<int>(params, params + paramsLength); *returnValue = cv::imencode(ext, *img, *buf, paramsVec) ? 1 : 0; END_WRAP }	cv::imencode()	group__imgcodecs.html	Encodes an image into a memory buffer.	ext : File extension that defines the output format. img : Image to be written. buf : Output buffer resized to fit the compressed image. params : Format-specific parameters. See cv::imwrite and cv::ImwriteFlags.	The function imencode compresses the image and stores it in the memory buffer that is resized to fit the result. See cv::imwrite for the list of supported formats and flags description.	4								
imgcodecs_haveImageReader_NotWindows	CVAPI(ExceptionStatus) imgcodecs_imwrite(const char *filename, cv::Mat *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PPM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be	3								
imgcodecs_haveImageReader_Windows	CVAPI(ExceptionStatus) imgcodecs_imwrite(const char *filename, cv::Mat *img, int *params, int paramsLength, int *returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); *returnValue = cv::imwrite(filename, *img, paramsVec) ? 1 : 0; END_WRAP }	cv::imwrite()	group__imgcodecs.html	Saves an image to a specified file.	filename : Name of the file. img : (Mat or vector of Mat) Image or Images to be saved. params : Format-specific parameters encoded as pairs (paramId_1, paramValue_1, paramId_2, paramValue_2, ...) see cv::ImwriteFlags	The function imwrite saves the image to the specified file. The image format is chosen based on the filename extension (see cv::imread for the list of extensions). In general, only 8-bit single-channel or 3-channel (with 'BGR' channel order) images can be saved using this function, with these exceptions:16-bit unsigned (CV_16U) images can be saved in the case of PNG, JPEG 2000, and TIFF formats 32-bit float (CV_32F) images can be saved in PPM, TIFF, OpenEXR, and Radiance HDR formats; 3-channel (CV_32FC3) TIFF images will be saved using the LogLuv high dynamic range encoding (4 bytes per pixel) PNG images with an alpha channel can be saved using this function. To do this, create 8-bit (or 16-bit) 4-channel image BGRA, where the alpha channel goes last. Fully transparent pixels should have alpha set to 0, fully opaque pixels should have alpha set to 255/65535 (see the code sample below). Multiple images (vector of Mat) can be	3								

imgcodecs_haveImageWriter	CVAPI(ExceptionStatus) imgcodecs_haveImageWriter(const char *filename, int *returnValue) { BEGIN_WRAP *returnValue = cv::haveImageWriter(filenam e) ? 1 : 0; END_WRAP }	cv::haveImageWriter()	group__imgcodecs.html	Returns true if an image with the specified filename can be encoded by OpenCV.	filename : File name of the image		1									
img_hash_ImgHashBase_compute	CVAPI(ExceptionStatus) img_hash_ImgHashBase_com pute(cv::img_hash::ImgHash Base *obj, cv::_InputArray *inputArr, cv::_OutputArray *outputArr) { BEGIN_WRAP obj->compute(*inputArr, *outputArr); END_WRAP }	cv::img_hash::ImgHashBase::compute()	classcv_1_1img__hash_1_1ImgHas hBase.html	Computes hash of the input image.	inputArr : input image want to compute hash value outputArr : hash of the image		2									
img_hash_ImgHashBase_compare	CVAPI(ExceptionStatus) img_hash_ImgHashBase_com pare(cv::img_hash::ImgHash Base *obj, cv::_InputArray *hashOne, cv::_InputArray *hashTwo, double *returnValue) { BEGIN_WRAP *returnValue = obj- >compare(*hashOne, *hashTwo); END_WRAP }	cv::img_hash::ImgHashBase::compare()	classcv_1_1img__hash_1_1ImgHas hBase.html	Compare the hash value between inOne and inTwo.	hashOne : Hash value one hashTwo : Hash value two		2									
img_hash_AverageHash_create	CVAPI(ExceptionStatus) img_hash_AverageHash_creat e(cv::Ptr<cv::img_hash::Aver ageHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::AverageHash:: create(); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::AverageHash::create()	classcv_1_1img__hash_1_1Average Hash.html				1									
img_hash_Ptr_AverageHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_AverageHash_d elete(cv::Ptr<cv::img_hash:: AverageHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::AverageHash	classcv_1_1img__hash_1_1Average Hash.html	Computes average hash value of the input image. This is a fast image hashing algorithm, but only work on simple case. For more details, please refer to [130]								•				
img_hash_Ptr_AverageHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_AverageHash_g et(cv::Ptr<cv::img_hash::Ave rageHash> *ptr, cv::img_hash::AverageHash **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::AverageHash	classcv_1_1img__hash_1_1Average Hash.html	Computes average hash value of the input image. This is a fast image hashing algorithm, but only work on simple case. For more details, please refer to [130]									•			
img_hash_BlockMeanHash_create	CVAPI(ExceptionStatus) img_hash_BlockMeanHash_cr eate(const int mode, cv::Ptr<cv::img_hash::Block MeanHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::BlockMeanHas h::create(mode); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::BlockMeanHash::create()	classcv_1_1img__hash_1_1BlockMe anHash.html				1									
img_hash_Ptr_BlockMeanHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_BlockMeanHash _delete(cv::Ptr<cv::img_hash ::BlockMeanHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::BlockMeanHash	classcv_1_1img__hash_1_1BlockMe anHash.html	Image hash based on block mean. See [289] for details.								•				
img_hash_Ptr_BlockMeanHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_BlockMeanHash _get(cv::Ptr<cv::img_hash::B lockMeanHash> *ptr, cv::img_hash::BlockMeanHas h **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::BlockMeanHash	classcv_1_1img__hash_1_1BlockMe anHash.html	Image hash based on block mean. See [289] for details.									•			
img_hash_BlockMeanHash_setMode	CVAPI(ExceptionStatus) img_hash_BlockMeanHash_se tMode(cv::img_hash::BlockM eanHash *obj, const int mode) { BEGIN_WRAP obj->setMode(mode); END_WRAP }	cv::img_hash::BlockMeanHash::setMode()	classcv_1_1img__hash_1_1BlockMe anHash.html	Create BlockMeanHash object.	mode : the mode		1									
img_hash_BlockMeanHash_getMean	CVAPI(ExceptionStatus) img_hash_BlockMeanHash_ge tMean(cv::img_hash::BlockM eanHash *obj, std::vector<double> *outVec) { BEGIN_WRAP const auto mean = obj- >getMean(); outVec->clear(); outVec- >assign(mean.begin(), mean.end()); END_WRAP }	cv::img_hash::BlockMeanHash::getMean()	classcv_1_1img__hash_1_1BlockMe anHash.html				2									
img_hash_ColorMomentHash_create	CVAPI(ExceptionStatus) img_hash_ColorMomentHash_ create(cv::Ptr<cv::img_hash: :ColorMomentHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::ColorMomentH ash::create(); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::ColorMomentHash::create()	classcv_1_1img__hash_1_1ColorMo mentHash.html				1									
img_hash_Ptr_ColorMomentHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_ColorMomentH ash_delete(cv::Ptr<cv::img_h ash::ColorMomentHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::ColorMomentHash	classcv_1_1img__hash_1_1ColorMo mentHash.html	Image hash based on color moments. See [238] for details.								•				
img_hash_Ptr_ColorMomentHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_ColorMomentH ash_get(cv::Ptr<cv::img_has h::ColorMomentHash> *ptr, cv::img_hash::ColorMomentH ash **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::ColorMomentHash	classcv_1_1img__hash_1_1ColorMo mentHash.html	Image hash based on color moments. See [238] for details.									•			
img_hash_MarrHildrethHash_create	CVAPI(ExceptionStatus) img_hash_MarrHildrethHash_c reate(const float alpha, const float scale, cv::Ptr<cv::img_hash::MarrHi ldrethHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::MarrHildrethHa sh::create(alpha, scale); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::MarrHildrethHash::create()	classcv_1_1img__hash_1_1MarrHild rethHash.html		alpha : int scale factor for marr wavelet (default=2). scale : int level of scale factor (default = 1)		2									
img_hash_Ptr_MarrHildrethHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_MarrHildrethHa sh_delete(cv::Ptr<cv::img_ha sh::MarrHildrethHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::MarrHildrethHash	classcv_1_1img__hash_1_1MarrHild rethHash.html	Marr-Hildreth Operator Based Hash, slowest but more discriminative. See [289] for details.								•				

img_hash_Ptr_MarrHildrethHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_MarrHildrethHash_get(cv::Ptr<cv::img_hash::MarrHildrethHash> *ptr, cv::img_hash::MarrHildrethHash **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::MarrHildrethHash	classcv_1_1img__hash_1_1MarrHildrethHash.html	Marr-Hildreth Operator Based Hash, slowest but more discriminative. See [289] for details.										•	
img_hash_MarrHildrethHash_setKernelParam	CVAPI(ExceptionStatus) img_hash_MarrHildrethHash_setKernelParam(cv::img_hash::MarrHildrethHash *obj, const float alpha, const float scale) { BEGIN_WRAP obj->setKernelParam(alpha, scale); END_WRAP }	cv::img_hash::MarrHildrethHash::setKernelParam()	classcv_1_1img__hash_1_1MarrHildrethHash.html	Set Mh kernel parameters.	alpha : int scale factor for marr wavelet (default=2). scale : int level of scale factor (default = 1)		2									
img_hash_MarrHildrethHash_getAlpha	CVAPI(ExceptionStatus) img_hash_MarrHildrethHash_getAlpha(cv::img_hash::MarrHildrethHash *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getAlpha(); END_WRAP }	cv::img_hash::MarrHildrethHash::getAlpha()	classcv_1_1img__hash_1_1MarrHildrethHash.html	self explain			1									
img_hash_MarrHildrethHash_getScale	CVAPI(ExceptionStatus) img_hash_MarrHildrethHash_getScale(cv::img_hash::MarrHildrethHash *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getScale(); END_WRAP }	cv::img_hash::MarrHildrethHash::getScale()	classcv_1_1img__hash_1_1MarrHildrethHash.html	self explain			1									
img_hash_PHash_create	CVAPI(ExceptionStatus) img_hash_PHash_create(cv::Ptr<cv::img_hash::PHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::PHash::create(); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::PHash::create()	classcv_1_1img__hash_1_1PHash.html				1									
img_hash_Ptr_PHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_PHash_delete(cv::Ptr<cv::img_hash::PHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::Phas												•		
img_hash_Ptr_PHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_PHash_get(cv::Ptr<cv::img_hash::PHash> *ptr, cv::img_hash::PHash **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::Phas													•	
img_hash_RadialVarianceHash_create	CVAPI(ExceptionStatus) img_hash_RadialVarianceHash_create(const double sigma, const int numOfAngleLine, cv::Ptr<cv::img_hash::RadialVarianceHash> **returnValue) { BEGIN_WRAP const auto ptr = cv::img_hash::RadialVarianceHash::create(sigma, numOfAngleLine); *returnValue = clone(ptr); END_WRAP }	cv::img_hash::RadialVarianceHash::create()	classcv_1_1img__hash_1_1RadialVarianceHash.html				2									
img_hash_Ptr_RadialVarianceHash_delete	CVAPI(ExceptionStatus) img_hash_Ptr_RadialVarianceHash_delete(cv::Ptr<cv::img_hash::RadialVarianceHash> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::img_hash::RadialVarianceHash	classcv_1_1img__hash_1_1RadialVarianceHash.html	Image hash based on Radon transform. See [238] for details.										•		
img_hash_Ptr_RadialVarianceHash_get	CVAPI(ExceptionStatus) img_hash_Ptr_RadialVarianceHash_get(cv::Ptr<cv::img_hash::RadialVarianceHash> *ptr, cv::img_hash::RadialVarianceHash **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::img_hash::RadialVarianceHash	classcv_1_1img__hash_1_1RadialVarianceHash.html	Image hash based on Radon transform. See [238] for details.											•	
img_hash_RadialVarianceHash_setNumOfAngle	CVAPI(ExceptionStatus) img_hash_RadialVarianceHash_setNumOfAngleLine(cv::img_hash::RadialVarianceHash *obj, const int value) { BEGIN_WRAP obj->setNumOfAngleLine(value); END_WRAP }	cv::img_hash::RadialVarianceHash::setNumOfAngleLine()	classcv_1_1img__hash_1_1RadialVarianceHash.html				1									
img_hash_RadialVarianceHash_setSigma	CVAPI(ExceptionStatus) img_hash_RadialVarianceHash_setSigma(cv::img_hash::RadialVarianceHash *obj, const double value) { BEGIN_WRAP obj->setSigma(value); END_WRAP }	cv::img_hash::RadialVarianceHash::setSigma()	classcv_1_1img__hash_1_1RadialVarianceHash.html				1									
img_hash_RadialVarianceHash_getNumOfAngle	CVAPI(ExceptionStatus) img_hash_RadialVarianceHash_getNumOfAngleLine(cv::img_hash::RadialVarianceHash *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNumOfAngleLine(); END_WRAP }	cv::img_hash::RadialVarianceHash::getNumOfAngleLine()	classcv_1_1img__hash_1_1RadialVarianceHash.html				1									
img_hash_RadialVarianceHash_getSigma	CVAPI(ExceptionStatus) img_hash_RadialVarianceHash_getSigma(cv::img_hash::RadialVarianceHash *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getSigma(); END_WRAP }	cv::img_hash::RadialVarianceHash::getSigma()	classcv_1_1img__hash_1_1RadialVarianceHash.html				1									
line_descriptor_LSDDetector_new1	CVAPI(ExceptionStatus) line_descriptor_LSDDetector_new1(cv::line_descriptor::LSDDetector** returnValue) { BEGIN_WRAP *returnValue = new cv::line_descriptor::LSDDetector; END_WRAP }	cv::line_descriptor::LSDDetector											•			
line_descriptor_LSDDetector_new2	CVAPI(ExceptionStatus) line_descriptor_LSDDetector_new2(const double scale, const double sigma_scale, const double quant, const double ang_th, const double log_eps, const double density_th, const int n_bins, cv::line_descriptor::LSDDetector** returnValue) { BEGIN_WRAP cv::line_descriptor::LSDParam param; param.scale = scale; param.sigma_scale = sigma_scale; param.quant = quant; param.ang_th = ang_th; param.log_eps = log_eps; param.density_th = density_th; param.n_bins = n_bins;	cv::line_descriptor::LSDDetector											•			

line_descriptor_LSDDetector_delete	CVAPI(ExceptionStatus) line_descriptor_LSDDetector_delete(cv::line_descriptor::LSDDetector* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::line_descriptor::LSDDetector												•		
line_descriptor_LSDDetector_detect1	CVAPI(ExceptionStatus) line_descriptor_LSDDetector_detect1(cv::line_descriptor::LSDDetector* obj, cv::Mat* image, std::vector<cv::line_descriptor::KeyLine> *keypoints, int scale, int numOctaves, cv::Mat* mask) { BEGIN_WRAP obj->detect(*image, *keypoints, scale, numOctaves, entity(mask)); END_WRAP }	cv::line_descriptor::LSDDetector::detect()	classcv_1_1line__descriptor_1_1LSDDetector.html	Detect lines inside an image.	image : input image keypoints : vector that will store extracted lines for one or more images scale : scale factor used in pyramids generation numOctaves : number of octaves inside pyramid mask : mask matrix to detect only KeyLines of interest		5								
line_descriptor_LSDDetector_detect2	CVAPI(ExceptionStatus) line_descriptor_LSDDetector_detect2(cv::line_descriptor::LSDDetector* obj, cv::Mat **images, int32_t imagesSize, std::vector<std::vector<cv::line_descriptor::KeyLine> >*keylines, int scale, int numOctaves, cv::Mat** masks, int32_t masksSize) { BEGIN_WRAP std::vector<cv::Mat> imagesVec(imagesSize); std::vector<cv::Mat> masksVec(masksSize); for (int i = 0; i < imagesSize; i++) { imagesVec[i] = *images[i];	cv::line_descriptor::LSDDetector::detect()	classcv_1_1line__descriptor_1_1LSDDetector.html	Detect lines inside an image.	image : input image keypoints : vector that will store extracted lines for one or more images scale : scale factor used in pyramids generation numOctaves : number of octaves inside pyramid mask : mask matrix to detect only KeyLines of interest		5								
optflow_motempl_updateMotionHistory	CVAPI(ExceptionStatus) optflow_motempl_updateMotionHistory(cv::_InputArray *silhouette, cv::_InputOutputArray *mhi, double timestamp, double duration) { BEGIN_WRAP cv::motempl::updateMotionHistory(*silhouette, *mhi, timestamp, duration); END_WRAP }	cv::motempl::updateMotionHistory()	group__optflow.html	Updates the motion history image by a moving silhouette.	silhouette : Silhouette mask that has non-zero pixels where the motion occurs. mhi : Motion history image that is updated by the function (single-channel, 32-bit floating-point). timestamp : Current time in milliseconds or other units. duration : Maximal duration of the motion track in the same units as timestamp .	The function updates the motion history image as follows:¥[¥texttt{mhi} (x,y)= ¥fork{¥texttt{timestamp}}{if ¥(¥texttt{silhouette}(x,y) ¥ne 0¥)}{0}{if ¥(¥texttt{silhouette}(x,y) = 0¥) and ¥(¥texttt{mhi} < (¥texttt{timestamp} - ¥texttt{duration})¥)}{¥texttt{mhi}(x,y)}{otherwise}¥]That is, MHI pixels where the motion occurs are set to the current timestamp , while the pixels where the motion happened last time a long time ago are cleared.The function, together with calcMotionGradient and calcGlobalOrientation , implements a motion templates technique described in [53] and [32] .	4								
optflow_motempl_calcMotionGradient	CVAPI(ExceptionStatus) optflow_motempl_calcMotionGradient(cv::_InputArray *mhi, cv::_OutputArray *mask, cv::_OutputArray *orientation, double delta1, double delta2, int apertureSize) { BEGIN_WRAP cv::motempl::calcMotionGradient(*mhi, *mask, *orientation, delta1, delta2, apertureSize); END_WRAP }	cv::motempl::calcMotionGradient()	group__optflow.html	Calculates a gradient orientation of a motion history image.	mhi : Motion history single-channel floating-point image. mask : Output mask image that has the type CV_8UC1 and the same size as mhi . Its non-zero elements mark pixels where the motion gradient data is correct. orientation : Output motion gradient orientation image that has the same type and the same size as mhi . Each pixel of the image is a motion orientation, from 0 to 360 degrees. delta1 : Minimal (or maximal) allowed difference between mhi values within a pixel neighborhood. delta2 : Maximal (or minimal) allowed difference between mhi values within a pixel neighborhood. That is, the function finds the minimum (¥(m(x,y)¥)) and maximum (¥(M(x,y)¥)) mhi values over ¥(3 ¥times 3¥) neighborhood of each pixel and marks the motion orientation at ¥((x, y)¥) as valid only if ¥(¥min (¥texttt{delta1} , ¥texttt{delta2}) ¥le M(x,y)-m(x,y) ¥le ¥max (¥texttt{delta1} , ¥texttt{delta2}).¥] apertureSize : Aperture size of the Sobel operator.	The function calculates a gradient orientation at each pixel ¥((x, y)¥) as:¥[¥texttt{orientation} (x,y)= ¥arctan(¥frac{d¥texttt{mhi}/dy}{d¥texttt{mhi}/dx})¥]In fact, fastAtan2 and phase are used so that the computed angle is measured in degrees and covers the full range 0..360. Also, the mask is filled to indicate pixels where the computed angle is valid.Note (Python) An example on how to perform a motion template technique can be found at opencv_source_code/samples/python2/motempl.py	6								
optflow_motempl_calcGlobalOrientation	CVAPI(ExceptionStatus) optflow_motempl_calcGlobalOrientation(cv::_InputArray *orientation, cv::_InputArray *mask, cv::_InputArray *mhi, double timestamp, double duration, double *returnValue) { BEGIN_WRAP *returnValue = cv::motempl::calcGlobalOrientation(*orientation, *mask, *mhi, timestamp, duration); END_WRAP }	cv::motempl::calcGlobalOrientation()	group__optflow.html	Calculates a global motion orientation in a selected region.	orientation : Motion gradient orientation image calculated by the function calcMotionGradient mask : Mask image. It may be a conjunction of a valid gradient mask, also calculated by calcMotionGradient , and the mask of a region whose direction needs to be calculated. mhi : Motion history image calculated by updateMotionHistory . timestamp : Timestamp passed to updateMotionHistory . duration : Maximum duration of a motion track in milliseconds, passed to updateMotionHistory	The function calculates an average motion direction in the selected region and returns the angle between 0 degrees and 360 degrees. The average direction is computed from the weighted orientation histogram, where a recent motion has a larger weight and the motion occurred in the past has a smaller weight, as recorded in mhi .	5								
optflow_motempl_segmentMotion	CVAPI(ExceptionStatus) optflow_motempl_segmentMotion(cv::_InputArray *mhi, cv::_OutputArray *segmask, std::vector<cv::Rect> *boundingRects, double timestamp, double segThresh) { BEGIN_WRAP cv::motempl::segmentMotion(*mhi, *segmask, *boundingRects, timestamp, segThresh); END_WRAP }	cv::motempl::segmentMotion()	group__optflow.html	Splits a motion history image into a few parts corresponding to separate independent motions (for example, left hand, right hand).	mhi : Motion history image. segmask : Image where the found mask should be stored, single-channel, 32-bit floating-point. boundingRects : Vector containing ROIs of motion connected components. timestamp : Current time in milliseconds or other units. segThresh : Segmentation threshold that is recommended to be equal to the interval between motion history "steps" or greater.	The function finds all of the motion segments and marks them in segmask with individual values (1,2,...). It also computes a vector with ROIs of motion connected components. After that the motion direction for every component can be calculated with calcGlobalOrientation using the extracted mask of the particular component.	5								
optflow_calcOpticalFlowSF1	CVAPI(ExceptionStatus) optflow_calcOpticalFlowSF1(cv::_InputArray *from, cv::_InputArray *to, cv::_OutputArray *flow, int layers, int averagingBlockSize, int maxFlow) { BEGIN_WRAP cv::optflow::calcOpticalFlowSF(*from, *to, *flow, layers, averagingBlockSize, maxFlow); END_WRAP }	cv::optflow::calcOpticalFlowSF()	group__optflow.html	Calculate an optical flow using "SimpleFlow" algorithm.	from : First 8-bit 3-channel image. to : Second 8-bit 3-channel image of the same size as prev flow : computed flow image that has the same size as prev and type CV_32FC2 layers : Number of layers averaging_block_size : Size of block through which we sum up when calculate cost function for pixel max_flow : maximal flow that we search at each level sigma_dist : vector smooth spatial sigma parameter sigma_color : vector smooth color sigma parameter postprocess_window : window size for postprocess cross bilateral filter sigma_dist_fix : spatial sigma for postprocess cross bilateralf filter sigma_color_fix : color sigma for postprocess cross bilateral filter occ_thr : threshold for detecting occlusions upscale_averaging_radius : window size for bilateral upscale operation upscale_sigma_dist : spatial sigma for bilateral upscale operation upscale_sigma_color : color sigma for bilateral upscale operation speed_up_thr : threshold to detect point with irregular flow - where flow should be recalculated after upscale	See [239] . And site of project - http://graphics.berkeley.edu/papers/Tao-SAN-2012-05/.Note An example using the simpleFlow algorithm can be found at samples/simpleflow_demo.cpp	6								
optflow_calcOpticalFlowSF2	CVAPI(ExceptionStatus) optflow_calcOpticalFlowSF2(cv::_InputArray *from, cv::_InputArray *to, cv::_OutputArray *flow, int layers, int averagingBlockSize, int maxFlow, double sigmaDist, double sigmaColor, int postprocessWindow, double sigmaDistFix, double sigmaColorFix, double occThr, int upscaleAveragingRadius, double upscaleSigmaDist, double upscaleSigmaColor, double speedUpThr) { BEGIN_WRAP cv::optflow::calcOpticalFlowSF(*from, *to, *flow, layers, averagingBlockSize, maxFlow, sigmaDist, sigmaColor, postprocessWindow,	cv::optflow::calcOpticalFlowSF()	group__optflow.html	Calculate an optical flow using "SimpleFlow" algorithm.	from : First 8-bit 3-channel image. to : Second 8-bit 3-channel image of the same size as prev flow : computed flow image that has the same size as prev and type CV_32FC2 layers : Number of layers averaging_block_size : Size of block through which we sum up when calculate cost function for pixel max_flow : maximal flow that we search at each level sigma_dist : vector smooth spatial sigma parameter sigma_color : vector smooth color sigma parameter postprocess_window : window size for postprocess cross bilateral filter sigma_dist_fix : spatial sigma for postprocess cross bilateralf filter sigma_color_fix : color sigma for postprocess cross bilateral filter occ_thr : threshold for detecting occlusions upscale_averaging_radius : window size for bilateral upscale operation upscale_sigma_dist : spatial sigma for bilateral upscale operation upscale_sigma_color : color sigma for bilateral upscale operation speed_up_thr : threshold to detect point with irregular flow - where flow should be recalculated after upscale	See [239] . And site of project - http://graphics.berkeley.edu/papers/Tao-SAN-2012-05/.Note An example using the simpleFlow algorithm can be found at samples/simpleflow_demo.cpp	16								
optflow_calcOpticalFlowSparseToDense	CVAPI(ExceptionStatus) optflow_calcOpticalFlowSparseToDense(cv::_InputArray *from, cv::_InputArray *to, cv::_OutputArray *flow, int grid_step, int k, float sigma, int use_post_proc, float fgs_lambda, float fgs_sigma) { BEGIN_WRAP cv::optflow::calcOpticalFlowSparseToDense(*from, *to, *flow, grid_step, k, sigma, use_post_proc != 0, fgs_lambda, fgs_sigma); END_WRAP }	cv::optflow::calcOpticalFlowSparseToDense()	group__optflow.html	Fast dense optical flow based on PyrLK sparse matches interpolation.	from : first 8-bit 3-channel or 1-channel image. to : second 8-bit 3-channel or 1-channel image of the same size as from flow : computed flow image that has the same size as from and CV_32FC2 type grid_step : stride used in sparse match computation. Lower values usually result in higher quality but slow down the algorithm. k : number of nearest-neighbor matches considered, when fitting a locally affine model. Lower values can make the algorithm noticeably faster at the cost of some quality degradation. sigma : parameter defining how fast the weights decrease in the locally-weighted affine fitting. Higher values can help preserve fine details, lower values can help to get rid of the noise in the output flow. use_post_proc : defines whether the ximgproc::fastGlobalSmootherFilter() is used for post-processing after interpolation fgs_lambda : see the respective parameter of the ximgproc::fastGlobalSmootherFilter() fgs_sigma : see the respective parameter of the ximgproc::fastGlobalSmootherFilter()		9								

quality_QualityBase_compute	CVAPI(ExceptionStatus) quality_QualityBase_compute(cv::quality::QualityBase *obj, cv::_InputArray *img, MyCvScalar *returnValue) { BEGIN_WRAP const auto ret = obj->compute(*img); *returnValue = c(ret); END_WRAP }	cv::quality::QualityBase::compute()															
quality_QualityBase_getQualityMap	CVAPI(ExceptionStatus) quality_QualityBase_getQualityMap(cv::quality::QualityBase *obj, cv::_OutputArray *dst) { BEGIN_WRAP obj->getQualityMap(*dst); END_WRAP }	cv::quality::QualityBase::getQualityMap()	classcv_1_1quality_1_1QualityBase.html	Returns output quality map that was generated during computation, if supported by the algorithm.			1									
quality_QualityBase_clear	CVAPI(ExceptionStatus) quality_QualityBase_clear(cv::quality::QualityBase *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::quality::QualityBase::clear()	classcv_1_1quality_1_1QualityBase.html	Implements Algorithm::clear()		Reimplemented from cv::Algorithm.Reimplemented in cv::quality::QualityPSNR, cv::quality::QualityGMSD, cv::quality::QualitySSIM, and cv::quality::QualityMSE.	1								
quality_QualityBase_empty	CVAPI(ExceptionStatus) quality_QualityBase_empty(cv::quality::QualityBase *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::quality::QualityBase::empty()	classcv_1_1quality_1_1QualityBase.html	Implements Algorithm::empty()		Reimplemented from cv::Algorithm.Reimplemented in cv::quality::QualityPSNR, cv::quality::QualityGMSD, cv::quality::QualitySSIM, and cv::quality::QualityMSE.	1								
quality_createQualityPSNR	CVAPI(ExceptionStatus) quality_createQualityPSNR(cv::_InputArray *ref, double maxPixelValue, cv::Ptr<cv::quality::QualityPSNR> **returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualityPSNR::create(*ref, maxPixelValue); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualityPSNR::create()	classcv_1_1quality_1_1QualityPSNR.html	Create an object which calculates quality.	ref : input image to use as the source for comparison maxPixelValue : maximum per-channel value for any individual pixel; eg 255 for uint8 image		2								
quality_Ptr_QualityPSNR_delete	CVAPI(ExceptionStatus) quality_Ptr_QualityPSNR_delete(cv::Ptr<cv::quality::QualityPSNR> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::quality::QualityPSNR	classcv_1_1quality_1_1QualityPSNR.html	Full reference peak signal to noise ratio (PSNR) algorithm https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.								•				
quality_QualityPSNR_staticCompute	CVAPI(ExceptionStatus) quality_QualityPSNR_staticCompute(cv::_InputArray *ref, cv::_InputArray *cmp, cv::_OutputArray *qualityMap, double maxPixelValue, MyCvScalar *returnValue) { BEGIN_WRAP cv::Scalar ret; if (qualityMap == nullptr) ret = cv::quality::QualityPSNR::compute(*ref, *cmp, cv::noArray(), maxPixelValue); else ret = cv::quality::QualityPSNR::compute(*ref, *cmp, *qualityMap, maxPixelValue); *returnValue = c(ret); END_WRAP }	cv::quality::QualityPSNR::compute()														
quality_QualityPSNR_getMaxPixelValue	CVAPI(ExceptionStatus) quality_QualityPSNR_getMaxPixelValue(cv::quality::QualityPSNR *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxPixelValue(); END_WRAP }	cv::quality::QualityPSNR::getMaxPixelValue()	classcv_1_1quality_1_1QualityPSNR.html	return the maximum pixel value used for PSNR computation			1									
quality_QualityPSNR_setMaxPixelValue	CVAPI(ExceptionStatus) quality_QualityPSNR_setMaxPixelValue(cv::quality::QualityPSNR *obj, double val) { BEGIN_WRAP obj->setMaxPixelValue(val); END_WRAP }	cv::quality::QualityPSNR::setMaxPixelValue()	classcv_1_1quality_1_1QualityPSNR.html	sets the maximum pixel value used for PSNR computation	val : Maximum pixel value		1									
quality_Ptr_QualityPSNR_get	CVAPI(ExceptionStatus) quality_Ptr_QualityPSNR_get(cv::Ptr<cv::quality::QualityPSNR>* ptr, cv::quality::QualityPSNR **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::quality::QualityPSNR	classcv_1_1quality_1_1QualityPSNR.html	Full reference peak signal to noise ratio (PSNR) algorithm https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.									•			
quality_createQualitySSIM	CVAPI(ExceptionStatus) quality_createQualitySSIM(cv::_InputArray* ref, cv::Ptr<cv::quality::QualitySSIM> **returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualitySSIM::create(*ref); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualitySSIM::create()	classcv_1_1quality_1_1QualitySSIM.html	Create an object which calculates quality.	ref : input image to use as the reference image for comparison		1									
quality_Ptr_QualitySSIM_delete	CVAPI(ExceptionStatus) quality_Ptr_QualitySSIM_delete(cv::Ptr<cv::quality::QualitySSIM>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::quality::QualitySSIM	classcv_1_1quality_1_1QualitySSIM.html	Full reference structural similarity algorithm https://en.wikipedia.org/wiki/Structural_similarity.								•				
quality_Ptr_QualitySSIM_get	CVAPI(ExceptionStatus) quality_Ptr_QualitySSIM_get(cv::Ptr<cv::quality::QualitySSIM>* ptr, cv::quality::QualitySSIM **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::quality::QualitySSIM	classcv_1_1quality_1_1QualitySSIM.html	Full reference structural similarity algorithm https://en.wikipedia.org/wiki/Structural_similarity.									•			
quality_QualitySSIM_staticCompute	CVAPI(ExceptionStatus) quality_QualitySSIM_staticCompute(cv::_InputArray* ref, cv::_InputArray* cmp, cv::_OutputArray* qualityMap, MyCvScalar *returnValue) { BEGIN_WRAP cv::Scalar ret; if (qualityMap == nullptr) ret = cv::quality::QualitySSIM::compute(*ref, *cmp, cv::noArray()); else ret = cv::quality::QualitySSIM::compute(*ref, *cmp, *qualityMap); *returnValue = c(ret); END_WRAP }	cv::quality::QualitySSIM::compute()														
quality_createQualityGMSD	CVAPI(ExceptionStatus) quality_createQualityGMSD(cv::_InputArray* ref, cv::Ptr<cv::quality::QualityGMSD> **returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualityGMSD::create(*ref); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualityGMSD::create()	classcv_1_1quality_1_1QualityGMSD.html	Create an object which calculates image quality.	ref : reference image		1									

quality_Ptr_QualityGMSD_delete	CVAPI(ExceptionStatus) quality_Ptr_QualityGMSD_delete(cv::Ptr<cv::quality::QualityGMSD>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::quality::QualityGMSD	classcv_1_1quality_1_1QualityGMSD.html	Full reference GMSD algorithm http://www4.comp.polyu.edu.hk/~cslzhang/IQA/GMSD/GMSD.htm.										•		
quality_Ptr_QualityGMSD_get	CVAPI(ExceptionStatus) quality_Ptr_QualityGMSD_get(cv::Ptr<cv::quality::QualityGMSD>* ptr, cv::quality::QualityGMSD **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::quality::QualityGMSD	classcv_1_1quality_1_1QualityGMSD.html	Full reference GMSD algorithm http://www4.comp.polyu.edu.hk/~cslzhang/IQA/GMSD/GMSD.htm.											•	
quality_QualityGMSD_staticCompute	CVAPI(ExceptionStatus) quality_QualityGMSD_staticCompute(cv::_InputArray* ref, cv::_InputArray* cmp, cv::_OutputArray* qualityMap, MyCvScalar *returnValue) { BEGIN_WRAP cv::Scalar ret; if (qualityMap == nullptr) ret = cv::quality::QualityGMSD::compute(*ref, *cmp, cv::noArray()); else ret = cv::quality::QualityGMSD::compute(*ref, *cmp, *qualityMap); *returnValue = c(ret); END_WRAP }	cv::quality::QualityGMSD::compute()														
quality_createQualityMSE	CVAPI(ExceptionStatus) quality_createQualityMSE(cv::_InputArray* ref, cv::Ptr<cv::quality::QualityMSE>* *returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualityMSE::create(*ref); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualityMSE::create()	classcv_1_1quality_1_1QualityMSE.html	Create an object which calculates quality.	ref : input image to use as the reference for comparison		1									
quality_Ptr_QualityMSE_delete	CVAPI(ExceptionStatus) quality_Ptr_QualityMSE_delete(cv::Ptr<cv::quality::QualityMSE>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::quality::QualityMSE	classcv_1_1quality_1_1QualityMSE.html	Full reference mean square error algorithm https://en.wikipedia.org/wiki/Mean_squared_error										•		
quality_Ptr_QualityMSE_get	CVAPI(ExceptionStatus) quality_Ptr_QualityMSE_get(cv::Ptr<cv::quality::QualityMSE>* ptr, cv::quality::QualityMSE **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::quality::QualityMSE	classcv_1_1quality_1_1QualityMSE.html	Full reference mean square error algorithm https://en.wikipedia.org/wiki/Mean_squared_error .											•	
quality_QualityMSE_staticCompute	CVAPI(ExceptionStatus) quality_QualityMSE_staticCompute(cv::_InputArray* ref, cv::_InputArray* cmp, cv::_OutputArray* qualityMap, MyCvScalar *returnValue) { BEGIN_WRAP cv::Scalar ret; if (qualityMap == nullptr) ret = cv::quality::QualityMSE::compute(*ref, *cmp, cv::noArray()); else ret = cv::quality::QualityMSE::compute(*ref, *cmp, *qualityMap); *returnValue = c(ret); END_WRAP }	cv::quality::QualityMSE::compute()														
quality_createQualityBRISQUE1	CVAPI(ExceptionStatus) quality_createQualityBRISQUE1(const char *modelFilePath, const char *rangeFilePath, cv::Ptr<cv::quality::QualityBRISQUE> **returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualityBRISQUE::create(modelFilePath, rangeFilePath); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualityBRISQUE::create()	classcv_1_1quality_1_1QualityBRISQUE.html	Create an object which calculates quality.	model_file_path : cv::String which contains a path to the BRISQUE model data, eg. /path/to/brisque_model_live.yml range_file_path : cv::String which contains a path to the BRISQUE range data, eg. /path/to/brisque_range_live.yml		2									
quality_createQualityBRISQUE2	CVAPI(ExceptionStatus) quality_createQualityBRISQUE2(cv::ml::SVM *model, cv::Mat *range, cv::Ptr<cv::quality::QualityBRISQUE> **returnValue) { BEGIN_WRAP const auto ptr = cv::quality::QualityBRISQUE::create(model, *range); *returnValue = clone(ptr); END_WRAP }	cv::quality::QualityBRISQUE::create()	classcv_1_1quality_1_1QualityBRISQUE.html	Create an object which calculates quality.	model_file_path : cv::String which contains a path to the BRISQUE model data, eg. /path/to/brisque_model_live.yml range_file_path : cv::String which contains a path to the BRISQUE range data, eg. /path/to/brisque_range_live.yml		2									
quality_Ptr_QualityBRISQUE_delete	CVAPI(ExceptionStatus) quality_Ptr_QualityBRISQUE_delete(cv::Ptr<cv::quality::QualityBRISQUE>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::quality::QualityBRISQUE	classcv_1_1quality_1_1QualityBRISQUE.html	BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) is a No Reference Image Quality Assessment (NR-IQA) algorithm. BRISQUE computes a score based on extracting Natural Scene Statistics (https://en.wikipedia.org/wiki/Scene_statistics) and calculating feature vectors. See Mittal et al. [174] for original paper and original implementation [173] . A trained model is provided in the /samples/ directory and is trained on the LIVE-R2 database [114] as in the original implementation. When evaluated against the TID2008 database [180] , the SROCC is -0.8424 versus the SROCC of -0.8354 in the original implementation. C++ code for the BRISQUE LIVE-R2 trainer and TID2008 evaluator are also provided in the /samples/ directory.										•	
quality_Ptr_QualityBRISQUE_get	CVAPI(ExceptionStatus) quality_Ptr_QualityBRISQUE_get(cv::Ptr<cv::quality::QualityBRISQUE>* ptr, cv::quality::QualityBRISQUE **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::quality::QualityBRISQUE	classcv_1_1quality_1_1QualityBRISQUE.html	BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator) is a No Reference Image Quality Assessment (NR-IQA) algorithm. BRISQUE computes a score based on extracting Natural Scene Statistics (https://en.wikipedia.org/wiki/Scene_statistics) and calculating feature vectors. See Mittal et al. [174] for original paper and original implementation [173] . A trained model is provided in the /samples/ directory and is trained on the LIVE-R2 database [114] as in the original implementation. When evaluated against the TID2008 database [180] , the SROCC is -0.8424 versus the SROCC of -0.8354 in the original implementation. C++ code for the BRISQUE LIVE-R2 trainer and TID2008 evaluator are also provided in the /samples/ directory.											•	
quality_QualityBRISQUE_staticCompute	CVAPI(ExceptionStatus) quality_QualityBRISQUE_staticCompute(cv::_InputArray* ref, const char* modelFilePath, const char* rangeFilePath, MyCvScalar *returnValue) { BEGIN_WRAP const auto ret = cv::quality::QualityBRISQUE::compute(*ref, modelFilePath, rangeFilePath); *returnValue = c(ret); END_WRAP }	cv::quality::QualityBRISQUE::compute()														
quality_QualityBRISQUE_computeFeatures	CVAPI(ExceptionStatus) quality_QualityBRISQUE_computeFeatures(cv::_InputArray* img, cv::_OutputArray *features) { BEGIN_WRAP cv::quality::QualityBRISQUE::computeFeatures(*img, *features); END_WRAP }	cv::quality::QualityBRISQUE::computeFeatures()	classcv_1_1quality_1_1QualityBRISQUE.html	static method for computing image features used by the BRISQUE algorithm	img : image (BGR(A) or grayscale) for which to compute features features : output row vector of features to cv::Mat or cv::UMat		2									

string_new1	CVAPI(std::string*) string_new1() { return new std::string; }	std::string						•				•				
string_new2	CVAPI(std::string*) string_new2(const char *str) { return new std::string(str); }	std::string						•				•				
string_delete	CVAPI(void) string_delete(std::string *s) { delete s; }	std::string						•					•			
string_c_str	CVAPI(const char*) string_c_str(std::string *s) { return s->c_str(); }	std::string::c_str()						•								
string_size	CVAPI(size_t) string_size(std::string *s) { return s->size(); }	std::string::size()						•								
vector_uchar_new1	CVAPI(std::vector<std::vector<uchar> >*) vector_vector_uchar_new1() { return new std::vector<std::vector<uchar> >; }	std::vector<std::vector<uchar>>						•				•				
vector_uchar_new2	CVAPI(std::vector<uchar>*) vector_uchar_new2(size_t size) { return new std::vector<uchar>(size); }	std::vector<uchar>()						•				•				
vector_uchar_new3	CVAPI(std::vector<uchar>*) vector_uchar_new3(uchar* data, size_t dataLength) { return new std::vector<uchar>(data, data + dataLength); }	new std::vector<uchar>()						•				•				
vector_uchar_getSize	CVAPI(size_t) vector_vector_uchar_getSize(std::vector<std::vector<uch ar> >* vec) { return vec->size(); }	std::vector<std::vector<uchar>>::size()						•								
vector_uchar_getPointer	CVAPI(uchar*) vector_uchar_getPointer(std:: vector<uchar>* vector) { return &(vector->at(0)); }	std::vector<uchar>						•						•		
vector_uchar_copy	CVAPI(void) vector_vector_uchar_copy(std ::vector<std::vector<uchar> >* vec, uchar** dst) { for (size_t i = 0; i < vec- >size(); i++) { auto& elem = vec- >at(i); void* src = &elem[0]; const auto length = sizeof(int) * elem.size(); memcpy(dst[i], src, length); } }	std::vector<std::vector<uchar>>						•								•
vector_uchar_delete	CVAPI(void) vector_vector_uchar_delete(st d::vector<std::vector<uchar > >* vec) { delete vec; }	std::vector<std::vector<uchar>>						•					•			
vector_int32_new1	CVAPI(std::vector<int>*) vector_int32_new1() { return new std::vector<int>; }	std::vector<int>						•				•				
vector_int32_new2	CVAPI(std::vector<int>*) vector_int32_new2(size_t size) { return new std::vector<int>(size); }	std::vector<int>						•				•				
vector_int32_new3	CVAPI(std::vector<int>*) vector_int32_new3(int* data, size_t dataLength) { return new std::vector<int>(data, data + dataLength); }	std::vector<int>						•				•				
vector_int32_getSize	CVAPI(size_t) vector_int32_getSize(std::vec tor<int>* vector) { return vector->size(); }	std::vector<int>::size()						•								
vector_int32_getPointer	CVAPI(int*) vector_int32_getPointer(std:: vector<int>* vector) { return &(vector->at(0)); }	std::vector<int>						•						•		
vector_int32_delete	CVAPI(void) vector_int32_delete(std::vect or<int>* vector) { delete vector; }	std::vector<int>						•					•			
vector_float_new1	CVAPI(std::vector<float>*) vector_float_new1() { return new std::vector<float>; }	std::vector<float>						•				•				
vector_float_new2	CVAPI(std::vector<float>*) vector_float_new2(size_t size) { return new std::vector<float>(size); }	std::vector<float>						•				•				
vector_float_new3	CVAPI(std::vector<float>*) vector_float_new3(float* data, size_t dataLength) { return new std::vector<float>(data, data + dataLength); }	std::vector<float>						•				•				
vector_float_getSize	CVAPI(size_t) vector_float_getSize(std::vect or<float>* vector) { return vector->size(); }	std::vector<float>::size()						•								
vector_float_getPointer	CVAPI(float*) vector_float_getPointer(std::v ector<float>* vector) { return &(vector->at(0)); }	std::vector<float>						•						•		
vector_float_delete	CVAPI(void) vector_float_delete(std::vecto r<float>* vector) { delete vector; }	std::vector<float>						•					•			
vector_double_new1	CVAPI(std::vector<std::vecto r<double> >*) vector_vector_double_new1() { return new std::vector<std::vector<doub le> >; }	std::vector<std::vector<double>>						•				•				
vector_double_new2	CVAPI(std::vector<double>*) vector_double_new2(size_t size) { return new std::vector<double>(size); }	std::vector<std::vector<double>>						•				•				
vector_double_new3	CVAPI(std::vector<double>*) vector_double_new3(double* data, size_t dataLength) { return new std::vector<double>(data, data + dataLength); }	std::vector<std::vector<double>>						•				•				

vector_double_getSize	CVAPI(size_t) vector_vector_double_getSize1(std::vector<std::vector<double> >* vec) { return vec->size(); }	std::vector<std::vector<double>>::size()						•									
vector_double_getPointer	CVAPI(double*) vector_double_getPointer(std::vector<double>* vector) { return &(vector->at(0)); }	std::vector<std::vector<double>>						•							•		
vector_double_delete	CVAPI(void) vector_vector_double_delete(std::vector<std::vector<double> >* vec) { delete vec; }	std::vector<std::vector<double>>						•						•			
vector_Vec2f_new1	CVAPI(std::vector<cv::Vec2f>*) vector_Vec2f_new1() { return new std::vector<cv::Vec2f>; }	std::vector<cv::Vec2f>						•					•				
vector_Vec2f_getSize	CVAPI(size_t) vector_Vec2f_getSize(std::vector<cv::Vec2f>* vector) { return vector->size(); }	std::vector<cv::Vec2f>::size()						•									
vector_Vec2f_getPointer	CVAPI(cv::Vec2f*) vector_Vec2f_getPointer(std::vector<cv::Vec2f>* vector) { return &(vector->at(0)); }	std::vector<cv::Vec2f>						•							•		
vector_Vec2f_delete	CVAPI(void) vector_Vec2f_delete(std::vector<cv::Vec2f>* vector) { delete vector; }	std::vector<cv::Vec2f>						•						•			
vector_Vec3f_new1	CVAPI(std::vector<cv::Vec3f>*) vector_Vec3f_new1() { return new std::vector<cv::Vec3f>; }	std::vector<cv::Vec3f>						•					•				
vector_Vec3f_getSize	CVAPI(size_t) vector_Vec3f_getSize(std::vector<cv::Vec3f>* vector) { return vector->size(); }	std::vector<cv::Vec3f>::size()						•									
vector_Vec3f_getPointer	CVAPI(cv::Vec3f*) vector_Vec3f_getPointer(std::vector<cv::Vec3f>* vector) { return &(vector->at(0)); }	std::vector<cv::Vec3f>						•							•		
vector_Vec3f_delete	CVAPI(void) vector_Vec3f_delete(std::vector<cv::Vec3f>* vector) { delete vector; }	std::vector<cv::Vec3f>						•						•			
vector_Vec4f_new1	CVAPI(std::vector<cv::Vec4f>*) vector_Vec4f_new1() { return new std::vector<cv::Vec4f>; }	std::vector<cv::Vec4f>						•					•				
vector_Vec4f_new3	CVAPI(std::vector<cv::Vec4f>*) vector_Vec4f_new3(cv::Vec4f* data, size_t dataLength) { return new std::vector<cv::Vec4f>(data, data + dataLength); }	std::vector<cv::Vec4f>						•					•				
vector_Vec4f_getSize	CVAPI(size_t) vector_Vec4f_getSize(std::vector<cv::Vec4f>* vector) { return vector->size(); }	std::vector<cv::Vec4f>::size()						•									
vector_Vec4f_getPointer	CVAPI(cv::Vec4f*) vector_Vec4f_getPointer(std::vector<cv::Vec4f>* vector) { return &(vector->at(0)); }	std::vector<cv::Vec4f>						•							•		
vector_Vec4f_delete	CVAPI(void) vector_Vec4f_delete(std::vector<cv::Vec4f>* vector) { delete vector; }	std::vector<cv::Vec4f>						•						•			
vector_Vec4i_new1	CVAPI(std::vector<cv::Vec4i>*) vector_Vec4i_new1() { return new std::vector<cv::Vec4i>; }	std::vector<cv::Vec4i>						•					•				
vector_Vec4i_new3	CVAPI(std::vector<cv::Vec4i>*) vector_Vec4i_new3(cv::Vec4i* data, size_t dataLength) { return new std::vector<cv::Vec4i>(data, data + dataLength); }	std::vector<cv::Vec4i>						•					•				
vector_Vec4i_getSize	CVAPI(size_t) vector_Vec4i_getSize(std::vector<cv::Vec4i>* vector) { return vector->size(); }	std::vector<cv::Vec4i>::size()						•									
vector_Vec4i_getPointer	CVAPI(cv::Vec4i*) vector_Vec4i_getPointer(std::vector<cv::Vec4i>* vector) { return &(vector->at(0)); }	std::vector<cv::Vec4i>						•							•		
vector_Vec4i_delete	CVAPI(void) vector_Vec4i_delete(std::vector<cv::Vec4i>* vector) { delete vector; }	std::vector<cv::Vec4i>						•						•			
vector_Vec6f_new1	CVAPI(std::vector<cv::Vec6f>*) vector_Vec6f_new1() { return new std::vector<cv::Vec6f>; }	std::vector<cv::Vec6f>						•					•				
vector_Vec6f_getSize	CVAPI(size_t) vector_Vec6f_getSize(std::vector<cv::Vec6f>* vector) { return vector->size(); }	std::vector<cv::Vec6f>::size()						•									
vector_Vec6f_getPointer	CVAPI(cv::Vec6f*) vector_Vec6f_getPointer(std::vector<cv::Vec6f>* vector) { return &(vector->at(0)); }	std::vector<cv::Vec6f>						•							•		
vector_Vec6f_delete	CVAPI(void) vector_Vec6f_delete(std::vector<cv::Vec6f>* vector) { delete vector; }	std::vector<cv::Vec6f>						•						•			
vector_Point2i_new1	CVAPI(std::vector<cv::Point>*) vector_Point2i_new1() { return new std::vector<cv::Point>; }	std::vector<cv::Point>						•					•				
vector_Point2i_new2	CVAPI(std::vector<cv::Point>*) vector_Point2i_new2(size_t size) { return new std::vector<cv::Point>(size); }	std::vector<cv::Point>						•					•				
vector_Point2i_new3	CVAPI(std::vector<cv::Point>*) vector_Point2i_new3(cv::Point* data, size_t dataLength) { return new std::vector<cv::Point>(data, data + dataLength); }	std::vector<cv::Point>						•					•				
vector_Point2i_getSize	CVAPI(size_t) vector_Point2i_getSize(std::vector<cv::Point>* vector) { return vector->size(); }	std::vector<cv::Point>::size()						•									

vector_Point2i_getPointer	CVAPI(cv::Point*) vector_Point2i_getPointer(std::vector<cv::Point>* vector) { return &(vector->at(0)); }	std::vector<cv::Point>						•						•		
vector_Point2i_delete	CVAPI(void) vector_Point2i_delete(std::vector<cv::Point>* vector) { delete vector; }	std::vector<cv::Point>						•					•			
vector_Point2f_new1	CVAPI(std::vector<cv::Point2f>*) vector_Point2f_new1() { return new std::vector<cv::Point2f>; }	std::vector<cv::Point2f>						•				•				
vector_Point2f_new2	CVAPI(std::vector<cv::Point2f>*) vector_Point2f_new2(size_t size) { return new std::vector<cv::Point2f>(size); }	std::vector<cv::Point2f>						•				•				
vector_Point2f_new3	CVAPI(std::vector<cv::Point2f>*) vector_Point2f_new3(cv::Point2f* data, size_t dataLength) { return new std::vector<cv::Point2f>(data, data + dataLength); }	std::vector<cv::Point2f>						•				•				
vector_Point2f_getSize	CVAPI(size_t) vector_Point2f_getSize(std::vector<cv::Point2f>* vector) { return vector->size(); }	std::vector<cv::Point2f>::size()						•								
vector_Point2f_getPointer	CVAPI(cv::Point2f*) vector_Point2f_getPointer(std::vector<cv::Point2f>* vector) { return &(vector->at(0)); }	std::vector<cv::Point2f>						•						•		
vector_Point2f_delete	CVAPI(void) vector_Point2f_delete(std::vector<cv::Point2f>* vector) { delete vector; }	std::vector<cv::Point2f>						•					•			
vector_Point2d_new1	CVAPI(std::vector<cv::Point2d>*) vector_Point2d_new1() { return new std::vector<cv::Point2d>; }	std::vector<cv::Point2d>						•				•				
vector_Point2d_getSize	CVAPI(size_t) vector_Point2d_getSize(std::vector<cv::Point2d>* vector) { return vector->size(); }	std::vector<cv::Point2d>::size()						•								
vector_Point2d_getPointer	CVAPI(cv::Point2d*) vector_Point2d_getPointer(std::vector<cv::Point2d>* vector) { return &(vector->at(0)); }	std::vector<cv::Point2d>						•						•		
vector_Point2d_delete	CVAPI(void) vector_Point2d_delete(std::vector<cv::Point2d>* vector) { delete vector; }	std::vector<cv::Point2d>						•					•			
vector_Point3f_new1	CVAPI(std::vector<cv::Point3f>*) vector_Point3f_new1() { return new std::vector<cv::Point3f>; }	std::vector<cv::Point3f>						•				•				
vector_Point3f_new2	CVAPI(std::vector<cv::Point3f>*) vector_Point3f_new2(size_t size) { return new std::vector<cv::Point3f>(size); }	std::vector<cv::Point3f>						•				•				
vector_Point3f_new3	CVAPI(std::vector<cv::Point3f>*) vector_Point3f_new3(cv::Point3f* data, size_t dataLength) { return new std::vector<cv::Point3f>(data, data + dataLength); }	std::vector<cv::Point3f>						•				•				
vector_Point3f_getSize	CVAPI(size_t) vector_Point3f_getSize(std::vector<cv::Point3f>* vector) { return vector->size(); }	std::vector<cv::Point3f>::size()						•								
vector_Point3f_getPointer	CVAPI(cv::Point3f*) vector_Point3f_getPointer(std::vector<cv::Point3f>* vector) { return &(vector->at(0)); }	std::vector<cv::Point3f>						•						•		
vector_Point3f_delete	CVAPI(void) vector_Point3f_delete(std::vector<cv::Point3f>* vector) { delete vector; }	std::vector<cv::Point3f>						•					•			
vector_Rect_new1	CVAPI(std::vector<cv::Rect>*) vector_Rect_new1() { return new std::vector<cv::Rect>; }	new std::vector<cv::Rect>						•				•				
vector_Rect_new2	CVAPI(std::vector<cv::Rect>*) vector_Rect_new2(size_t size) { return new std::vector<cv::Rect>(size); }	new std::vector<cv::Rect>						•				•				
vector_Rect_new3	CVAPI(std::vector<cv::Rect>*) vector_Rect_new3(cv::Rect* data, size_t dataLength) { return new std::vector<cv::Rect>(data, data + dataLength); }	new std::vector<cv::Rect>						•				•				
vector_Rect_getSize	CVAPI(size_t) vector_Rect_getSize(std::vector<cv::Rect>* vector) { return vector->size(); }	new std::vector<cv::Rect>::size()						•								
vector_Rect_getPointer	CVAPI(cv::Rect*) vector_Rect_getPointer(std::vector<cv::Rect> *vector) { return &(vector->at(0)); }	new std::vector<cv::Rect>						•						•		
vector_Rect_delete	CVAPI(void) vector_Rect_delete(std::vector<cv::Rect> *vector) { //vector->~vector(); delete vector; }	new std::vector<cv::Rect>						•					•			
vector_Rect2d_new1	CVAPI(std::vector<cv::Rect2d>*) vector_Rect2d_new1() { return new std::vector<cv::Rect2d>; }	std::vector<cv::Rect2d>						•				•				
vector_Rect2d_new2	CVAPI(std::vector<cv::Rect2d>*) vector_Rect2d_new2(size_t size) { return new std::vector<cv::Rect2d>(size); }	std::vector<cv::Rect2d>						•				•				

vector_Rect2d_new3	CVAPI(std::vector<cv::Rect2d>*) vector_Rect2d_new3(cv::Rect2d* data, size_t dataLength) { return new std::vector<cv::Rect2d>(data, data + dataLength); }	std::vector<cv::Rect2d>						•				•				
vector_Rect2d_getSize	CVAPI(size_t) vector_Rect2d_getSize(std::vector<cv::Rect2d>* vector) { return vector->size(); }	std::vector<cv::Rect2d>::size()						•								
vector_Rect2d_getPointer	CVAPI(cv::Rect2d*) vector_Rect2d_getPointer(std::vector<cv::Rect2d>* vector) { return &(vector->at(0)); }	std::vector<cv::Rect2d>						•						•		
vector_Rect2d_delete	CVAPI(void) vector_Rect2d_delete(std::vector<cv::Rect2d> *vector) { delete vector; }	std::vector<cv::Rect2d>						•					•			
vector_RotatedRect_new1	CVAPI(std::vector<cv::RotatedRect>*) vector_RotatedRect_new1() { return new std::vector<cv::RotatedRect>; }	std::vector<cv::RotatedRect>						•				•				
vector_RotatedRect_new2	CVAPI(std::vector<cv::RotatedRect>*) vector_RotatedRect_new2(size_t size) { return new std::vector<cv::RotatedRect>(size); }	std::vector<cv::RotatedRect>						•				•				
vector_RotatedRect_new3	CVAPI(std::vector<cv::RotatedRect>*) vector_RotatedRect_new3(cv::RotatedRect* data, size_t dataLength) { return new std::vector<cv::RotatedRect>(data, data + dataLength); }	std::vector<cv::RotatedRect>						•				•				
vector_RotatedRect_getSize	CVAPI(size_t) vector_RotatedRect_getSize(std::vector<cv::RotatedRect>* vector) { return vector->size(); }	std::vector<cv::RotatedRect>::size()						•								
vector_RotatedRect_getPointer	CVAPI(cv::RotatedRect*) vector_RotatedRect_getPointer(std::vector<cv::RotatedRect>* vector) { return &(vector->at(0)); }	std::vector<cv::RotatedRect>						•						•		
vector_RotatedRect_delete	CVAPI(void) vector_RotatedRect_delete(std::vector<cv::RotatedRect> *vector) { delete vector; }	std::vector<cv::RotatedRect>						•					•			
vector_KeyPoint_new1	CVAPI(std::vector<cv::KeyPoint>*) vector_KeyPoint_new1() { return new std::vector<cv::KeyPoint>; }	std::vector<cv::KeyPoint>						•				•				
vector_KeyPoint_new2	CVAPI(std::vector<cv::KeyPoint>*) vector_KeyPoint_new2(size_t size) { return new std::vector<cv::KeyPoint>(size); }	std::vector<cv::KeyPoint>						•				•				
vector_KeyPoint_new3	CVAPI(std::vector<cv::KeyPoint>*) vector_KeyPoint_new3(cv::KeyPoint *data, size_t dataLength) { return new std::vector<cv::KeyPoint>(data, data + dataLength); }	std::vector<cv::KeyPoint>						•				•				
vector_KeyPoint_getSize	CVAPI(size_t) vector_KeyPoint_getSize(std::vector<cv::KeyPoint>* vector) { return vector->size(); }	std::vector<cv::KeyPoint>::size()						•								
vector_KeyPoint_getPointer	CVAPI(cv::KeyPoint*) vector_KeyPoint_getPointer(std::vector<cv::KeyPoint>* vector) { return &(vector->at(0)); }	std::vector<cv::KeyPoint>						•						•		
vector_KeyPoint_delete	CVAPI(void) vector_KeyPoint_delete(std::vector<cv::KeyPoint>* vector) { //vector->~vector(); delete vector; }	std::vector<cv::KeyPoint>						•					•			
vector_DMatch_new1	CVAPI(std::vector<cv::DMatch>*) vector_DMatch_new1() { return new std::vector<cv::DMatch>; }	std::vector<cv::DMatch>						•				•				
vector_DMatch_new2	CVAPI(std::vector<cv::DMatch>*) vector_DMatch_new2(size_t size) { return new std::vector<cv::DMatch>(size); }	std::vector<cv::DMatch>						•				•				
vector_DMatch_new3	CVAPI(std::vector<cv::DMatch>*) vector_DMatch_new3(cv::DMatch *data, size_t dataLength) { return new std::vector<cv::DMatch>(data, data + dataLength); }	std::vector<cv::DMatch>						•				•				
vector_DMatch_getSize	CVAPI(size_t) vector_DMatch_getSize(std::vector<cv::DMatch>* vector) { return vector->size(); }	std::vector<cv::DMatch>::size()						•								
vector_DMatch_getPointer	CVAPI(cv::DMatch*) vector_DMatch_getPointer(std::vector<cv::DMatch>* vector) { return &(vector->at(0)); }	std::vector<cv::DMatch>						•						•		
vector_DMatch_delete	CVAPI(void) vector_DMatch_delete(std::vector<cv::DMatch>* vector) { delete vector; }	std::vector<cv::DMatch>						•					•			
vector_Mat_new1	CVAPI(std::vector<cv::Mat>*) vector_Mat_new1() { return new std::vector<cv::Mat>; }	std::vector<cv::Mat>						•				•				
vector_Mat_new2	CVAPI(std::vector<cv::Mat>*) vector_Mat_new2(uint32_t size) { return new std::vector<cv::Mat>(size); }	std::vector<cv::Mat>						•				•				

vector_Mat_new3	CVAPI(std::vector<cv::Mat>*) vector_Mat_new3(cv::Mat **data, uint32_t dataLength) { auto *vec = new std::vector<cv::Mat>(dataLe ngth); for (size_t i = 0; i < dataLength; i++) { (*vec)[i] = *(data[i]); } return vec; }	std::vector<cv::Mat>							•					•			
vector_Mat_getSize	CVAPI(size_t) vector_Mat_getSize(std::vect or<cv::Mat>* vector) { return vector->size(); }	std::vector<cv::Mat>::size()							•								
vector_Mat_getPointer	CVAPI(cv::Mat*) vector_Mat_getPointer(std::v ector<cv::Mat>* vector) { return &(vector->at(0)); }	std::vector<cv::Mat>							•							•	
vector_Mat_delete	CVAPI(void) vector_Mat_delete(std::vecto r<cv::Mat>* vector) { //vector->~vector(); delete vector; }	std::vector<cv::Mat>							•						•		
vector_Mat_assignToArray	CVAPI(void) vector_Mat_assignToArray(std ::vector<cv::Mat>* vector, cv::Mat** arr) { for (size_t i = 0; i < vector->size(); i++) { (vector- >at(i)).assignTo(*(arr[i])); } }	std::vector<cv::Mat>							•	•							
vector_DTrees_Node_new1	CVAPI(std::vector<cv::ml::D Trees::Node>*) vector_DTrees_Node_new1() { return new std::vector<cv::ml::DTrees:: Node>; }	std::vector<cv::ml::DTrees::Node>							•					•			
vector_DTrees_Node_getSize	CVAPI(size_t) vector_DTrees_Node_getSize(std::vector<cv::ml::DTrees:: Node> *vector) { return vector->size(); }	std::vector<cv::ml::DTrees::Node>::size()							•								
vector_DTrees_Node_getPointer	CVAPI(cv::ml::DTrees::Node *) vector_DTrees_Node_getPoint er(std::vector<cv::ml::DTree s::Node> *vector) { return &(vector->at(0)); }	std::vector<cv::ml::DTrees::Node>							•							•	
vector_DTrees_Node_delete	CVAPI(void) vector_DTrees_Node_delete(s td::vector<cv::ml::DTrees::N ode> *vector) { delete vector; }	std::vector<cv::ml::DTrees::Node>							•						•		
vector_DTrees_Split_new1	CVAPI(std::vector<cv::ml::D Trees::Split>*) vector_DTrees_Split_new1() { return new std::vector<cv::ml::DTrees:: Split>; }	std::vector<cv::ml::DTrees::Split>							•					•			
vector_DTrees_Split_getSize	CVAPI(size_t) vector_DTrees_Split_getSize(std::vector<cv::ml::DTrees:: Split> *vector) { return vector->size(); }	std::vector<cv::ml::DTrees::Split>::size()							•								
vector_DTrees_Split_getPointer	CVAPI(cv::ml::DTrees::Split*) vector_DTrees_Split_getPoint er(std::vector<cv::ml::DTree s::Split> *vector) { return &(vector->at(0)); }	std::vector<cv::ml::DTrees::Split>							•							•	
vector_DTrees_Split_delete	CVAPI(void) vector_DTrees_Split_delete(st d::vector<cv::ml::DTrees::S plit> *vector) { delete vector; }	std::vector<cv::ml::DTrees::Split>							•						•		
vector_ImageFeatures_new1	CVAPI(std::vector<cv::detail: :ImageFeatures>*) vector_ImageFeatures_new1( ) { return new std::vector<cv::detail::Image Features>; }	std::vector<cv::detail::ImageFeatures>							•					•			
vector_ImageFeatures_getSize	CVAPI(size_t) vector_ImageFeatures_getSiz e(std::vector<cv::detail::Image Features>* vector) { return vector->size(); }	std::vector<cv::detail::ImageFeatures>							•								
vector_ImageFeatures_getKeypointsSize	CVAPI(void) vector_ImageFeatures_getKe ypointsSize(std::vector<cv::detail::Image Features>* vector, size_t *dst) { for (size_t i = 0; i < vector->size(); i++) { dst[i] = vector- >at(i).keypoints.size(); } }	std::vector<cv::detail::ImageFeatures>							•	•							
vector_ImageFeatures_getElements	CVAPI(void) vector_ImageFeatures_getEle ments(std::vector<cv::detail::Image Features>* vector, detail_ImageFeatures* dstArray) { for (size_t i = 0; i < vector->size(); i++) { const auto &src = vector->at(i); auto &dst = dstArray[i]; dst.img_idx = src.img_idx; dst.img_size = c(src.img_size); //std::memcpy(dst.keypoints, &src.keypoints[0], sizeof(cv::KeyPoint)*src.keyp oints.size()); std::copy(src.keypoints.begin(	std::vector<cv::detail::ImageFeatures>							•	•							
vector_ImageFeatures_delete	CVAPI(void) vector_ImageFeatures_delete (std::vector<cv::detail::Imag eFeatures>* vector) { delete vector; }	std::vector<cv::detail::ImageFeatures>							•						•		
vector_vector_uchar_new1	CVAPI(std::vector<std::vecto r<uchar> >*) vector_vector_uchar_new1() { return new std::vector<std::vector<ucha r> >;	std::vector<std::vector<uchar>>							•					•			
vector_vector_uchar_getSize1	CVAPI(size_t) vector_vector_uchar_getSize1 (std::vector<std::vector<uch ar> >* vec) { return vec->size(); }	std::vector<std::vector<uchar>>							•								

vector_vector_uchar_getSize2	CVAPI(void) vector_vector_uchar_getSize2(std::vector<std::vector<uchar> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<uchar>>						•	•							
vector_vector_uchar_copy	CVAPI(void) vector_vector_uchar_copy(std::vector<std::vector<uchar> >* vec, uchar** dst) { for (size_t i = 0; i < vec->size(); i++) { auto& elem = vec->at(i); void* src = &elem[0]; const auto length = sizeof(int) * elem.size(); memcpy(dst[i], src, length); } }	std::vector<std::vector<uchar>>						•								•
vector_vector_uchar_delete	CVAPI(void) vector_vector_uchar_delete(std::vector<std::vector<uchar> >* vec) { delete vec; }	std::vector<std::vector<uchar>>						•						•		
vector_vector_int_new1	CVAPI(std::vector<std::vector<int> >*) vector_vector_int_new1() { return new std::vector<std::vector<int> >; }	std::vector<std::vector<int>>						•					•			
vector_vector_int_getSize1	CVAPI(size_t) vector_vector_int_getSize1(std::vector<std::vector<int> >* vec) { return vec->size(); }	std::vector<std::vector<int>>::size()						•								
vector_vector_int_getSize2	CVAPI(void) vector_vector_int_getSize2(std::vector<std::vector<int> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<int>>						•	•							
vector_vector_int_copy	CVAPI(void) vector_vector_int_copy(std::vector<std::vector<int> >* vec, int** dst) { for (size_t i = 0; i < vec->size(); i++) { auto& elem = vec->at(i); void* src = &elem[0]; const auto length = sizeof(int) * elem.size(); memcpy(dst[i], src, length); } }	std::vector<std::vector<int>>						•								•
vector_vector_int_delete	CVAPI(void) vector_vector_int_delete(std::vector<std::vector<int> >* vec) { delete vec; }	std::vector<std::vector<int>>						•						•		
vector_vector_double_new1	CVAPI(std::vector<std::vector<double> >*) vector_vector_double_new1() { return new std::vector<std::vector<double> >; }	std::vector<std::vector<double>>						•					•			
vector_vector_double_getSize1	CVAPI(size_t) vector_vector_double_getSize1(std::vector<std::vector<double> >* vec) { return vec->size(); }	std::vector<std::vector<double>>::size()						•								
vector_vector_double_getSize2	CVAPI(void) vector_vector_double_getSize2(std::vector<std::vector<double> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<double>>						•	•							
vector_vector_double_copy	CVAPI(void) vector_vector_double_copy(std::vector<std::vector<double> >* vec, double** dst) { for (size_t i = 0; i < vec->size(); i++) { std::vector<double>& elem = vec->at(i); void* src = &elem[0]; const size_t length = sizeof(double) * elem.size(); memcpy(dst[i], src, length); } }	std::vector<std::vector<double>>						•								•
vector_vector_double_delete	CVAPI(void) vector_vector_double_delete(std::vector<std::vector<double> >* vec) { delete vec; }	std::vector<std::vector<double>>						•						•		
vector_vector_KeyPoint_new1	CVAPI(std::vector<std::vector<cv::KeyPoint> >*) vector_vector_KeyPoint_new1() { return new std::vector<std::vector<cv::KeyPoint> >; }	std::vector<std::vector<cv::KeyPoint>>						•					•			
vector_vector_KeyPoint_new3	CVAPI(std::vector<std::vector<cv::KeyPoint> >*) vector_vector_KeyPoint_new3(cv::KeyPoint** values, int size1, int* size2) { std::vector<std::vector<cv::KeyPoint> >* vec = new std::vector<std::vector<cv::KeyPoint> >(size1); for (int i = 0; i < size1; i++) { vec->at(i) = std::vector<cv::KeyPoint>(values[i], values[i] + size2[i]); } return vec; }	std::vector<std::vector<cv::KeyPoint>>						•					•			
vector_vector_KeyPoint_getSize1	CVAPI(size_t) vector_vector_KeyPoint_getSize1(std::vector<std::vector<cv::KeyPoint> >* vec) { return vec->size(); }	std::vector<std::vector<cv::KeyPoint>>::size()						•								
vector_vector_KeyPoint_getSize2	CVAPI(void) vector_vector_KeyPoint_getSize2(std::vector<std::vector<cv::KeyPoint> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<cv::KeyPoint>>						•	•							

vector_vector_KeyPoint_copy	CVAPI(void) vector_vector_KeyPoint_copy(std::vector<std::vector<cv::KeyPoint> >* vec, cv::KeyPoint** dst) { copyFromVectorToArray(vec, dst); }	std::vector<std::vector<cv::KeyPoint>>						•								•
vector_vector_KeyPoint_delete	CVAPI(void) vector_vector_KeyPoint_delete(std::vector<std::vector<cv::KeyPoint> >* vec) { delete vec; }	std::vector<std::vector<cv::KeyPoint>>						•						•		
vector_vector_DMatch_new1	CVAPI(std::vector<std::vector<cv::DMatch> >*) vector_vector_DMatch_new1( ) { return new std::vector<std::vector<cv::DMatch> >; }	std::vector<std::vector<cv::DMatch>>						•					•			
vector_vector_DMatch_getSize1	CVAPI(size_t) vector_vector_DMatch_getSize1(std::vector<std::vector<cv::DMatch> >* vec) { return vec->size(); }	std::vector<std::vector<cv::DMatch>>::size()						•								
vector_vector_DMatch_getSize2	CVAPI(void) vector_vector_DMatch_getSize2(std::vector<std::vector<cv::DMatch> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<cv::DMatch>>						•	•							
vector_vector_DMatch_copy	CVAPI(void) vector_vector_DMatch_copy(std::vector<std::vector<cv::DMatch> >* vec, cv::DMatch** dst) { copyFromVectorToArray(vec, dst); }	std::vector<std::vector<cv::DMatch>>						•								•
vector_vector_DMatch_delete	CVAPI(void) vector_vector_DMatch_delete(std::vector<std::vector<cv::DMatch> >* vec) { delete vec; }	std::vector<std::vector<cv::DMatch>>						•						•		
vector_vector_Point_new1	CVAPI(std::vector<std::vector<cv::Point> >*) vector_vector_Point_new1() { return new std::vector<std::vector<cv::Point> >; }	std::vector<std::vector<cv::Point>>						•					•			
vector_vector_Point_new2	CVAPI(std::vector<std::vector<cv::Point> >*) vector_vector_Point_new2(size_t size) { return new std::vector<std::vector<cv::Point> >(size); }	std::vector<std::vector<cv::Point>>						•					•			
vector_vector_Point_getSize1	CVAPI(size_t) vector_vector_Point_getSize1(std::vector<std::vector<cv::Point> >* vec) { return vec->size(); }	std::vector<std::vector<cv::Point>>::size()						•								
vector_vector_Point_getSize2	CVAPI(void) vector_vector_Point_getSize2(std::vector<std::vector<cv::Point> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<cv::Point>>						•	•							
vector_vector_Point_copy	CVAPI(void) vector_vector_Point_copy(std::vector<std::vector<cv::Point> >* vec, cv::Point** dst) { copyFromVectorToArray(vec, dst); }	std::vector<std::vector<cv::Point>>						•								•
vector_vector_Point_delete	CVAPI(void) vector_vector_Point_delete(std::vector<std::vector<cv::Point> >* vec) { delete vec; }	std::vector<std::vector<cv::Point>>						•						•		
vector_vector_Point2f_new1	CVAPI(std::vector<std::vector<cv::Point2f> >*) vector_vector_Point2f_new1() { return new std::vector<std::vector<cv::Point2f> >; }	std::vector<std::vector<cv::Point2f>>						•					•			
vector_vector_Point2f_getSize1	CVAPI(size_t) vector_vector_Point2f_getSize1(std::vector<std::vector<cv::Point2f> >* vec) { return vec->size(); }	std::vector<std::vector<cv::Point2f>>::size()						•								
vector_vector_Point2f_getSize2	CVAPI(void) vector_vector_Point2f_getSize2(std::vector<std::vector<cv::Point2f> >* vec, size_t* sizes) { for (size_t i = 0; i < vec->size(); i++) { sizes[i] = vec->at(i).size(); } }	std::vector<std::vector<cv::Point2f>>						•	•							
vector_vector_Point2f_copy	CVAPI(void) vector_vector_Point2f_copy(std::vector<std::vector<cv::Point2f> >* vec, cv::Point2f** dst) { copyFromVectorToArray(vec, dst); }	std::vector<std::vector<cv::Point2f>>						•								•
vector_vector_Point2f_delete	CVAPI(void) vector_vector_Point2f_delete(std::vector<std::vector<cv::Point2f> >* vec) { delete vec; }	std::vector<std::vector<cv::Point2f>>						•						•		
vector_string_new1	CVAPI(std::vector<std::string>*) vector_string_new1() { return new std::vector<std::string>; }	std::vector<std::string>						•					•			
vector_string_new2	CVAPI(std::vector<std::string>*) vector_string_new2(size_t size) { return new std::vector<std::string>(size); }	std::vector<std::string>						•					•			
vector_string_getSize	CVAPI(size_t) vector_string_getSize(std::vector<std::string>* vec) { return vec->size(); }	std::vector<std::string>::size()						•								

vector_string_getElements	CVAPI(void) vector_string_getElements(std::vector<std::string>* vector, const char** cStringPointers, int32_t* stringLengths) { for (size_t i = 0; i < vector->size(); i++) { const auto& elem = vector->at(i); cStringPointers[i] = elem.c_str(); stringLengths[i] = static_cast<int32_t>(elem.size()); } }	std::vector<std::string>						•	•						
vector_string_delete	CVAPI(void) vector_string_delete(std::vector<std::string>* vector) { delete vector; }	std::vector<std::string>						•					•			
videoio_VideoCapture_new1	CVAPI(ExceptionStatus) videoio_VideoCapture_new1(cv::VideoCapture **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoCapture; END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.								•			
videoio_VideoCapture_new2	CVAPI(ExceptionStatus) videoio_VideoCapture_new2(const char *filename, int apiPreference, cv::VideoCapture **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoCapture(filename, apiPreference); END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.								•			
videoio_VideoCapture_new3	CVAPI(ExceptionStatus) videoio_VideoCapture_new3(int device, int apiPreference, cv::VideoCapture **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoCapture(device, apiPreference); END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.								•			
videoio_VideoCapture_new4	CVAPI(ExceptionStatus) videoio_VideoCapture_new4(const char* filename, int apiPreference, int* params, int paramsLength, cv::VideoCapture** returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); * returnValue = new cv::VideoCapture(filename, apiPreference, paramsVec); END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.								•			
videoio_VideoCapture_new5	CVAPI(ExceptionStatus) videoio_VideoCapture_new5(int device, int apiPreference, int* params, int paramsLength, cv::VideoCapture** returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); * returnValue = new cv::VideoCapture(device, apiPreference, paramsVec); END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.								•			
videoio_VideoCapture_delete	CVAPI(ExceptionStatus) videoio_VideoCapture_delete(cv::VideoCapture *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences. Here is how the class can be used: #include <opencv2/core.hpp>#include <opencv2/videoio.hpp>#include <opencv2/highgui.hpp>#include <iostream>#include <stdio.h>using namespace cv;using namespace std;int main(int, char**){ Mat frame; //--- INITIALIZE VIDEOCAPTURE VideoCapture cap; // open the default camera using default API // cap.open(0); // OR advance usage: select any API backend int deviceID = 0; // 0 = open default camera int apiID = cv::CAP_ANY; // 0 = autodetect default API // open selected camera using selected API cap.open(deviceID, apiID); // check if we succeeded if (!cap.isOpened()) { cerr << "ERROR! Unable to open camera\n"; return -1; } //--- GRAB AND WRITE LOOP cout << "Start grabbing" << endl << "Press any key to terminate" << endl; for (;;) { // wait for a new frame from camera and store it into 'frame' cap.read(frame); // check if we succeeded if (frame.empty()) {								•		
videoio_VideoCapture_open1	CVAPI(ExceptionStatus) videoio_VideoCapture_open1(cv::VideoCapture *obj, const char *filename, int apiPreference, int *returnValue) { BEGIN_WRAP *returnValue = obj->open(filename, apiPreference) ? 1 : 0; END_WRAP }	cv::VideoCapture::open()	classcv_1_1VideoCapture.html	Opens a video file or a capturing device or an IP video stream for video capturing.		This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.Parameters are same as the constructor VideoCapture(const String& filename, int apiPreference = CAP_ANY)The method first calls VideoCapture::release to close the already opened file or camera.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/facedetect.cpp, samples/cpp/laplace.cpp, samples/cpp/lkdemo.cpp, samples/cpp/peopledetect.cpp, samples/cpp/polar_transforms.cpp, samples/cpp/segment_objects.cpp, samples/cpp/train_HOG.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, samples/dnn/segmentation.cpp, samples/dnn/text_detection.cpp, and	2							
videoio_VideoCapture_open2	CVAPI(ExceptionStatus) videoio_VideoCapture_open2(cv::VideoCapture *obj, int device, int apiPreference, int *returnValue) { BEGIN_WRAP *returnValue = obj->open(device, apiPreference) ? 1 : 0; END_WRAP }	cv::VideoCapture::open()	classcv_1_1VideoCapture.html	Opens a video file or a capturing device or an IP video stream for video capturing.		This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.Parameters are same as the constructor VideoCapture(const String& filename, int apiPreference = CAP_ANY)The method first calls VideoCapture::release to close the already opened file or camera.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/facedetect.cpp, samples/cpp/laplace.cpp, samples/cpp/lkdemo.cpp, samples/cpp/peopledetect.cpp, samples/cpp/polar_transforms.cpp, samples/cpp/segment_objects.cpp, samples/cpp/train_HOG.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, samples/dnn/segmentation.cpp, samples/dnn/text_detection.cpp, and	2							
videoio_VideoCapture_isOpened	CVAPI(ExceptionStatus) videoio_VideoCapture_isOpened(cv::VideoCapture *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isOpened() ? 1 : 0; END_WRAP }	cv::VideoCapture::isOpened()	classcv_1_1VideoCapture.html	Returns true if video capturing has been initialized already.		If the previous call to VideoCapture constructor or VideoCapture::open() succeeded, the method returns true.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/facedetect.cpp, samples/cpp/laplace.cpp, samples/cpp/lkdemo.cpp, samples/cpp/peopledetect.cpp, samples/cpp/polar_transforms.cpp, samples/cpp/segment_objects.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/videoio/video-write/video-write.cpp, samples/cpp/videowriter_basic.cpp, and	1							
videoio_VideoCapture_release	CVAPI(ExceptionStatus) videoio_VideoCapture_release(cv::VideoCapture *obj) { BEGIN_WRAP obj->release(); END_WRAP }	cv::VideoCapture::release()	classcv_1_1VideoCapture.html	Closes video file or capturing device.		The method is automatically called by subsequent VideoCapture::open and by VideoCapture destructor.The C function also deallocates memory and clears *capture pointer.	1							

videoio_VideoCapture_grab	CVAPI(ExceptionStatus) videoio_VideoCapture_grab(cv::VideoCapture *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->grab() ? 1 : 0; END_WRAP }	cv::VideoCapture::grab()	classcv_1_1VideoCapture.html	Grabs the next frame from video file or capturing device.		The method/function grabs the next frame from video file or camera and returns true (non-zero) in the case of success.The primary use of the function is in multi-camera environments, especially when the cameras do not have hardware synchronization. That is, you call VideoCapture::grab() for each camera and after that call the slower method VideoCapture::retrieve() to decode and get frame from each camera. This way the overhead on demosaicing or motion jpeg decompression etc. is eliminated and the retrieved frames from different cameras will be closer in time.Also, when a connected camera is multi-head (for example, a stereo camera or a Kinect device), the correct way of retrieving data from it is to call VideoCapture::grab() first and then call VideoCapture::retrieve() one or more times with different values of the channel parameter.Using Kinect and other OpenNI compatible depth sensors	1								
videoio_VideoCapture_retrieve_OutputArray	CVAPI(ExceptionStatus) videoio_VideoCapture_retrieve_OutputArray(cv::VideoCapture *obj, cv::_OutputArray *image, int flag, int *returnValue) { BEGIN_WRAP *returnValue = obj->retrieve(*image, flag) ? 1 : 0; END_WRAP }	cv::VideoCapture::retrieve()	classcv_1_1VideoCapture.html	Decodes and returns the grabbed video frame.	image : [out] flag : it could be a frame index or a driver specific flag	The method decodes and returns the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the method returns false and the function returns an empty image (with cv::Mat, test it with Mat::empty()).See alsoread()Note1n C API, functions cvRetrieveFrame() and cv.RetrieveFrame() return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using cvCloneImage and then do whatever you	2								
videoio_VideoCapture_retrieve_Mat	CVAPI(ExceptionStatus) videoio_VideoCapture_retrieve_Mat(cv::VideoCapture *obj, cv::Mat *image, int flag, int *returnValue) { BEGIN_WRAP *returnValue = obj->retrieve(*image, flag) ? 1 : 0; END_WRAP }	cv::VideoCapture::retrieve()	classcv_1_1VideoCapture.html	Decodes and returns the grabbed video frame.	image : [out] flag : it could be a frame index or a driver specific flag	The method decodes and returns the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the method returns false and the function returns an empty image (with cv::Mat, test it with Mat::empty()).See alsoread()Note1n C API, functions cvRetrieveFrame() and cv.RetrieveFrame() return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using cvCloneImage and then do whatever you	2								
videoio_VideoCapture_operatorRightShift_Mat	CVAPI(ExceptionStatus) videoio_VideoCapture_operatorRightShift_Mat(cv::VideoCapture *obj, cv::Mat *image) { BEGIN_WRAP (*obj) >> (*image); END_WRAP }	cv::VideoCapture	classcv_1_1VideoCapture.html	Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences. Here is how the class can be used: #include <opencv2/core.hpp>#include <opencv2/videoio.hpp>#include <opencv2/highgui.hpp>#include <iostream>#include <stdio.h>using namespace cv;using namespace std;int main(int, char**){ Mat frame; //--- INITIALIZE VIDEOCAPTURE VideoCapture cap; // open the default camera using default API // cap.open(0); // OR advance usage: select any API backend int deviceID = 0; // 0 = open default camera int apiID = cv::CAP_ANY; // 0 = autodetect default API // open selected camera using selected API cap.open(deviceID, apiID); // check if we succeeded if (!cap.isOpened()) { cerr << "ERROR! Unable to open camera\n"; return -1; } //--- GRAB AND WRITE LOOP cout << "Start grabbing" << endl << "Press any key to terminate" << endl; for (;;) { // wait for a new frame from camera and store it into 'frame' cap.read(frame); // check if					•						
videoio_VideoCapture_read_OutputArray	CVAPI(ExceptionStatus) videoio_VideoCapture_read_OutputArray(cv::VideoCapture *obj, cv::_OutputArray *image, int *returnValue) { BEGIN_WRAP *returnValue = obj->read(*image) ? 1 : 0; END_WRAP }	cv::VideoCapture::read()	classcv_1_1VideoCapture.html	Grabs, decodes and returns the next video frame.	image : [out]	The method/function combines VideoCapture::grab() and VideoCapture::retrieve() in one call. This is the most convenient method for reading video files or capturing data from decode and returns the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the method returns false and the function returns empty image (with cv::Mat, test it with Mat::empty()).Note1n C API, functions cvRetrieveFrame() and cv.RetrieveFrame() return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using cvCloneImage and then do whatever you want with the copy.Examples:	1								
videoio_VideoCapture_read_Mat	CVAPI(ExceptionStatus) videoio_VideoCapture_read_Mat(cv::VideoCapture *obj, cv::Mat *image, int *returnValue) { BEGIN_WRAP *returnValue = obj->read(*image) ? 1 : 0; END_WRAP }	cv::VideoCapture::read()	classcv_1_1VideoCapture.html	Grabs, decodes and returns the next video frame.	image : [out]	The method/function combines VideoCapture::grab() and VideoCapture::retrieve() in one call. This is the most convenient method for reading video files or capturing data from decode and returns the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the method returns false and the function returns empty image (with cv::Mat, test it with Mat::empty()).Note1n C API, functions cvRetrieveFrame() and cv.RetrieveFrame() return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using cvCloneImage and then do whatever you want with the copy.Examples:	1								
videoio_VideoCapture_set	CVAPI(ExceptionStatus) videoio_VideoCapture_set(cv::VideoCapture *obj, int propId, double value, int *returnValue) { BEGIN_WRAP *returnValue = obj->set(propId, value) ? 1 : 0; END_WRAP }	cv::VideoCapture::set()	classcv_1_1VideoCapture.html	Sets a property in the VideoCapture.	propId : Property identifier from cv::VideoCaptureProperties (eg. cv::CAP_PROP_POS_MSEC, cv::CAP_PROP_POS_FRAMES, ...) or one from Additional flags for video I/O API backends value : Value of the property.	NoteEven if it returns true this doesn't ensure that the property value has been accepted by the capture device. See note in VideoCapture::get()Examples: samples/cpp/laplace.cpp.	2								
videoio_VideoCapture_get	CVAPI(ExceptionStatus) videoio_VideoCapture_get(cv::VideoCapture *obj, int propId, double *returnValue) { BEGIN_WRAP *returnValue = obj->get(propId); END_WRAP }	cv::VideoCapture::get()	classcv_1_1VideoCapture.html	Returns the specified VideoCapture property.	propId : Property identifier from cv::VideoCaptureProperties (eg. cv::CAP_PROP_POS_MSEC, cv::CAP_PROP_POS_FRAMES, ...) or one from Additional flags for video I/O API backends	NoteReading / writing properties involves many layers. Some unexpected result might happens along this chain. VideoCapture -> API Backend -> Operating System -> Device Driver -> Device Hardware The returned value might be different from what really used by the device or it could be encoded using device dependent rules (eg. steps or percentage). Effective behaviour depends from device driver and API BackendExamples: samples/cpp/laplace.cpp, and samples/cpp/tutorial_code/videoio/video-write/video-write.cpp	1								
videoio_VideoCapture_getBackendName	CVAPI(ExceptionStatus) videoio_VideoCapture_getBackendName(cv::VideoCapture *obj, std::string *returnValue) { BEGIN_WRAP returnValue->assign(obj->getBackendName()); END_WRAP }	cv::VideoCapture::getBackendName()	classcv_1_1VideoCapture.html	Returns used backend API name.		NoteStream should be opened.	1								
videoio_VideoCapture_setExceptionMode	CVAPI(ExceptionStatus) videoio_VideoCapture_setExceptionMode(cv::VideoCapture *obj, int enable) { BEGIN_WRAP obj->setExceptionMode(enable != 0); END_WRAP }	cv::VideoCapture::setExceptionMode()	classcv_1_1VideoCapture.html	Switches exceptions mode		methods raise exceptions if not successful instead of returning an error code	1								
videoio_VideoCapture_getExceptionMode	CVAPI(ExceptionStatus) videoio_VideoCapture_getExceptionMode(cv::VideoCapture *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getExceptionMode() ? 1 : 0; END_WRAP }	cv::VideoCapture::getExceptionMode()	classcv_1_1VideoCapture.html	query if exception mode is active			1								
videoio_VideoCapture_waitAny	CVAPI(ExceptionStatus) videoio_VideoCapture_waitAny(cv::VideoCapture** streams, const size_t streamsSize, std::vector<int> *readyIndex, const int64 timeoutNs, int *returnValue) { BEGIN_WRAP std::vector<cv::VideoCapture> streamsVec(streamsSize); for (size_t i = 0; i < streamsSize; i++) streamsVec[i] = *streams[i]; *returnValue = cv::VideoCapture::waitAny(streamsVec, *readyIndex, timeoutNs) ? 1 : 0; END_WRAP }	cv::VideoCapture::waitAny()	classcv_1_1VideoCapture.html	Wait for ready frames from VideoCapture.	streams : input video streams readyIndex : stream indexes with grabbed frames (ready to use .retrieve() to fetch actual frame) timeoutNs : number of nanoseconds (0 - infinite)	Exceptions ExceptionException on stream errors (check .isOpened() to filter out malformed streams) or VideoCapture type is not supportedThe primary use of the function is in multi-camera environments. The method fills the ready state vector, grabs video frame, if camera is ready.After this call use VideoCapture::retrieve() to decode and fetch frame data.	3								

videoio_VideoWriter_new1	CVAPI(ExceptionStatus) videoio_VideoWriter_new1(cv ::VideoWriter **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoWriter; END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences.									•			
videoio_VideoWriter_new2	CVAPI(ExceptionStatus) videoio_VideoWriter_new2(const char *filename, int fourcc, double fps, MyCvSize frameSize, int isColor, cv::VideoWriter **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoWriter(filename, fourcc, fps, cpp(frameSize), isColor != 0); END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences.									•			
videoio_VideoWriter_new3	CVAPI(ExceptionStatus) videoio_VideoWriter_new3(const char *filename, int apiPreference, int fourcc, double fps, MyCvSize frameSize, int isColor, cv::VideoWriter **returnValue) { BEGIN_WRAP *returnValue = new cv::VideoWriter(filename, apiPreference, fourcc, fps, cpp(frameSize), isColor != 0); END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences.									•			
videoio_VideoWriter_new4	CVAPI(ExceptionStatus) videoio_VideoWriter_new4(const char* filename, int fourcc, double fps, MyCvSize frameSize, int* params, int paramsLength, cv::VideoWriter** returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); * returnValue = new cv::VideoWriter(filename, fourcc, fps, cpp(frameSize), paramsVec); END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences.									•			
videoio_VideoWriter_new5	CVAPI(ExceptionStatus) videoio_VideoWriter_new5(const char* filename, int apiPreference, int fourcc, double fps, MyCvSize frameSize, int* params, int paramsLength, cv::VideoWriter** returnValue) { BEGIN_WRAP std::vector<int> paramsVec; paramsVec.assign(params, params + paramsLength); * returnValue = new cv::VideoWriter(filename, apiPreference, fourcc, fps, cpp(frameSize), paramsVec); END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences.									•			
videoio_VideoWriter_delete	CVAPI(ExceptionStatus) videoio_VideoWriter_delete(cv ::VideoWriter *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::VideoWriter	classcv_1_1VideoWriter.html	Video writer class. The class provides C++ API for writing video files or image sequences. Examples: samples/cpp/tutorial_code/videoio/video-write/video-write.cpp, samples/cpp/videowriter_basic.cpp, and samples/tapi/hog.cpp.										•		
videoio_VideoWriter_open1	CVAPI(ExceptionStatus) videoio_VideoWriter_open1(cv::VideoWriter *obj, const char *filename, int fourcc, double fps, MyCvSize frameSize, int isColor, int *returnValue) { BEGIN_WRAP *returnValue = obj- >open(filename, fourcc, fps, cpp(frameSize), isColor != 0) ? 1 : 0; END_WRAP }	cv::VideoWriter::open()	classcv_1_1VideoWriter.html	Initializes or reinitializes video writer.			The method opens video writer. Parameters are the same as in the constructor VideoWriter::VideoWriter.The method first calls VideoWriter::release to close the already opened file.Examples: samples/cpp/tutorial_code/videoio/video-write/video-write.cpp, samples/cpp/videowriter_basic.cpp, and samples/tapi/hog.cpp.	5								
videoio_VideoWriter_open2	CVAPI(ExceptionStatus) videoio_VideoWriter_open2(cv::VideoWriter *obj, const char *filename, int apiPreference, int fourcc, double fps, MyCvSize frameSize, int isColor, int *returnValue) { BEGIN_WRAP *returnValue = obj- >open(filename, apiPreference, fourcc, fps, cpp(frameSize), isColor != 0) ? 1 : 0; END_WRAP }	cv::VideoWriter::open()	classcv_1_1VideoWriter.html	Initializes or reinitializes video writer.			The method opens video writer. Parameters are the same as in the constructor VideoWriter::VideoWriter.The method first calls VideoWriter::release to close the already opened file.Examples: samples/cpp/tutorial_code/videoio/video-write/video-write.cpp, samples/cpp/videowriter_basic.cpp, and samples/tapi/hog.cpp.	6								
videoio_VideoWriter_isOpened	CVAPI(ExceptionStatus) videoio_VideoWriter_isOpened (cv::VideoWriter *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >isOpened() ? 1 : 0; END_WRAP }	cv::VideoWriter::isOpened()	classcv_1_1VideoWriter.html	Returns true if video writer has been successfully initialized.			Examples: samples/cpp/videowriter_basic.cpp, and samples/tapi/hog.cpp.	1								
videoio_VideoWriter_release	CVAPI(ExceptionStatus) videoio_VideoWriter_release(c v::VideoWriter *obj) { BEGIN_WRAP obj->release(); END_WRAP }	cv::VideoWriter::release()	classcv_1_1VideoWriter.html	Closes the video writer.			The method is automatically called by subsequent VideoWriter::open and by the VideoWriter destructor.	1								
videoio_VideoWriter_write	CVAPI(ExceptionStatus) videoio_VideoWriter_write(cv: :VideoWriter *obj, cv::_InputArray *image) { BEGIN_WRAP obj->write(*image); END_WRAP }	cv::VideoWriter::write()	classcv_1_1VideoWriter.html	Writes the next video frame.	image : The written frame. In general, color images are expected in BGR format.		The function/method writes the specified image to video file. It must have the same size as has been specified when opening the video writer.Examples: samples/cpp/videowriter_basic.cpp.	1								
videoio_VideoWriter_set	CVAPI(ExceptionStatus) videoio_VideoWriter_set(cv:: VideoWriter *obj, int propId, double value, int *returnValue) { BEGIN_WRAP *returnValue = obj- >set(propId, value) ? 1 : 0; END_WRAP }	cv::VideoWriter::set()	classcv_1_1VideoWriter.html	Sets a property in the VideoWriter.	propId : Property identifier from cv::VideoWriterProperties (eg. cv::VIDEOWRITER_PROP_QUALITY) or one of Additional flags for video I/O API backends value : Value of the property.			2								
videoio_VideoWriter_get	CVAPI(ExceptionStatus) videoio_VideoWriter_get(cv:: VideoWriter *obj, int propId, double *returnValue) { BEGIN_WRAP *returnValue = obj- >get(propId); END_WRAP }	cv::VideoWriter::get()	classcv_1_1VideoWriter.html	Returns the specified VideoWriter property.	propId : Property identifier from cv::VideoWriterProperties (eg. cv::VIDEOWRITER_PROP_QUALITY) or one of Additional flags for video I/O API backends			1								
videoio_VideoWriter_fourcc	CVAPI(ExceptionStatus) videoio_VideoWriter_fourcc(ch ar c1, char c2, char c3, char c4, int *returnValue) { BEGIN_WRAP *returnValue = cv::VideoWriter::fourcc(c1, c2, c3, c4); END_WRAP }	cv::VideoWriter::fourcc()	classcv_1_1VideoWriter.html	Concatenates 4 chars to a fourcc code.			This static method constructs the fourcc code of the codec to be used in the constructor VideoWriter::VideoWriter or VideoWriter::open.Examples: samples/cpp/videowriter_basic.cpp.	4								
videoio_VideoWriter_getBackendName	CVAPI(ExceptionStatus) videoio_VideoWriter_getBacke ndName(cv::VideoWriter *obj, std::string *returnValue) { BEGIN_WRAP returnValue->assign(obj- >getBackendName()); END_WRAP }	cv::VideoWriter::getBackendName()	classcv_1_1VideoWriter.html	Returns used backend API name.			NoteStream should be opened.	1								

xfeatures2d_BriefDescriptorExtractor_create	CVAPI(ExceptionStatus) xfeatures2d_BriefDescriptorExtractor_create(int bytes, cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> **returnValue) { BEGIN_WRAP const auto ptr = cv::xfeatures2d::BriefDescriptorExtractor::create(bytes); *returnValue = clone(ptr); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor::create()	classcv_1_1xfeatures2d_1_1BriefDescriptorExtractor.html					1								
xfeatures2d_Ptr_BriefDescriptorExtractor_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_BriefDescriptorExtractor_delete(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor	classcv_1_1xfeatures2d_1_1BriefDescriptorExtractor.html	Class for computing BRIEF descriptors described in [40] . Parameters bytes legth of the descriptor in bytes, valid values are: 16, 32 (default) or 64 . use_orientation sample patterns using keypoints orientation, disabled by default.									•		
xfeatures2d_BriefDescriptorExtractor_read	CVAPI(ExceptionStatus) xfeatures2d_BriefDescriptorExtractor_read(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> *obj, cv::FileNode *fn) { BEGIN_WRAP obj->get()->read(*fn); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor::read()														
xfeatures2d_BriefDescriptorExtractor_write	CVAPI(ExceptionStatus) xfeatures2d_BriefDescriptorExtractor_write(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->get()->write(*fs); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor::write()														
xfeatures2d_BriefDescriptorExtractor_descriptorSize	CVAPI(ExceptionStatus) xfeatures2d_BriefDescriptorExtractor_descriptorSize(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->get()->descriptorSize(); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor::descriptorSize()														
xfeatures2d_BriefDescriptorExtractor_descriptorType	CVAPI(ExceptionStatus) xfeatures2d_BriefDescriptorExtractor_descriptorType(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor> *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->get()->descriptorType(); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor::descriptorType()														
xfeatures2d_Ptr_BriefDescriptorExtractor_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_BriefDescriptorExtractor_get(cv::Ptr<cv::xfeatures2d::BriefDescriptorExtractor>* ptr, cv::xfeatures2d::BriefDescriptorExtractor **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::BriefDescriptorExtractor	classcv_1_1xfeatures2d_1_1BriefDescriptorExtractor.html	Class for computing BRIEF descriptors described in [40] . Parameters bytes legth of the descriptor in bytes, valid values are: 16, 32 (default) or 64 . use_orientation sample patterns using keypoints orientation, disabled by default.									•		
xfeatures2d_FREAK_create	CVAPI(ExceptionStatus) xfeatures2d_FREAK_create(int orientationNormalized, int scaleNormalized, float patternScale, int nOctaves, int *selectedPairs, int selectedPairsLength, cv::Ptr<cv::xfeatures2d::FREAK> **returnValue) { BEGIN_WRAP std::vector<int> selectedPairsVec; if (selectedPairs != nullptr) selectedPairsVec = std::vector<int>(selectedPairs, selectedPairs + selectedPairsLength); const auto ptr = cv::xfeatures2d::FREAK::create(orientationNormalized != 0, scaleNormalized != 0, patternScale, nOctaves, selectedPairsVec);	cv::xfeatures2d::FREAK::create()	classcv_1_1xfeatures2d_1_1FREAK.html		orientationNormalized : Enable orientation normalization. scaleNormalized : Enable scale normalization. patternScale : Scaling of the description pattern. nOctaves : Number of octaves covered by the detected keypoints. selectedPairs : (Optional) user defined selected pairs indexes,				5							
xfeatures2d_Ptr_FREAK_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_FREAK_delete(cv::Ptr<cv::xfeatures2d::FREAK> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xfeatures2d::FREAK	classcv_1_1xfeatures2d_1_1FREAK.html	Class implementing the FREAK (Fast Retina Keypoint) keypoint descriptor, described in [8] . The algorithm propose a novel keypoint descriptor inspired by the human visual system and more precisely the retina, coined Fast Retina Key- point (FREAK). A cascade of binary strings is computed by efficiently comparing image intensities over a retinal sampling pattern. FREAKs are in general faster to compute with lower memory load and also more robust than SIFT, SURF or BRISK. They are competitive alternatives to existing keypoints in particular for embedded applications. Note An example on how to use the FREAK descriptor can be found at opencv_source_code/samples/cpp/freak_demo.cpp									•		
xfeatures2d_Ptr_FREAK_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_FREAK_get(cv::Ptr<cv::xfeatures2d::FREAK> *ptr, cv::xfeatures2d::FREAK **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::FREAK	classcv_1_1xfeatures2d_1_1FREAK.html	Class implementing the FREAK (Fast Retina Keypoint) keypoint descriptor, described in [8] . The algorithm propose a novel keypoint descriptor inspired by the human visual system and more precisely the retina, coined Fast Retina Key- point (FREAK). A cascade of binary strings is computed by efficiently comparing image intensities over a retinal sampling pattern. FREAKs are in general faster to compute with lower memory load and also more robust than SIFT, SURF or BRISK. They are competitive alternatives to existing keypoints in particular for embedded applications. Note An example on how to use the FREAK descriptor can be found at opencv_source_code/samples/cpp/freak_demo.cpp									•		
xfeatures2d_StarDetector_create	CVAPI(ExceptionStatus) xfeatures2d_StarDetector_create(int maxSize, int responseThreshold, int lineThresholdProjected, int lineThresholdBinarized, int suppressNonmaxSize, cv::Ptr<cv::xfeatures2d::StarDetector> **returnValue) { BEGIN_WRAP const auto ptr = cv::xfeatures2d::StarDetector::create(maxSize, responseThreshold, lineThresholdProjected, lineThresholdBinarized, suppressNonmaxSize); *returnValue = clone(ptr); END_WRAP }	cv::xfeatures2d::StarDetector::create()	classcv_1_1xfeatures2d_1_1StarDetector.html	the full constructor					5							
xfeatures2d_Ptr_StarDetector_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_StarDetector_delete(cv::Ptr<cv::xfeatures2d::StarDetector> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xfeatures2d::StarDetector	classcv_1_1xfeatures2d_1_1StarDetector.html	The class implements the keypoint detector introduced by [2], synonym of StarDetector. :									•		
xfeatures2d_Ptr_StarDetector_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_StarDetector_get(cv::Ptr<cv::xfeatures2d::StarDetector> *ptr, cv::xfeatures2d::StarDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::StarDetector	classcv_1_1xfeatures2d_1_1StarDetector.html	The class implements the keypoint detector introduced by [2], synonym of StarDetector. :									•		

xfeatures2d_LUCID_create	CVAPI(ExceptionStatus) xfeatures2d_LUCID_create(const int lucid_kernel, const int blur_kernel, cv::Ptr<cv::xfeatures2d::LUCID> **returnValue) { BEGIN_WRAP const auto ptr = cv::xfeatures2d::LUCID::create(lucid_kernel, blur_kernel); *returnValue = clone(ptr); END_WRAP }	cv::xfeatures2d::LUCID::create()	classcv_1_1xfeatures2d_1_1LUCID.html			lucid_kernel : kernel for descriptor construction, where 1=3x3, 2=5x5, 3=7x7 and so forth blur_kernel : kernel for blurring image prior to descriptor construction, where 1=3x3, 2=5x5, 3=7x7 and so forth		2								
xfeatures2d_Ptr_LUCID_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_LUCID_delete(cv::Ptr<cv::xfeatures2d::LUCID> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xfeatures2d::LUCID	classcv_1_1xfeatures2d_1_1LUCID.html	Class implementing the locally uniform comparison image descriptor, described in [295]. An image descriptor that can be computed very fast, while being about as robust as, for example, SURF or BRIEF. NoteIt requires a color image as input.								•		
xfeatures2d_Ptr_LUCID_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_LUCID_get(cv::Ptr<cv::xfeatures2d::LUCID> *ptr, cv::xfeatures2d::LUCID **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::LUCID	classcv_1_1xfeatures2d_1_1LUCID.html	Class implementing the locally uniform comparison image descriptor, described in [295]. An image descriptor that can be computed very fast, while being about as robust as, for example, SURF or BRIEF. NoteIt requires a color image as input.									•	
xfeatures2d_LATCH_create	CVAPI(ExceptionStatus) xfeatures2d_LATCH_create(int bytes, int rotationInvariance, int half_ssd_size, double sigma, cv::Ptr<cv::xfeatures2d::LATCH> **returnValue) { BEGIN_WRAP const auto ptr = cv::xfeatures2d::LATCH::create(bytes, rotationInvariance != 0, half_ssd_size, sigma); *returnValue = clone(ptr); END_WRAP }	cv::xfeatures2d::LATCH::create()	classcv_1_1xfeatures2d_1_1LATCH.html					4							
xfeatures2d_Ptr_LATCH_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_LATCH_delete(cv::Ptr<cv::xfeatures2d::LATCH> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xfeatures2d::LATCH	classcv_1_1xfeatures2d_1_1LATCH.html	latch Class for computing the LATCH descriptor. If you find this code useful, please add a reference to the following paper in your work: Gil Levi and Tal Hassner, "LATCH: Learned Arrangements of Three Patch Codes", arXiv preprint arXiv:1501.03719, 15 Jan. 2015 LATCH is a binary descriptor based on learned comparisons of triplets of image patches. bytes is the size of the descriptor - can be 64, 32, 16, 8, 4, 2 or 1 rotationInvariance - whether or not the descriptor should compansate for orientation changes. half_ssd_size - the size of half of the mini-patches size. For example, if we would like to compare triplets of patches of size 7x7x then the half_ssd_size should be (7-1)/2 = 3. sigma - sigma value for GaussianBlur smoothing of the source image. Source image will be used without smoothing in case sigma value is 0. Note: the descriptor can be coupled with any keypoint extractor. The only demand is that if you use set rotationInvariance = True then you will have to use an extractor which estimates the patch orientation (in degrees). Examples for such extractors are ORB and SIFT. Note: a complete example can be found under /samples/cpp/tutorial_code/xfeatures2D/latch_match.cpp.								•		
xfeatures2d_Ptr_LATCH_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_LATCH_get(cv::Ptr<cv::xfeatures2d::LATCH> *ptr, cv::xfeatures2d::LATCH **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::LATCH	classcv_1_1xfeatures2d_1_1LATCH.html	latch Class for computing the LATCH descriptor. If you find this code useful, please add a reference to the following paper in your work: Gil Levi and Tal Hassner, "LATCH: Learned Arrangements of Three Patch Codes", arXiv preprint arXiv:1501.03719, 15 Jan. 2015 LATCH is a binary descriptor based on learned comparisons of triplets of image patches. bytes is the size of the descriptor - can be 64, 32, 16, 8, 4, 2 or 1 rotationInvariance - whether or not the descriptor should compansate for orientation changes. half_ssd_size - the size of half of the mini-patches size. For example, if we would like to compare triplets of patches of size 7x7x then the half_ssd_size should be (7-1)/2 = 3. sigma - sigma value for GaussianBlur smoothing of the source image. Source image will be used without smoothing in case sigma value is 0. Note: the descriptor can be coupled with any keypoint extractor. The only demand is that if you use set rotationInvariance = True then you will have to use an extractor which estimates the patch orientation (in degrees). Examples for such extractors are ORB and SIFT. Note: a complete example can be found under /samples/cpp/tutorial_code/xfeatures2D/latch_match.cpp.									•	
xfeatures2d_SURF_create	CVAPI(ExceptionStatus) xfeatures2d_SURF_create(double hessianThreshold, int nOctaves, int nOctaveLayers, int extended, int upright, cv::Ptr<cv::xfeatures2d::SURF> **returnValue) { BEGIN_WRAP const auto ptr = cv::xfeatures2d::SURF::create(hessianThreshold, nOctaves, nOctaveLayers, extended != 0, upright != 0); *returnValue = clone(ptr); END_WRAP }	cv::xfeatures2d::SURF::create()	classcv_1_1xfeatures2d_1_1SURF.html		hessianThreshold : Threshold for hessian keypoint detector used in SURF. nOctaves : Number of pyramid octaves the keypoint detector will use. nOctaveLayers : Number of octave layers within each octave. extended : Extended descriptor flag (true - use extended 128-element descriptors; false - use 64-element descriptors). upright : Up-right or rotated features flag (true - do not compute orientation of features; false - compute orientation).		5								
xfeatures2d_Ptr_SURF_delete	CVAPI(ExceptionStatus) xfeatures2d_Ptr_SURF_delete(cv::Ptr<cv::xfeatures2d::SURF> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xfeatures2d::SURF	classcv_1_1xfeatures2d_1_1SURF.html	Class for extracting Speeded Up Robust Features from an image [16] . The algorithm parameters: member int extended 0 means that the basic descriptors (64 elements each) shall be computed 1 means that the extended descriptors (128 elements each) shall be computed member int upright 0 means that detector computes orientation of each feature. 1 means that the orientation is not computed (which is much, much faster). For example, if you match images from a stereo pair, or do image stitching, the matched features likely have very similar angles, and you can speed up feature extraction by setting upright=1. member double hessianThreshold Threshold for the keypoint detector. Only features, whose hessian is larger than hessianThreshold are retained by the detector. Therefore, the larger the value, the less keypoints you will get. A good									•	
xfeatures2d_Ptr_SURF_get	CVAPI(ExceptionStatus) xfeatures2d_Ptr_SURF_get(cv::Ptr<cv::xfeatures2d::SURF> *ptr, cv::xfeatures2d::SURF **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xfeatures2d::SURF	classcv_1_1xfeatures2d_1_1SURF.html	Class for extracting Speeded Up Robust Features from an image [16] . The algorithm parameters: member int extended 0 means that the basic descriptors (64 elements each) shall be computed 1 means that the extended descriptors (128 elements each) shall be computed member int upright 0 means that detector computes orientation of each feature. 1 means that the orientation is not computed (which is much, much faster). For example, if you match images from a stereo pair, or do image stitching, the matched features likely have very similar angles, and you can speed up feature extraction by setting upright=1. member double hessianThreshold Threshold for the keypoint detector. Only features, whose hessian is larger than hessianThreshold are retained by the detector. Therefore, the larger the value, the less keypoints you will get. A good									•	
xfeatures2d_SURF_getHessianThreshold	CVAPI(ExceptionStatus) xfeatures2d_SURF_getHessianThreshold(cv::xfeatures2d::SURF *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getHessianThreshold(); END_WRAP }	cv::xfeatures2d::SURF::getHessianThreshold()	classcv_1_1xfeatures2d_1_1SURF.html					1							
xfeatures2d_SURF_getNOctaves	CVAPI(ExceptionStatus) xfeatures2d_SURF_getNOctaves(cv::xfeatures2d::SURF *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaves(); END_WRAP }	cv::xfeatures2d::SURF::getNOctaves()	classcv_1_1xfeatures2d_1_1SURF.html					1							

xfeatures2d_SURF_getNOctaveLayers	CVAPI(ExceptionStatus) xfeatures2d_SURF_getNOctaveLayers(cv::xfeatures2d::SURF *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaveLayers(); END_WRAP }	cv::xfeatures2d::SURF::getNOctaveLayers()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_getExtended	CVAPI(ExceptionStatus) xfeatures2d_SURF_getExtended(cv::xfeatures2d::SURF *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getExtended() ? 1 : 0; END_WRAP }	cv::xfeatures2d::SURF::getExtended()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_getUpright	CVAPI(ExceptionStatus) xfeatures2d_SURF_getUpright(cv::xfeatures2d::SURF *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getUpright() ? 1 : 0; END_WRAP }	cv::xfeatures2d::SURF::getUpright()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_setHessianThreshold	CVAPI(ExceptionStatus) xfeatures2d_SURF_setHessianThreshold(cv::xfeatures2d::SURF *obj, double value) { BEGIN_WRAP obj->setHessianThreshold(value); END_WRAP }	cv::xfeatures2d::SURF::setHessianThreshold()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_setNOctaves	CVAPI(ExceptionStatus) xfeatures2d_SURF_setNOctaves(cv::xfeatures2d::SURF *obj, int value) { BEGIN_WRAP obj->setNOctaves(value); END_WRAP }	cv::xfeatures2d::SURF::setNOctaves()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_setNOctaveLayers	CVAPI(ExceptionStatus) xfeatures2d_SURF_setNOctaveLayers(cv::xfeatures2d::SURF *obj, int value) { BEGIN_WRAP obj->setNOctaveLayers(value); END_WRAP }	cv::xfeatures2d::SURF::setNOctaveLayers()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_setExtended	CVAPI(ExceptionStatus) xfeatures2d_SURF_setExtended(cv::xfeatures2d::SURF *obj, int value) { BEGIN_WRAP obj->setExtended(value != 0); END_WRAP }	cv::xfeatures2d::SURF::setExtended()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xfeatures2d_SURF_setUpright	CVAPI(ExceptionStatus) xfeatures2d_SURF_setUpright(cv::xfeatures2d::SURF *obj, int value) { BEGIN_WRAP obj->setUpright(value != 0); END_WRAP }	cv::xfeatures2d::SURF::setUpright()	classcv_1_1xfeatures2d_1_1SURF.html				1									
xphoto_bm3dDenoising1	CVAPI(ExceptionStatus) xphoto_bm3dDenoising1(cv::_InputArray *src, cv::_InputOutputArray *dstStep1, cv::_OutputArray *dstStep2, float h, int templateWindowSize, int searchWindowSize, int blockMatchingStep1, int blockMatchingStep2, int groupSize, int slidingStep, float beta, int normType, int step, int transformType) { BEGIN_WRAP cv::xphoto::bm3dDenoising(*src, *dstStep1, *dstStep2, h, templateWindowSize, searchWindowSize, blockMatchingStep1, blockMatchingStep2, groupSize, slidingStep, beta, normType, step, transformType); END_WRAP }	cv::xphoto::bm3dDenoising()	group__xphoto.html	Performs image denoising using the Block-Matching and 3D-filtering algorithm http://www.cs.tut.fi/~foi/GCF-BM3D/BM3D_TIP_2007.pdf with several computational optimizations. Noise expected to be a gaussian white noise.	src : Input 8-bit or 16-bit 1-channel image. dstStep1 : Output image of the first step of BM3D with the same size and type as src. dstStep2 : Output image of the second step of BM3D with the same size and type as src. h : Parameter regulating filter strength. Big h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise. templateWindowSize : Size in pixels of the template patch that is used for block-matching. Should be power of 2. searchWindowSize : Size in pixels of the window that is used to perform block-matching. Affect performance linearly: greater searchWindowsSize - greater denoising time. Must be larger than templateWindowSize. blockMatchingStep1 : Block matching threshold for the first step of BM3D (hard thresholding), i.e. maximum distance for which two blocks are considered similar. Value expressed in euclidean distance. blockMatchingStep2 : Block matching threshold for the second step of BM3D (Wiener filtering), i.e. maximum distance for which two blocks are considered similar. Value expressed in euclidean distance. groupSize : Maximum size of the 3D group for collaborative filtering.	This function expected to be applied to grayscale images. Advanced usage of this function can be manual denoising of colored image in different colorspaces.See also fastNlMeansDenoising	14								
xphoto_bm3dDenoising2	CVAPI(ExceptionStatus) xphoto_bm3dDenoising2(cv::_InputArray *src, cv::_OutputArray *dst, float h, int templateWindowSize, int searchWindowSize, int blockMatchingStep1, int blockMatchingStep2, int groupSize, int slidingStep, float beta, int normType, int step, int transformType) { BEGIN_WRAP cv::xphoto::bm3dDenoising(*src, *dst, h, templateWindowSize, searchWindowSize, blockMatchingStep1, blockMatchingStep2, groupSize, slidingStep, beta, normType, step, transformType); END_WRAP }	cv::xphoto::bm3dDenoising()	group__xphoto.html	Performs image denoising using the Block-Matching and 3D-filtering algorithm http://www.cs.tut.fi/~foi/GCF-BM3D/BM3D_TIP_2007.pdf with several computational optimizations. Noise expected to be a gaussian white noise.	src : Input 8-bit or 16-bit 1-channel image. dst : Output image with the same size and type as src. h : Parameter regulating filter strength. Big h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise. templateWindowSize : Size in pixels of the template patch that is used for block-matching. Should be power of 2. searchWindowSize : Size in pixels of the window that is used to perform block-matching. Affect performance linearly: greater searchWindowsSize - greater denoising time. Must be larger than templateWindowSize. blockMatchingStep1 : Block matching threshold for the first step of BM3D (hard thresholding), i.e. maximum distance for which two blocks are considered similar. Value expressed in euclidean distance. blockMatchingStep2 : Block matching threshold for the second step of BM3D (Wiener filtering), i.e. maximum distance for which two blocks are considered similar. Value expressed in euclidean distance. groupSize : Maximum size of the 3D group for collaborative filtering. slidingStep : Sliding step to process every next reference block. beta : Kaiser window parameter that affects the	This function expected to be applied to grayscale images. Advanced usage of this function can be manual denoising of colored image in different colorspaces.See also fastNlMeansDenoising	13								
xphoto_dctDenoising	CVAPI(ExceptionStatus) xphoto_dctDenoising(cv::Mat *src, cv::Mat *dst, const double sigma, const int psize) { BEGIN_WRAP cv::xphoto::dctDenoising(*src, *dst, sigma, psize); END_WRAP }	cv::xphoto::dctDenoising()	group__xphoto.html	The function implements simple dct-based denoising.	src : source image dst : destination image sigma : expected noise standard deviation psize : size of block side where dct is computed	http://www.ipol.im/pub/art/2011/ys-dct/.See also fastNlMeansDenoising	4								
xphoto_inpaint	CVAPI(ExceptionStatus) xphoto_inpaint(cv::Mat *src, cv::Mat *mask, cv::Mat *dst, int algorithm) { BEGIN_WRAP cv::xphoto::inpaint(*src, *mask, *dst, static_cast<const cv::xphoto::InpaintTypes>(algorithm)); END_WRAP }	cv::xphoto::inpaint()	group__xphoto.html	The function implements different single-image inpainting algorithms.	src : source image INPAINT_SHIFTMAP: it could be of any type and any number of channels from 1 to 4. In case of 3- and 4-channels images the function expect them in CIELab colorspace or similar one, where first color component shows intensity, while second and third shows colors. Nonetheless you can try any colorspaces. INPAINT_FSR_BEST or INPAINT_FSR_FAST: 1-channel grayscale or 3-channel BGR image. mask : mask (CV_8UC1), where non-zero pixels indicate valid image area, while zero pixels indicate area to be inpainted dst : destination image algorithmType : see xphoto::InpaintTypes	See the original papers [105] (Shiftmap) or [90] and [214] (FSR) for details.	4								
xphoto_oilPainting	CVAPI(ExceptionStatus) xphoto_oilPainting(cv::_InputArray *src, cv::_OutputArray *dst, int size, int dynRatio, int code) { BEGIN_WRAP if (code >= 0) cv::xphoto::oilPainting(*src, *dst, size, dynRatio, code); else cv::xphoto::oilPainting(*src, *dst, size, dynRatio); END_WRAP }	cv::xphoto::oilPainting()	group__xphoto.html	oilPainting See the book [39] for details.	src : Input three-channel or one channel image (either CV_8UC3 or CV_8UC1) dst : Output image of the same size and type as src. size : neighbouring size is 2-size+1 dynRatio : image is divided by dynRatio before histogram processing code : color space conversion code(see ColorConversionCodes). Histogram will used only first plane		8								
xphoto_TonemapDurand_getSaturation	CVAPI(ExceptionStatus) xphoto_TonemapDurand_getSaturation(cv::xphoto::TonemapDurand *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getSaturation(); END_WRAP }	cv::xphoto::TonemapDurand::getSaturation()	group__xphoto.html				1									
xphoto_TonemapDurand_setSaturation	CVAPI(ExceptionStatus) xphoto_TonemapDurand_setSaturation(cv::xphoto::TonemapDurand *obj, float saturation) { BEGIN_WRAP obj->setSaturation(saturation); END_WRAP }	cv::xphoto::TonemapDurand::setSaturation()	group__xphoto.html				1									

xphoto_TonemapDurand_getContrast	CVAPI(ExceptionStatus) xphoto_TonemapDurand_getContrast(cv::xphoto::TonemapDurand *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getContrast(); END_WRAP }	cv::xphoto::TonemapDurand::getContrast()	group__xphoto.html				1										
xphoto_TonemapDurand_setContrast	CVAPI(ExceptionStatus) xphoto_TonemapDurand_setContrast(cv::xphoto::TonemapDurand *obj, float contrast) { BEGIN_WRAP obj->setContrast(contrast); END_WRAP }	cv::xphoto::TonemapDurand::setContrast()	group__xphoto.html				1										
xphoto_TonemapDurand_getSigmaSpace	CVAPI(ExceptionStatus) xphoto_TonemapDurand_getSigmaSpace(cv::xphoto::TonemapDurand *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getSigmaSpace(); END_WRAP }	cv::xphoto::TonemapDurand::getSigmaSpace()	group__xphoto.html				1										
xphoto_TonemapDurand_setSigmaSpace	CVAPI(ExceptionStatus) xphoto_TonemapDurand_setSigmaSpace(cv::xphoto::TonemapDurand *obj, float sigma_space) { BEGIN_WRAP obj->setSigmaSpace(sigma_space); END_WRAP }	cv::xphoto::TonemapDurand::setSigmaSpace()	group__xphoto.html				1										
xphoto_TonemapDurand_getSigmaColor	CVAPI(ExceptionStatus) xphoto_TonemapDurand_getSigmaColor(cv::xphoto::TonemapDurand *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getSigmaColor(); END_WRAP }	cv::xphoto::TonemapDurand::getSigmaColor()	group__xphoto.html				1										
xphoto_TonemapDurand_setSigmaColor	CVAPI(ExceptionStatus) xphoto_TonemapDurand_setSigmaColor(cv::xphoto::TonemapDurand *obj, float sigma_color) { BEGIN_WRAP obj->setSigmaColor(sigma_color); END_WRAP }	cv::xphoto::TonemapDurand::setSigmaColor()	group__xphoto.html				1										
xphoto_createTonemapDurand	CVAPI(ExceptionStatus) xphoto_createTonemapDurand(float gamma, float contrast, float saturation, float sigma_space, float sigma_color, cv::Ptr<cv::xphoto::TonemapDurand> **returnValue) { BEGIN_WRAP const auto p = cv::xphoto::createTonemapDurand(gamma, contrast, saturation, sigma_space, sigma_color); *returnValue = clone(p); END_WRAP }	cv::xphoto::createTonemapDurand()	group__xphoto.html	Creates TonemapDurand object.	gamma : gamma value for gamma correction. See createTonemap contrast : resulting contrast on logarithmic scale, i. e. log(max / min), where max and min are maximum and minimum luminance values of the resulting image. saturation : saturation enhancement value. See createTonemapDrago sigma_color : bilateral filter sigma in color space sigma_space : bilateral filter sigma in coordinate space	You need to set the OPENCV_ENABLE_NONFREE option in cmake to use those. Use them at your own risk.	5								
xphoto_Ptr_TonemapDurand_delete	CVAPI(ExceptionStatus) xphoto_Ptr_TonemapDurand_delete(cv::Ptr<cv::xphoto::TonemapDurand> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::xphoto::TonemapDurand	classcv_1_1xphoto_1_1TonemapDurand.html	This algorithm decomposes image into two layers: base layer and detail layer using bilateral filter and compresses contrast of the base layer thus preserving all the details. This implementation uses regular bilateral filter from OpenCV. Saturation enhancement is possible as in cv::TonemapDrago. For more information see [63] .								•		
xphoto_Ptr_TonemapDurand_get	CVAPI(ExceptionStatus) xphoto_Ptr_TonemapDurand_get(cv::Ptr<cv::xphoto::TonemapDurand> *ptr, cv::xphoto::TonemapDurand **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xphoto::TonemapDurand	classcv_1_1xphoto_1_1TonemapDurand.html	This algorithm decomposes image into two layers: base layer and detail layer using bilateral filter and compresses contrast of the base layer thus preserving all the details. This implementation uses regular bilateral filter from OpenCV. Saturation enhancement is possible as in cv::TonemapDrago. For more information see [63] .										•
xphoto_applyChannelGains	CVAPI(ExceptionStatus) xphoto_applyChannelGains(cv::_InputArray *src, cv::_OutputArray *dst, float gainB, float gainG, float gainR) { BEGIN_WRAP cv::xphoto::applyChannelGain(*src, *dst, gainB, gainG, gainR); END_WRAP }	cv::xphoto::applyChannelGains()	group__xphoto.html	Implements an efficient fixed-point approximation for applying channel gains, which is the last step of multiple white balance algorithms.	src : Input three-channel image in the BGR color space (either CV_8UC3 or CV_16UC3) dst : Output image of the same size and type as src. gainB : gain for the B channel gainG : gain for the G channel gainR : gain for the R channel		5								
xphoto_createGrayworldWB	CVAPI(ExceptionStatus) xphoto_createGrayworldWB(cv::Ptr<cv::xphoto::GrayworldWB> **returnValue) { BEGIN_WRAP const auto ptr = cv::xphoto::createGrayworldWB(); *returnValue = new cv::Ptr<cv::xphoto::GrayworldWB>(ptr); END_WRAP }	cv::xphoto::createGrayworldWB	group__xphoto.html	Creates an instance of GrayworldWB.										
xphoto_Ptr_GrayworldWB_delete	CVAPI(ExceptionStatus) xphoto_Ptr_GrayworldWB_delete(cv::Ptr<cv::xphoto::GrayworldWB> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::xphoto::createGrayworldWB	group__xphoto.html	Creates an instance of GrayworldWB.								•		
xphoto_Ptr_GrayworldWB_get	CVAPI(ExceptionStatus) xphoto_Ptr_GrayworldWB_get(cv::Ptr<cv::xphoto::GrayworldWB>* ptr, cv::xphoto::GrayworldWB **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xphoto::createGrayworldWB	group__xphoto.html	Creates an instance of GrayworldWB.										•
xphoto_GrayworldWB_balanceWhite	CVAPI(ExceptionStatus) xphoto_GrayworldWB_balanceWhite(cv::xphoto::GrayworldWB* ptr, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP ptr->balanceWhite(*src, *dst); END_WRAP }	cv::xphoto::createGrayworldWB::balanceWhite()													
xphoto_GrayworldWB_SaturationThreshold_get	CVAPI(ExceptionStatus) xphoto_GrayworldWB_SaturationThreshold_get(cv::xphoto::GrayworldWB* ptr, float* returnValue) { BEGIN_WRAP *returnValue = ptr->getSaturationThreshold(); END_WRAP }	cv::xphoto::createGrayworldWB::getSaturationThreshold()													
xphoto_GrayworldWB_SaturationThreshold_set	CVAPI(ExceptionStatus) xphoto_GrayworldWB_SaturationThreshold_set(cv::xphoto::GrayworldWB* ptr, float val) { BEGIN_WRAP ptr->setSaturationThreshold(val); END_WRAP }	cv::xphoto::createGrayworldWB::setSaturationThreshold()													

xphoto_createLearningBasedWB	CVAPI(ExceptionStatus) xphoto_createLearningBasedWB(const char* path_to_model, cv::Ptr<cv::xphoto::LearningBasedWB> **returnValue) { BEGIN_WRAP const std::string str_path_to_model(path_to_model); const auto ptr = cv::xphoto::createLearningBasedWB(str_path_to_model); *returnValue = new cv::Ptr<cv::xphoto::LearningBasedWB>(ptr); END_WRAP }	cv::xphoto::createLearningBasedWB()	group__xphoto.html	Creates an instance of LearningBasedWB.	path_to_model : Path to a .yml file with the model. If not specified, the default model is used		1									
xphoto_Ptr_LearningBasedWB_delete	CVAPI(ExceptionStatus) xphoto_Ptr_LearningBasedWB_delete(cv::Ptr<cv::xphoto::LearningBasedWB> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::xphoto::LearningBasedWB	classcv_1_1xphoto_1_1LearningBasedWB.html	More sophisticated learning-based automatic white balance algorithm. As GrayworldWB, this algorithm works by applying different gains to the input image channels, but their computation is a bit more involved compared to the simple gray-world assumption. More details about the algorithm can be found in [46] . To mask out saturated pixels this function uses only pixels that satisfy the following condition: ¥[¥frac{¥texttt{max}(R,G,B)}{¥texttt{range_max_val}} < ¥texttt{saturation_thresh} ¥] Currently supports images of type CV_8UC3 and CV_16UC3.								•		
xphoto_Ptr_LearningBasedWB_get	CVAPI(ExceptionStatus) xphoto_Ptr_LearningBasedWB_get(cv::Ptr<cv::xphoto::LearningBasedWB>* ptr, cv::xphoto::LearningBasedWB **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xphoto::LearningBasedWB	classcv_1_1xphoto_1_1LearningBasedWB.html	More sophisticated learning-based automatic white balance algorithm. As GrayworldWB, this algorithm works by applying different gains to the input image channels, but their computation is a bit more involved compared to the simple gray-world assumption. More details about the algorithm can be found in [46] . To mask out saturated pixels this function uses only pixels that satisfy the following condition: ¥[¥frac{¥texttt{max}(R,G,B)}{¥texttt{range_max_val}} < ¥texttt{saturation_thresh} ¥] Currently supports images of type CV_8UC3 and CV_16UC3.								•		
xphoto_LearningBasedWB_balanceWhite	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_balanceWhite(cv::xphoto::LearningBasedWB* ptr, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP ptr->balanceWhite(*src, *dst); END_WRAP }	cv::xphoto::LearningBasedWB::balanceWhite()														
xphoto_LearningBasedWB_extractSimpleFeatures	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_extractSimpleFeatures(cv::xphoto::LearningBasedWB* ptr, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP ptr- >extractSimpleFeatures(*src, *dst); END_WRAP }	cv::xphoto::LearningBasedWB::extractSimpleFeatures()	classcv_1_1xphoto_1_1LearningBasedWB.html	Implements the feature extraction part of the algorithm.	src : Input three-channel image (BGR color space is assumed). dst : An array of four (r,g) chromaticity tuples corresponding to the features listed above.	In accordance with [46] , computes the following features for the input image:Chromaticity of an average (R,G,B) tuple Chromaticity of the brightest (R,G,B) tuple (while ignoring saturated pixels) Chromaticity of the dominant (R,G,B) tuple (the one that has the highest value in the RGB histogram) Mode of the chromaticity palette, that is constructed by taking 300 most common colors according to the RGB histogram and projecting them on the chromaticity plane. Mode is the most high-density point of the palette, which is computed by a straightforward fixed-bandwidth kernel density estimator with a Epanechnikov kernel function	2								
xphoto_LearningBasedWB_HistBinNum_set	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_HistBinNum_set(cv::xphoto::LearningBasedWB* ptr, int val) { BEGIN_WRAP ptr->setHistBinNum(val); END_WRAP }	cv::xphoto::LearningBasedWB::setHistBinNum()	classcv_1_1xphoto_1_1LearningBasedWB.html	Defines the size of one dimension of a three-dimensional RGB histogram that is used internally by the algorithm. It often makes sense to increase the number of bins for images with higher bit depth (e.g. 256 bins for a 12 bit image).		See alsogetHistBinNum	1								
xphoto_LearningBasedWB_RangeMaxVal_set	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_RangeMaxVal_set(cv::xphoto::LearningBasedWB* ptr, int val) { BEGIN_WRAP ptr->setRangeMaxVal(val); END_WRAP }	cv::xphoto::LearningBasedWB::setRangeMaxVal()	classcv_1_1xphoto_1_1LearningBasedWB.html	Maximum possible value of the input image (e.g. 255 for 8 bit images, 4095 for 12 bit images)		See alsogetRangeMaxVal	1								
xphoto_LearningBasedWB_SaturationThreshold	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_SaturationThreshold_set(cv::xphoto::LearningBasedWB* ptr, float val) { BEGIN_WRAP ptr- >setSaturationThreshold(val); END_WRAP }	cv::xphoto::LearningBasedWB::setSaturationThreshold()	classcv_1_1xphoto_1_1LearningBasedWB.html	Threshold that is used to determine saturated pixels, i.e. pixels where at least one of the channels exceeds ¥(¥texttt{saturation_threshold}¥times¥texttt{range_max_val}¥) are ignored.		See alsogetSaturationThreshold	1								
xphoto_LearningBasedWB_HistBinNum_get	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_HistBinNum_get(cv::xphoto::LearningBasedWB* ptr, int *returnValue) { BEGIN_WRAP *returnValue = ptr- >getHistBinNum(); END_WRAP }	cv::xphoto::LearningBasedWB::getHistBinNum()	classcv_1_1xphoto_1_1LearningBasedWB.html	Defines the size of one dimension of a three-dimensional RGB histogram that is used internally by the algorithm. It often makes sense to increase the number of bins for images with higher bit depth (e.g. 256 bins for a 12 bit image).		See alsosetHistBinNum	1								
xphoto_LearningBasedWB_RangeMaxVal_get	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_RangeMaxVal_get(cv::xphoto::LearningBasedWB* ptr, int *returnValue) { BEGIN_WRAP *returnValue = ptr- >getRangeMaxVal(); END_WRAP }	cv::xphoto::LearningBasedWB::getRangeMaxVal()	classcv_1_1xphoto_1_1LearningBasedWB.html	Maximum possible value of the input image (e.g. 255 for 8 bit images, 4095 for 12 bit images)		See alsosetRangeMaxVal	1								
xphoto_LearningBasedWB_SaturationThreshold_get	CVAPI(ExceptionStatus) xphoto_LearningBasedWB_SaturationThreshold_get(cv::xphoto::LearningBasedWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr- >getSaturationThreshold(); END_WRAP }	cv::xphoto::LearningBasedWB::getSaturationThreshold()	classcv_1_1xphoto_1_1LearningBasedWB.html	Threshold that is used to determine saturated pixels, i.e. pixels where at least one of the channels exceeds ¥(¥texttt{saturation_threshold}¥times¥texttt{range_max_val}¥) are ignored.		See alsosetSaturationThreshold	1								
xphoto_createSimpleWB	CVAPI(ExceptionStatus) xphoto_createSimpleWB(cv::Ptr<cv::xphoto::SimpleWB> **returnValue) { BEGIN_WRAP const auto ptr = cv::xphoto::createSimpleWB( ); *returnValue = new cv::Ptr<cv::xphoto::SimpleWB>(ptr); END_WRAP }	cv::xphoto::createSimpleWB()	group__xphoto.html	Creates an instance of SimpleWB.			1									
xphoto_Ptr_SimpleWB_delete	CVAPI(ExceptionStatus) xphoto_Ptr_SimpleWB_delete(cv::Ptr<cv::xphoto::SimpleWB> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::xphoto::SimpleWB	classcv_1_1xphoto_1_1SimpleWB.html	A simple white balance algorithm that works by independently stretching each of the input image channels to the specified range. For increased robustness it ignores the top and bottom ¥(p¥%¥) of pixel values.								•		
xphoto_Ptr_SimpleWB_get	CVAPI(ExceptionStatus) xphoto_Ptr_SimpleWB_get(cv::Ptr<cv::xphoto::SimpleWB>* ptr, cv::xphoto::SimpleWB **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::xphoto::SimpleWB	classcv_1_1xphoto_1_1SimpleWB.html	A simple white balance algorithm that works by independently stretching each of the input image channels to the specified range. For increased robustness it ignores the top and bottom ¥(p¥%¥) of pixel values.								•		
xphoto_SimpleWB_balanceWhite	CVAPI(ExceptionStatus) xphoto_SimpleWB_balanceWhite(cv::xphoto::SimpleWB* ptr, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP ptr->balanceWhite(*src, *dst); END_WRAP }	cv::xphoto::SimpleWB::balanceWhite()														

xphoto_SimpleWB_InputMax_get	CVAPI(ExceptionStatus) xphoto_SimpleWB_InputMax_get(cv::xphoto::SimpleWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr->getInputMax(); END_WRAP }	cv::xphoto::SimpleWB::getInputMax()	classcv_1_1xphoto_1_1SimpleWB.html	Input image range maximum value.		See alsosetInputMax	1									
xphoto_SimpleWB_InputMax_set	CVAPI(ExceptionStatus) xphoto_SimpleWB_InputMax_set(cv::xphoto::SimpleWB* ptr, float val) { BEGIN_WRAP ptr->setInputMax(val); END_WRAP }	cv::xphoto::SimpleWB::setInputMax()	classcv_1_1xphoto_1_1SimpleWB.html	Input image range maximum value.		See alsogetInputMax	1									
xphoto_SimpleWB_InputMin_get	CVAPI(ExceptionStatus) xphoto_SimpleWB_InputMin_get(cv::xphoto::SimpleWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr->getInputMin(); END_WRAP }	cv::xphoto::SimpleWB::getInputMin()	classcv_1_1xphoto_1_1SimpleWB.html	Input image range minimum value.		See alsosetInputMin	1									
xphoto_SimpleWB_InputMin_set	CVAPI(ExceptionStatus) xphoto_SimpleWB_InputMin_set(cv::xphoto::SimpleWB* ptr, float val) { BEGIN_WRAP ptr->setInputMin(val); END_WRAP }	cv::xphoto::SimpleWB::setInputMin()	classcv_1_1xphoto_1_1SimpleWB.html	Input image range minimum value.		See alsogetInputMin	1									
xphoto_SimpleWB_OutputMax_get	CVAPI(ExceptionStatus) xphoto_SimpleWB_OutputMax_get(cv::xphoto::SimpleWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr->getOutputMax(); END_WRAP }	cv::xphoto::SimpleWB::getOutputMax()	classcv_1_1xphoto_1_1SimpleWB.html	Output image range maximum value.		See alsosetOutputMax	1									
xphoto_SimpleWB_OutputMax_set	CVAPI(ExceptionStatus) xphoto_SimpleWB_OutputMax_set(cv::xphoto::SimpleWB* ptr, float val) { BEGIN_WRAP ptr->setOutputMax(val); END_WRAP }	cv::xphoto::SimpleWB::setOutputMax()	classcv_1_1xphoto_1_1SimpleWB.html	Output image range maximum value.		See alsogetOutputMax	1									
xphoto_SimpleWB_OutputMin_get	CVAPI(ExceptionStatus) xphoto_SimpleWB_OutputMin_get(cv::xphoto::SimpleWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr->getOutputMin(); END_WRAP }	cv::xphoto::SimpleWB::getOutputMin()	classcv_1_1xphoto_1_1SimpleWB.html	Output image range minimum value.		See alsosetOutputMin	1									
xphoto_SimpleWB_OutputMin_set	CVAPI(ExceptionStatus) xphoto_SimpleWB_OutputMin_set(cv::xphoto::SimpleWB* ptr, float val) { BEGIN_WRAP ptr->setOutputMin(val); END_WRAP }	cv::xphoto::SimpleWB::setOutputMin()	classcv_1_1xphoto_1_1SimpleWB.html	Output image range minimum value.		See alsogetOutputMin	1									
xphoto_SimpleWB_P_get	CVAPI(ExceptionStatus) xphoto_SimpleWB_P_get(cv::xphoto::SimpleWB* ptr, float *returnValue) { BEGIN_WRAP *returnValue = ptr->getP(); END_WRAP }	cv::xphoto::SimpleWB::getP()	classcv_1_1xphoto_1_1SimpleWB.html	Percent of top/bottom values to ignore.		See alsosetP	1									
xphoto_SimpleWB_P_set	CVAPI(ExceptionStatus) xphoto_SimpleWB_P_set(cv::xphoto::SimpleWB* ptr, float val) { BEGIN_WRAP ptr->setP(val); END_WRAP }	cv::xphoto::SimpleWB::setP()	classcv_1_1xphoto_1_1SimpleWB.html	Percent of top/bottom values to ignore.		See alsogetP	1									
calib3d_Rodrigues	CVAPI(ExceptionStatus) calib3d_Rodrigues(cv::_InputArray *src, cv::_OutputArray *dst, cv::_OutputArray *jacobian) { BEGIN_WRAP cv::Rodrigues(*src, *dst, ensity(jacobian)); END_WRAP }	cv::Rodrigues()	group__calib3d.html	Converts a rotation matrix to a rotation vector or vice versa.	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 calibrateCamera, stereoCalibrate, or solvePnP .NoteMore 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 [83]	3									
calib3d_findHomography_InputArray	CVAPI(ExceptionStatus) calib3d_findHomography_InputArray(cv::_InputArray *srcPoints, cv::_InputArray *dstPoints, int method, double ransacReprojThreshold, cv::_OutputArray *mask, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::findHomography(*srcPoints, *dstPoints, method, ransacReprojThreshold, entity(mask)); *returnValue = new cv::Mat(ret); END_WRAP }	cv::findHomography()	group__calib3d.html	Finds a perspective transformation between two planes.	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 RANSAC - RANSAC-based robust method LMEDS - Least-Median robust method 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 ¥[¥| ¥textit{dstPoints} _i - ¥textit{convertPointsHomogeneous} (¥textit{H} * ¥textit{srcPoints} _i) ¥|_2 > ¥textit{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 (	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	5									
calib3d_findHomography_vector	CVAPI(ExceptionStatus) calib3d_findHomography_vector(cv::Point2d *srcPoints, int srcPointsLength, cv::Point2d *dstPoints, int dstPointsLength, int method, double ransacReprojThreshold, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat srcPointsMat(srcPointsLength, 1, CV_64FC2, srcPoints); const cv::Mat dstPointsMat(dstPointsLength, 1, CV_64FC2, dstPoints); const auto ret = cv::findHomography(srcPointsMat, dstPointsMat, method, ransacReprojThreshold, entity(mask)); *returnValue = new cv::Mat(ret);	cv::findHomography()	group__calib3d.html	Finds a perspective transformation between two planes.	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 RANSAC - RANSAC-based robust method LMEDS - Least-Median robust method 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 ¥[¥| ¥textit{dstPoints} _i - ¥textit{convertPointsHomogeneous} (¥textit{H} * ¥textit{srcPoints} _i) ¥|_2 > ¥textit{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 (	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	5									
calib3d_RQDecomp3x3_InputArray	CVAPI(ExceptionStatus) calib3d_RQDecomp3x3_InputArray(cv::_InputArray *src, cv::_OutputArray *mtxR, cv::_OutputArray *mtxQ, cv::_OutputArray *qx, cv::_OutputArray *qy, cv::_OutputArray *qz, cv::Vec3d *outVec) { BEGIN_WRAP *outVec = cv::RQDecomp3x3(*src, *mtxR, *mtxQ, entity(qx), entity(qy), entity(qz)); END_WRAP }	cv::RQDecomp3x3()	group__calib3d.html	Computes an RQ decomposition of 3x3 matrices.	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 [225] . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions.	6									

calib3d_RQDecomp3x3_Mat	CVAPI(ExceptionStatus) calib3d_RQDecomp3x3_Mat(cv::Mat *src, cv::Mat *mtxR, cv::Mat *mtxQ, cv::Mat *qx, cv::Mat *qy, cv::Mat *qz, cv::Vec3d *outVec) { BEGIN_WRAP *outVec = cv::RQDecomp3x3(*src, *mtxR, *mtxQ, *qx, *qy, *qz); END_WRAP }	cv::RQDecomp3x3()	group__calib3d.html	Computes an RQ decomposition of 3x3 matrices.	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 [225] . Returned tree rotation matrices and corresponding three Euler angles are only	6								
calib3d_decomposeProjectionMatrix_InputArray	CVAPI(ExceptionStatus) calib3d_decomposeProjection Matrix_InputArray(cv::_InputArray *projMatrix, cv::_OutputArray *cameraMatrix, cv::_OutputArray *rotMatrix, cv::_OutputArray *transVect, cv::_OutputArray *rotMatrixX, cv::_OutputArray *rotMatrixY, cv::_OutputArray *rotMatrixZ, cv::_OutputArray *eulerAngles) { BEGIN_WRAP cv::decomposeProjectionMatri x(*projMatrix, *cameraMatrix, *rotMatrix, *transVect, entity(rotMatrixX), entity(rotMatrixY), entity(rotMatrixZ), entity(eulerAngles)); END_WRAP }	cv::decomposeProjectionMatrix()	group__calib3d.html	Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.	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 [225] . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions.The function is based on RQDecomp3x3 .	8								
calib3d_decomposeProjectionMatrix_Mat	CVAPI(ExceptionStatus) calib3d_decomposeProjection Matrix_Mat(cv::Mat *projMatrix, cv::Mat *cameraMatrix, cv::Mat *rotMatrix, cv::Mat *transVect, cv::Mat *rotMatrixX, cv::Mat *rotMatrixY, cv::Mat *rotMatrixZ, cv::Mat *eulerAngles) { BEGIN_WRAP cv::decomposeProjectionMatri x(*projMatrix, *cameraMatrix, *rotMatrix, *transVect, *rotMatrixX, *rotMatrixY, *rotMatrixZ, *eulerAngles); END_WRAP }	cv::decomposeProjectionMatrix()	group__calib3d.html	Decomposes a projection matrix into a rotation matrix and a camera intrinsic matrix.	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 [225] . Returned tree rotation matrices and corresponding three Euler angles are only one of the possible solutions.The function is based on RQDecomp3x3 .	8								
calib3d_matMulDeriv	CVAPI(ExceptionStatus) calib3d_matMulDeriv(cv::_InputArray *a, cv::_InputArray *b, cv::_OutputArray *dABdA, cv::_OutputArray *dABdB) { BEGIN_WRAP cv::matMulDeriv(*a, *b, *dABdA, *dABdB); END_WRAP }	cv::matMulDeriv()	group__calib3d.html	Computes partial derivatives of the matrix product for each multiplied matrix.	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.	4								
calib3d_composeRT_InputArray	CVAPI(ExceptionStatus) calib3d_composeRT_InputArra y(cv::_InputArray *rvec1, cv::_InputArray *tvec1, cv::_InputArray *rvec2, cv::_InputArray *tvec2, cv::_OutputArray *rvec3, cv::_OutputArray *tvec3, cv::_OutputArray *dr3dr1, cv::_OutputArray *dr3dt1, cv::_OutputArray *dr3dr2, cv::_OutputArray *dr3dt2, cv::_OutputArray *dt3dr1, cv::_OutputArray *dt3dt1, cv::_OutputArray *dt3dr2, cv::_OutputArray *dt3dt2) { BEGIN_WRAP cv::composeRT(*rvec1, *tvec1, *rvec2, *tvec2, *rvec3, *tvec3, entity(dr3dr1), entity(dr3dt1), entity(dr3dr2), entity(dr3dt2), entity(dt3dr1), entity(dt3dt1), entity(dt3dr2),	cv::composeRT()	group__calib3d.html	Combines two rotation-and-shift transformations.	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 dr3dt1 : Optional output derivative of rvec3 with regard to tvec1 dr3dr2 : Optional output derivative of rvec3 with regard to tvec2 dr3dt2 : Optional output derivative of rvec3 with regard to tvec2 dt3dr1 : Optional output derivative of tvec3 with regard to rvec1 dt3dt1 : Optional output derivative of tvec3 with regard to tvec1 dt3dr2 : Optional output derivative of tvec3 with regard to rvec2 dt3dt2 : 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.	14								
calib3d_composeRT_Mat	CVAPI(ExceptionStatus) calib3d_composeRT_Mat(cv::Mat *rvec1, cv::Mat *tvec1, cv::Mat *rvec2, cv::Mat *tvec2, cv::Mat *rvec3, cv::Mat *tvec3, cv::Mat *dr3dr1, cv::Mat *dr3dt1, cv::Mat *dr3dr2, cv::Mat *dr3dt2, cv::Mat *dt3dr1, cv::Mat *dt3dt1, cv::Mat *dt3dr2, cv::Mat *dt3dt2) { BEGIN_WRAP cv::composeRT(*rvec1, *tvec1, *rvec2, *tvec2, *rvec3, *tvec3, entity(dr3dr1), entity(dr3dt1), entity(dr3dr2), entity(dr3dt2), entity(dt3dr1), entity(dt3dt1), entity(dt3dr2),	cv::composeRT()	group__calib3d.html	Combines two rotation-and-shift transformations.	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 dr3dt1 : Optional output derivative of rvec3 with regard to tvec1 dr3dr2 : Optional output derivative of rvec3 with regard to rvec2 dr3dt2 : Optional output derivative of rvec3 with regard to tvec2 dt3dr1 : Optional output derivative of tvec3 with regard to rvec1 dt3dt1 : Optional output derivative of tvec3 with regard to tvec1 dt3dr2 : Optional output derivative of tvec3 with regard to rvec2 dt3dt2 : 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.	14								
calib3d_projectPoints_InputArray	CVAPI(ExceptionStatus) calib3d_projectPoints_InputAr ray(cv::_InputArray *objectPoints, cv::_InputArray *rvec, cv::_InputArray *tvec, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_OutputArray *imagePoints, cv::_OutputArray *jacobian, double aspectRatio) { BEGIN_WRAP cv::projectPoints(*objectPoint s, *rvec, *tvec, *cameraMatrix, *distCoeffs, *imagePoints, entity(jacobian), aspectRatio); END_WRAP }	cv::projectPoints()	group__calib3d.html	Projects 3D points to an image plane.	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 (Rodrigues) that, together with tvec, performs a change of basis from world to camera coordinate system, see 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 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 calibrateCamera, solvePnP, and stereoCalibrate. The function itself can also be used to compute a re-projection error, given the current intrinsic and extrinsic parameters.NoteBy 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.	8								
calib3d_projectPoints_Mat	CVAPI(ExceptionStatus) calib3d_projectPoints_Mat(cv::Mat *objectPoints, cv::Mat *rvec, cv::Mat *tvec, cv::Mat *cameraMatrix, cv::Mat *distCoeffs, cv::Mat *imagePoints, cv::Mat *jacobian, double aspectRatio) { BEGIN_WRAP cv::projectPoints(*objectPoint s, *rvec, *tvec, *cameraMatrix, *distCoeffs, *imagePoints, *jacobian, aspectRatio); END_WRAP }	cv::projectPoints()	group__calib3d.html	Projects 3D points to an image plane.	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 (Rodrigues) that, together with tvec, performs a change of basis from world to camera coordinate system, see 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 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 calibrateCamera, solvePnP, and stereoCalibrate. The function itself can also be used to compute a re-projection error, given the current intrinsic and extrinsic parameters.NoteBy 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.	8								

calib3d_solvePnP_InputArray	CVAPI(ExceptionStatus) calib3d_solvePnP_InputArray(cv::_InputArray *objectPoints, cv::_InputArray *imagePoints, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_OutputArray *rvec, cv::_OutputArray *tvec, int useExtrinsicGuess, int flags) { BEGIN_WRAP cv::solvePnP(*objectPoints, *imagePoints, *cameraMatrix, entity(distCoeffs), *rvec, *tvec, useExtrinsicGuess != 0, flags); END_WRAP }	cv::solvePnP()	group__calib3d.html	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:	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 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: SOLVEPNP_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection	P3P methods (SOLVEPNP_P3P, SOLVEPNP_AP3P): need 4 input points to return a unique solution. SOLVEPNP_IPPE Input points must be >= 4 and object points must be coplanar. 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.The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera	8								
calib3d_solvePnP_vector	CVAPI(ExceptionStatus) calib3d_solvePnP_vector(cv::Point3f *objectPoints, int objectPointsLength, cv::Point2f *imagePoints, int imagePointsLength, double *cameraMatrix, double *distCoeffs, int distCoeffsLength, double *rvec, double *tvec, int useExtrinsicGuess, int flags) { BEGIN_WRAP const cv::Mat objectPointsMat(objectPointsLength, 1, CV_32FC3, objectPoints); const cv::Mat imagePointsMat(imagePointsLength, 1, CV_32FC2, imagePoints); cv::Mat distCoeffsMat; if (distCoeffs != nullptr) distCoeffsMat = cv::Mat(distCoeffsLength, 1, CV_64FC1, distCoeffs);	cv::solvePnP()	group__calib3d.html	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:	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 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: SOLVEPNP_ITERATIVE Iterative method is based on a Levenberg-Marquardt optimization. In this case the function finds such a pose that minimizes reprojection	P3P methods (SOLVEPNP_P3P, SOLVEPNP_AP3P): need 4 input points to return a unique solution. SOLVEPNP_IPPE Input points must be >= 4 and object points must be coplanar. 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.The function estimates the object pose given a set of object points, their corresponding image projections, as well as the camera	12								
calib3d_solvePnPRansac_InputArray	CVAPI(ExceptionStatus) calib3d_solvePnPRansac_InputArray(cv::_InputArray *objectPoints, cv::_InputArray *imagePoints, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_OutputArray *rvec, cv::_OutputArray *tvec, bool useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, cv::_OutputArray *inliers, int flags) { BEGIN_WRAP cv::solvePnPRansac(*objectPoints, *imagePoints, *cameraMatrix, entity(distCoeffs), *rvec, *tvec,	cv::solvePnPRansac()	group__calib3d.html	Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.	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 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. 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	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 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.	12								
calib3d_solvePnPRansac_vector	CVAPI(ExceptionStatus) calib3d_solvePnPRansac_vector(cv::Point3f *objectPoints, int objectPointsLength, cv::Point2f *imagePoints, int imagePointsLength, double *cameraMatrix, double *distCoeffs, int distCoeffsLength, double *rvec, double *tvec, int useExtrinsicGuess, int iterationsCount, float reprojectionError, double confidence, std::vector<int> *inliers, int flags) { BEGIN_WRAP const cv::Mat objectPointsMat(objectPointsLength, 1, CV_64FC3, objectPoints); const cv::Mat imagePointsMat(imagePointsLength, 1, CV_64FC2,	cv::solvePnPRansac()	group__calib3d.html	Finds an object pose from 3D-2D point correspondences using the RANSAC scheme.	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 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. 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	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 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.	16								
calib3d_initCameraMatrix2D_Mat	CVAPI(ExceptionStatus) calib3d_initCameraMatrix2D_Mat(cv::Mat **objectPoints, int objectPointsLength, cv::Mat **imagePoints, int imagePointsLength, MyCvSize imageSize, double aspectRatio, cv::Mat **returnValue) { BEGIN_WRAP std::vector<cv::Mat> objectPointsVec(objectPointsLength); for (auto i = 0; i < objectPointsLength; i++) objectPointsVec[i] = *objectPoints[i]; std::vector<cv::Mat> imagePointsVec(imagePointsLength); for (auto i = 0; i < objectPointsLength; i++) imagePointsVec[i] = *imagePoints[i];	cv::initCameraMatrix2D()	group__calib3d.html	Finds an initial camera intrinsic matrix from 3D-2D point correspondences.	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.	4								
calib3d_initCameraMatrix2D_array	CVAPI(ExceptionStatus) calib3d_initCameraMatrix2D_array(cv::Point3f **objectPoints, int opSize1, int *opSize2, cv::Point2f **imagePoints, int ipSize1, int *ipSize2, MyCvSize imageSize, double aspectRatio, cv::Mat **returnValue) { BEGIN_WRAP std::vector<std::vector<cv::Point3f> > objectPointsVec(opSize1); for (auto i = 0; i < opSize1; i++) objectPointsVec[i] = std::vector<cv::Point3f>(objectPoints[i], objectPoints[i] + opSize2[i]); std::vector<std::vector<cv::Point3f> > imagePointsVec(ipSize1);	cv::initCameraMatrix2D()	group__calib3d.html	Finds an initial camera intrinsic matrix from 3D-2D point correspondences.	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.	4								
calib3d_findChessboardCorners_InputArray	CVAPI(ExceptionStatus) calib3d_findChessboardCorners_InputArray(cv::_InputArray *image, MyCvSize patternSize, cv::_OutputArray *corners, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::findChessboardCorners(*image, cpp(patternSize), *corners, flags) ? 1 : 0; END_WRAP }	cv::findChessboardCorners()	group__calib3d.html	Finds the positions of internal corners of the chessboard.	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 = cv::Size(points_per_row,points_per_colum) = cv::Size(columns,rows)). corners : Output array of detected corners. flags : Various operation flags that can be zero or a combination of the following values: 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). CALIB_CB_NORMALIZE_IMAGE Normalize the image gamma with equalizeHist before applying fixed or adaptive thresholding. 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. 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 cornersMat gray = ...; //source imagevector<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	4								

calib3d_findChessboardCorners_vector	CVAPI(ExceptionStatus) calib3d_findChessboardCorners_vector(cv::_InputArray *image, MyCvSize patternSize, std::vector<cv::Point2f> *corners, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::findChessboardCorners(*image, cpp(patternSize), *corners, flags) ? 1 : 0; END_WRAP }	cv::findChessboardCorners()	group__calib3d.html	Finds the positions of internal corners of the chessboard.	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 = cv::Size(points_per_row,points_per_colum) = cv::Size(columns,rows)). corners : Output array of detected corners. flags : Various operation flags that can be zero or a combination of the following values: 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). CALIB_CB_NORMALIZE_IMAGE Normalize the image gamma with equalizeHist before applying fixed or adaptive thresholding. 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. 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 cornersMat gray = ...; //source imagevector<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	4								
calib3d_checkChessboard	CVAPI(ExceptionStatus) calib3d_checkChessboard(cv::_InputArray *img, MyCvSize size, int *returnValue) { BEGIN_WRAP *returnValue = cv::checkChessboard(*img, cpp(size)) ? 1 : 0; END_WRAP }	cv::checkChessboard()	group__calib3d.html				2									
calib3d_findChessboardCornersSB_OutputArray	CVAPI(ExceptionStatus) calib3d_findChessboardCornersSB_OutputArray(cv::_InputArray *image, MyCvSize patternSize, cv::_OutputArray *corners, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::findChessboardCornersSB(*image, cpp(patternSize), *corners, flags) ? 1 : 0; END_WRAP }	cv::findChessboardCornersSB()	group__calib3d.html	Finds the positions of internal corners of the chessboard using a sector based approach.	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 = cv::Size(points_per_row,points_per_colum) = cv::Size(columns,rows)). corners : Output array of detected corners. flags : Various operation flags that can be zero or a combination of the following values: CALIB_CB_NORMALIZE_IMAGE Normalize the image gamma with equalizeHist before detection. CALIB_CB_EXHAUSTIVE Run an exhaustive search to improve detection rate. CALIB_CB_ACCURACY Up sample input image to improve sub-pixel accuracy due to aliasing effects. CALIB_CB_LARGER The detected pattern is allowed to be larger than patternSize (see description). CALIB_CB_MARKER The detected pattern must have a marker (see description). This should be used if an accurate camera calibration is required. meta : Optional output array of detected corners (CV_8UC1 and size = cv::Size(columns,rows)). Each entry stands for one corner of the pattern and can have one of the following values: 0 = no meta data attached 1 = left-top corner of a black cell 2 = left-top corner of a white cell	The function is analog to findChessboardCorners but uses a localized radon transformation approximated by box filters being more robust to all sort of noise, faster on larger images and is able to directly return the sub-pixel position of the internal chessboard corners. The Method is based on the paper [61] "Accurate Detection and Localization of Checkerboard Corners for Calibration" demonstrating that the returned sub-pixel positions are more accurate than the one returned by cornerSubPix allowing a precise camera calibration for demanding applications.In the case, the flags CALIB_CB_LARGER or CALIB_CB_MARKER are given, the result can be recovered from the optional meta array. Both flags are helpful to use calibration patterns exceeding the field of view of the camera. These oversized patterns allow more accurate calibrations as corners can be utilized, which are as close as possible to the image borders. For a consistent coordinate system across all images, the optional marker (see	4								
calib3d_findChessboardCornersSB_vector	CVAPI(ExceptionStatus) calib3d_findChessboardCornersSB_vector(cv::_InputArray *image, MyCvSize patternSize, std::vector<cv::Point2f> *corners, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::findChessboardCornersSB(*image, cpp(patternSize), *corners, flags) ? 1 : 0; END_WRAP }	cv::findChessboardCornersSB()	group__calib3d.html	Finds the positions of internal corners of the chessboard using a sector based approach.	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 = cv::Size(points_per_row,points_per_colum) = cv::Size(columns,rows)). corners : Output array of detected corners. flags : Various operation flags that can be zero or a combination of the following values: CALIB_CB_NORMALIZE_IMAGE Normalize the image gamma with equalizeHist before detection. CALIB_CB_EXHAUSTIVE Run an exhaustive search to improve detection rate. CALIB_CB_ACCURACY Up sample input image to improve sub-pixel accuracy due to aliasing effects. CALIB_CB_LARGER The detected pattern is allowed to be larger than patternSize (see description). CALIB_CB_MARKER The detected pattern must have a marker (see description). This should be used if an accurate camera calibration is required. meta : Optional output array of detected corners (CV_8UC1 and size = cv::Size(columns,rows)). Each entry stands for one corner of the pattern and can have one of the following values: 0 = no meta data attached 1 = left-top corner of a black cell 2 = left-top corner of a white cell	The function is analog to findChessboardCorners but uses a localized radon transformation approximated by box filters being more robust to all sort of noise, faster on larger images and is able to directly return the sub-pixel position of the internal chessboard corners. The Method is based on the paper [61] "Accurate Detection and Localization of Checkerboard Corners for Calibration" demonstrating that the returned sub-pixel positions are more accurate than the one returned by cornerSubPix allowing a precise camera calibration for demanding applications.In the case, the flags CALIB_CB_LARGER or CALIB_CB_MARKER are given, the result can be recovered from the optional meta array. Both flags are helpful to use calibration patterns exceeding the field of view of the camera. These oversized patterns allow more accurate calibrations as corners can be utilized, which are as close as possible to the image borders. For a consistent coordinate system across all images, the optional marker (see	4								
calib3d_find4QuadCornerSubpix_InputArray	CVAPI(ExceptionStatus) calib3d_find4QuadCornerSubpix_InputArray(cv::_InputArray *img, cv::_InputOutputArray *corners, MyCvSize regionSize, int *returnValue) { BEGIN_WRAP *returnValue = cv::find4QuadCornerSubpix(*img, *corners, cpp(regionSize)) ? 1 : 0; END_WRAP }	cv::find4QuadCornerSubpix()	group__calib3d.html	finds subpixel-accurate positions of the chessboard corners			3									
calib3d_find4QuadCornerSubpix_vector	CVAPI(ExceptionStatus) calib3d_find4QuadCornerSubpix_vector(cv::_InputArray *img, std::vector<cv::Point2f> *corners, MyCvSize regionSize, int *returnValue) { BEGIN_WRAP *returnValue = cv::find4QuadCornerSubpix(*img, *corners, cpp(regionSize)) ? 1 : 0; END_WRAP }	cv::find4QuadCornerSubpix()	group__calib3d.html	finds subpixel-accurate positions of the chessboard corners			3									
calib3d_drawChessboardCorners_InputArray	CVAPI(ExceptionStatus) calib3d_drawChessboardCorners_InputArray(cv::_InputOutputArray *image, MyCvSize patternSize, cv::_InputArray *corners, int patternWasFound) { BEGIN_WRAP cv::drawChessboardCorners(*image, cpp(patternSize), *corners, patternWasFound != 0); END_WRAP }	cv::drawChessboardCorners()	group__calib3d.html	Renders the detected chessboard corners.	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.Examples: samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	4								
calib3d_drawChessboardCorners_array	CVAPI(ExceptionStatus) calib3d_drawChessboardCorners_array(cv::_InputOutputArray *image, MyCvSize patternSize, cv::Point2f *corners, int cornersLength, int patternWasFound) { BEGIN_WRAP const std::vector<cv::Point2f> cornersVec(corners, corners + cornersLength); cv::drawChessboardCorners(*image, cpp(patternSize), cornersVec, patternWasFound != 0); END_WRAP }	cv::drawChessboardCorners()	group__calib3d.html	Renders the detected chessboard corners.	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.Examples: samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	4								
calib3d_drawFrameAxes	CVAPI(ExceptionStatus) calib3d_drawFrameAxes(cv::_InputOutputArray *image, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *rvec, cv::_InputArray *tvec, float length, int thickness) { BEGIN_WRAP cv::drawFrameAxes(*image, *cameraMatrix, *distCoeffs, *rvec, *tvec, length, thickness); END_WRAP }	cv::drawFrameAxes()	group__calib3d.html	Draw axes of the world/object coordinate system from pose estimation.	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 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.	See alsosolvePnPThis 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.Examples: samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	7								
calib3d_findCirclesGrid_InputArray	CVAPI(ExceptionStatus) calib3d_findCirclesGrid_InputArray(cv::_InputArray *image, MyCvSize patternSize, cv::_OutputArray *centers, int flags, cv::FeatureDetector* blobDetector, int *returnValue) { BEGIN_WRAP if (blobDetector == nullptr) { *returnValue = cv::findCirclesGrid(*image, cpp(patternSize), *centers, flags) ? 1 : 0; } else { const cv::Ptr<cv::FeatureDetector> detectorPtr(blobDetector, BlobDetectorDeleter); // don't delete *returnValue =	cv::findCirclesGrid()	group__calib3d.html	Finds centers in the grid of circles.	image : grid view of input circles; it must be an 8-bit grayscale or color image. patternSize : number of circles per row and column (patternSize = Size(points_per_row, points_per_column)). centers : output array of detected centers. flags : various operation flags that can be one of the following values: CALIB_CB_SYMMETRIC_GRID uses symmetric pattern of circles. CALIB_CB_ASYMMETRIC_GRID uses asymmetric pattern of circles. CALIB_CB_CLUSTERING uses a special algorithm for grid detection. It is more robust to perspective distortions but much more sensitive to background clutter. blobDetector : feature detector that finds blobs like dark circles on light background. If blobDetector is NULL then image represents Point2f array of candidates. parameters : struct for finding circles in a grid pattern.	The function attempts to determine whether the input image contains a grid of circles. If it is, the function locates centers of the circles. The function returns a non-zero value if all of the centers have been found and they have been 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.Sample usage of detecting and drawing the centers of circles: Size patternsize(7,7); //number of centersMat gray = ...; //source imagevector<Point2f> centers; //this will be filled by the detected centersbool patternfound = findCirclesGrid(gray, patternsize, centers);drawChessboardCorners(img, patternsize, Mat(centers), patternfound);fragmentNotoThe 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.	9								

calib3d_findCirclesGrid_vector	CVAPI(ExceptionStatus) calib3d_findCirclesGrid_vector (cv::_InputArray *image, MyCvSize patternSize, std::vector<cv::Point2f> *centers, int flags, cv::FeatureDetector* blobDetector, int *returnValue) { BEGIN_WRAP if (blobDetector == nullptr) { *returnValue = cv::findCirclesGrid(*image, cpp(patternSize), *centers, flags) ? 1 : 0; } else { const cv::Ptr<cv::FeatureDetector> detectorPtr(blobDetector, BlobDetectorDeleter); // don't delete	cv::findCirclesGrid()	group__calib3d.html	Finds centers in the grid of circles.	image : grid view of input circles; it must be an 8-bit grayscale or color image. patternSize : number of circles per row and column (patternSize = Size(points_per_row, points_per_column)). centers : output array of detected centers. flags : various operation flags that can be one of the following values: CALIB_CB_SYMMETRIC_GRID uses symmetric pattern of circles. CALIB_CB_ASYMMETRIC_GRID uses asymmetric pattern of circles. CALIB_CB_CLUSTERING uses a special algorithm for grid detection. It is more robust to perspective distortions but much more sensitive to background clutter. blobDetector : feature detector that finds blobs like dark circles on light background. If blobDetector is NULL then image represents Point2f array of candidates. parameters : struct for finding circles in a grid pattern.	The function attempts to determine whether the input image contains a grid of circles. If it is, the function locates centers of the circles. The function returns a non-zero value if all of the centers have been found and they have been 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.Sample usage of detecting and drawing the centers of circles: Size patternsize(7,7); //number of centersMat gray = ...; //source imagevector<Point2f> centers; //this will be filled by the detected centersbool patternfound = findCirclesGrid(gray, patternsize, centers);drawChessboardCorners(img, patternsize, Mat(centers), patternfound);fragmentNoteThe 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.	9								
calib3d_calibrateCamera_InputArray	CVAPI(ExceptionStatus) calib3d_calibrateCamera_InputArray(cv::Mat **objectPoints, int objectPointsSize, cv::Mat **imagePoints, int imagePointsSize, MyCvSize imageSize, cv::_InputOutputArray *cameraMatrix, cv::_InputOutputArray *distCoeffs, std::vector<cv::Mat> *rvecs, std::vector<cv::Mat> *tvecs, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP std::vector<cv::Mat> objectPointsVec(objectPointsSize); for (auto i = 0; i < objectPointsSize; i++) objectPointsVec[i] =	cv::calibrateCamera()	group__calib3d.html	Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.	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	The function estimates the intrinsic camera parameters and extrinsic parameters for each of the views. The algorithm is based on [293] and [31] . 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 findChessboardCorners). Currently, initialization of intrinsic parameters (when 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	9								
calib3d_calibrateCamera_vector	CVAPI(ExceptionStatus) calib3d_calibrateCamera_vector(cv::Point3f **objectPoints, int opSize1, int *opSize2, cv::Point2f * *imagePoints, int ipSize1, int *ipSize2, MyCvSize imageSize, double *cameraMatrix, double *distCoeffs, int distCoeffsSize, std::vector<cv::Mat> *rvecs, std::vector<cv::Mat> *tvecs, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP std::vector<std::vector<cv::Point3f> > objectPointsVec(opSize1); for (auto i = 0; i < opSize1; i++) objectPointsVec[i] =	cv::calibrateCamera()	group__calib3d.html	Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.	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	The function estimates the intrinsic camera parameters and extrinsic parameters for each of the views. The algorithm is based on [293] and [31] . 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 findChessboardCorners). Currently, initialization of intrinsic parameters (when 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	9								
calib3d_calibrationMatrixValues_InputArray	CVAPI(ExceptionStatus) calib3d_calibrationMatrixValues_InputArray(cv::_InputArray *cameraMatrix, MyCvSize imageSize, double apertureWidth, double apertureHeight, double *fovx, double *fovy, double *focalLength, cv::Point2d *principalPoint, double *aspectRatio) { BEGIN_WRAP double fovx0, fovy0, focalLength0, aspectRatio0; cv::Point2d principalPoint0; cv::calibrationMatrixValues(*cameraMatrix, cpp(imageSize), apertureWidth, apertureHeight, fovx0, fovy0, focalLength0, principalPoint0, aspectRatio0); *fovx = fovx0; *fovy = fovy0;	cv::calibrationMatrixValues()	group__calib3d.html	Computes useful camera characteristics from the camera intrinsic matrix.	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.NoteDo 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).	9								
calib3d_calibrationMatrixValues_array	CVAPI(ExceptionStatus) calib3d_calibrationMatrixValues_array(double *cameraMatrix, MyCvSize imageSize, double apertureWidth, double apertureHeight, double *fovx, double *fovy, double *focalLength, cv::Point2d *principalPoint, double *aspectRatio) { BEGIN_WRAP const cv::Mat cameraMatrixM(3, 3, CV_64FC1, cameraMatrix); cv::_InputArray cameraMatrix1(cameraMatrixM); calib3d_calibrationMatrixValues_InputArray(&cameraMatrix1, imageSize, apertureWidth, apertureHeight, fovx, fovy, focalLength, principalPoint, aspectRatio);	cv::calibrationMatrixValues()	group__calib3d.html	Computes useful camera characteristics from the camera intrinsic matrix.	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.NoteDo 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).	9								
calib3d_stereoCalibrate_InputArray	CVAPI(ExceptionStatus) calib3d_stereoCalibrate_InputArray(cv::_InputArray **objectPoints, int opSize, cv::_InputArray **imagePoints1, int ip1Size, cv::_InputArray **imagePoints2, int ip2Size, cv::_InputOutputArray *cameraMatrix1, cv::_InputOutputArray *distCoeffs1, cv::_InputOutputArray *cameraMatrix2, cv::_InputOutputArray *distCoeffs2, MyCvSize imageSize, cv::_OutputArray *R, cv::_OutputArray *T, cv::_OutputArray *E, cv::_OutputArray *F, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP	cv::stereoCalibrate()	group__calib3d.html	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.	objectPoints : Vector of vectors of the calibration pattern points. The same structure as in 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 calibrateCamera. imagePoints2 : Vector of vectors of the projections of the calibration pattern points, observed by the second camera. The same structure as in calibrateCamera. cameraMatrix1 : Input/output camera intrinsic matrix for the first camera, the same as in calibrateCamera. Furthermore, for the stereo case, additional flags may be used, see below. distCoeffs1 : Input/output vector of distortion coefficients, the same as in 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.	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	14								
calib3d_stereoCalibrate_array	CVAPI(ExceptionStatus) calib3d_stereoCalibrate_array(cv::Point3f **objectPoints, int opSize1, int *opSizes2, cv::Point2f **imagePoints1, int ip1Size1, int *ip1Sizes2, cv::Point2f **imagePoints2, int ip2Size1, int *ip2Sizes2, double *cameraMatrix1, double *distCoeffs1, int dc1Size, double *cameraMatrix2, double *distCoeffs2, int dc2Size, MyCvSize imageSize, cv::_OutputArray *R, cv::_OutputArray *T, cv::_OutputArray *E, cv::_OutputArray *F, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP	cv::stereoCalibrate()	group__calib3d.html	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.	objectPoints : Vector of vectors of the calibration pattern points. The same structure as in 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 calibrateCamera. imagePoints2 : Vector of vectors of the projections of the calibration pattern points, observed by the second camera. The same structure as in calibrateCamera. cameraMatrix1 : Input/output camera intrinsic matrix for the first camera, the same as in calibrateCamera. Furthermore, for the stereo case, additional flags may be used, see below. distCoeffs1 : Input/output vector of distortion coefficients, the same as in 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.	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	14								

calib3d_stereoRectify_InputArray	CVAPI(ExceptionStatus) calib3d_stereoRectify_InputArray(cv::_InputArray *cameraMatrix1, cv::_InputArray *distCoeffs1, cv::_InputArray *cameraMatrix2, cv::_InputArray *distCoeffs2, MyCvSize imageSize, cv::_InputArray *R, cv::_InputArray *T, cv::_OutputArray *R1, cv::_OutputArray *R2, cv::_OutputArray *P1, cv::_OutputArray *P2, cv::_OutputArray *Q, int flags, double alpha, CvSize newImageSize, MyCvRect *validPixROI1, MyCvRect *validPixROI2) { BEGIN_WRAP cv::Rect _validPixROI1, _validPixROI2;	cv::stereoRectify()	group__calib3d.html	Computes rectification transforms for each head of a calibrated stereo camera.	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 stereoCalibrate. T : Translation vector from the coordinate system of the first camera to the second camera, see 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	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: ¥[¥textit{P1} = ¥begin{bmatrix} f & 0 & cx_1 & 0 ¥¥ 0 & f & cy & 0 ¥¥ 0 & 0 & 1 & 0 ¥end{bmatrix}¥]	17								
calib3d_stereoRectify_array	CVAPI(ExceptionStatus) calib3d_stereoRectify_array(double *cameraMatrix1, double *distCoeffs1, int dc1Size, double *cameraMatrix2, double *distCoeffs2, int dc2Size, MyCvSize imageSize, double *R, double *T, double *R1, double *R2, double *P1, double *P2, double *Q, int flags, double alpha, MyCvSize newImageSize, MyCvRect *validPixROI1, MyCvRect *validPixROI2) { BEGIN_WRAP cv::Mat cameraMatrix1M(3, 3, CV_64FC1, cameraMatrix1); cv::Mat cameraMatrix2M(3, 3, CV_64FC1, cameraMatrix2); cv::Mat	cv::stereoRectify()	group__calib3d.html	Computes rectification transforms for each head of a calibrated stereo camera.	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 stereoCalibrate. T : Translation vector from the coordinate system of the first camera to the second camera, see 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	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: ¥[¥textit{P1} = ¥begin{bmatrix} f & 0 & cx_1 & 0 ¥¥ 0 & f & cy & 0 ¥¥ 0 & 0 & 1 & 0 ¥end{bmatrix}¥]	17								
calib3d_stereoRectifyUncalibrated_InputArray	CVAPI(ExceptionStatus) calib3d_stereoRectifyUncalibrated_InputArray(cv::_InputArray *points1, cv::_InputArray *points2, cv::_InputArray *F, MyCvSize imgSize, cv::_OutputArray *H1, cv::_OutputArray *H2, double threshold, int *returnValue) { BEGIN_WRAP *returnValue = cv::stereoRectifyUncalibrated(*points1, *points2, *F, cpp(imgSize), *H1, *H2, threshold) ? 1 : 0; END_WRAP }	cv::stereoRectifyUncalibrated()	group__calib3d.html	Computes a rectification transform for an uncalibrated stereo camera.	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 ¥(|¥textit{points2[i]}^T*¥textit{F}*¥textit{points1[i]}|)>¥textit{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 [104] .NoteWhile 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	7								
calib3d_stereoRectifyUncalibrated_array	CVAPI(ExceptionStatus) calib3d_stereoRectifyUncalibrated_array(cv::Point2d *points1, int points1Size, cv::Point2d *points2, int points2Size, cv::_InputArray *F, MyCvSize imgSize, double *H1, double *H2, double threshold, int *returnValue) { BEGIN_WRAP const cv::Mat points1Mat(points1Size, 1, CV_64FC2, points1); const cv::Mat points2Mat(points2Size, 1, CV_64FC2, points2); cv::Mat H1M(3, 3, CV_64FC1, H1); cv::Mat H2M(3, 3, CV_64FC1, H2); *returnValue = cv::stereoRectifyUncalibrated(points1Mat, points2Mat, *F,	cv::stereoRectifyUncalibrated()	group__calib3d.html	Computes a rectification transform for an uncalibrated stereo camera.	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 ¥(|¥textit{points2[i]}^T*¥textit{F}*¥textit{points1[i]}|)>¥textit{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 [104] .NoteWhile 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 .	7								
calib3d_rectify3Collinear_InputArray	CVAPI(ExceptionStatus) calib3d_rectify3Collinear_InputArray(cv::_InputArray *cameraMatrix1, cv::_InputArray *distCoeffs1, cv::_InputArray *cameraMatrix2, cv::_InputArray *distCoeffs2, cv::_InputArray *cameraMatrix3, cv::_InputArray *distCoeffs3, cv::_InputArray **imgpt1, int imgpt1Size, cv::_InputArray **imgpt2, int imgpt2Size, MyCvSize imageSize, cv::_InputArray *R12, cv::_InputArray *T12, cv::_InputArray *R13, cv::_InputArray *T13, cv::_OutputArray *R1, cv::_OutputArray *R2, cv::_OutputArray *R3, cv::_OutputArray *P1, cv::_OutputArray *P2,	cv::rectify3Collinear()	group__calib3d.html	computes the rectification transformations for 3-head camera, where all the heads are on the same line.			25								
calib3d_getOptimalNewCameraMatrix_InputArr	CVAPI(ExceptionStatus) calib3d_getOptimalNewCameraMatrix_InputArray(cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, MyCvSize imageSize, double alpha, MyCvSize newImgSize, MyCvRect* validPixROI, int centerPrincipalPoint, cv::Mat **returnValue) { BEGIN_WRAP cv::Rect _validPixROI; const auto mat = cv::getOptimalNewCameraMatrix(*cameraMatrix, entby(distCoeffs), cpp(imageSize), alpha, cpp(newImgSize), &_validPixROI, centerPrincipalPoint != 0); *validPixROI = ct_(_validPixROI); *returnValue = new	cv::getOptimalNewCameraMatrix()	group__calib3d.html	Returns the new camera intrinsic matrix based on the free scaling parameter.	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.	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 .	7								
calib3d_getOptimalNewCameraMatrix_array	CVAPI(ExceptionStatus) calib3d_getOptimalNewCameraMatrix_array(double *cameraMatrix, double *distCoeffs, int distCoeffsSize, MyCvSize imageSize, double alpha, MyCvSize newImgSize, MyCvRect* validPixROI, int centerPrincipalPoint, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat cameraMatrixM(3, 3, CV_64FC1, cameraMatrix); const auto distCoeffsM = (distCoeffs == nullptr) ? cv::Mat() : cv::Mat(distCoeffsSize, 1, CV_64FC1, distCoeffs); cv::Rect _validPixROI; const auto mat = cv::getOptimalNewCameraMa	cv::getOptimalNewCameraMatrix()	group__calib3d.html	Returns the new camera intrinsic matrix based on the free scaling parameter.	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.	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 .	7								

calib3d_calibrateHandEye	CVAPI(ExceptionStatus) calib3d_calibrateHandEye(cv::Mat **R_gripper2baseMats, const int32_t R_gripper2baseMatsSize, cv::Mat **t_gripper2baseMats, const int32_t t_gripper2baseMatsSize, cv::Mat **R_target2camMats, const int32_t R_target2camMatsSize, cv::Mat **t_target2camMats, const int32_t t_target2camMatsSize, cv::_OutputArray *R_cam2gripper, cv::_OutputArray *t_cam2gripper, int32_t method) { BEGIN_WRAP std::vector<cv::Mat>	cv::calibrateHandEye()	group__calib3d.html	Computes Hand-Eye calibration: $_{}^{g}\textrm{T}_{c}$.	R_gripper2base : [in] t_gripper2base : [in] R_target2cam : [in] t_target2cam : [in] R_cam2gripper : [out] t_cam2gripper : [out] method : [in]	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 [249] F. Park, B. Martin Robot Sensor Calibration: Solving AX = XB on the Euclidean Group [190] R. Horaud, F. Dornaika Hand-Eye Calibration [112]Another approach consists in estimating simultaneously the rotation and the translation (simultaneous solutions), with the following implemented methods:N. Andreff, R. Horaud, B. Espiau On-line Hand-Eye Calibration [11] K. Daniilidis Hand-Eye Calibration Using Dual Quaternions [52]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	7								
calib3d_calibrateRobotWorldHandEye_OutputA	CVAPI(ExceptionStatus) calib3d_calibrateRobotWorldH andEye_OutputArray(cv::Mat** R_world2camMats, int32_t R_world2camMatsSize, cv::Mat** t_world2camMats, int32_t t_world2camMatsSize, cv::Mat** R_base2gripperMats, int32_t R_base2gripperMatsSize, cv::Mat** t_base2gripperMats, int32_t t_base2gripperMatsSize, cv::_OutputArray* R_base2world, cv::_OutputArray* t_base2world, cv::_OutputArray* R_gripper2cam, cv::_OutputArray* t_gripper2cam, int32_t method) { BEGIN_WRAP	cv::calibrateRobotWorldHandEye()	group__calib3d.html	Computes Robot-World/Hand-Eye calibration: $_{}^{w}\textrm{T}_{b}$ and $_{}^{c}\textrm{T}_{g}$.	R_world2cam : [in] t_world2cam : [in] R_base2gripper : [in] t_base2gripper : [in] R_base2world : [out] t_base2world : [out] R_gripper2cam : [out] t_gripper2cam : [out] method : [in]	The function performs the Robot-World/Hand-Eye calibration using various methods. One approach consists in estimating the rotation then the translation (separable solutions):M. Shah, Solving the robot-world/hand-eye calibration problem using the kronecker product [221]Another approach consists in estimating simultaneously the rotation and the translation (simultaneous solutions), with the following implemented method:A. Li, L. Wang, and D. Wu, Simultaneous robot-world and hand-eye calibration using dual-quaternions and kronecker product [143]The following picture describes the Robot-World/Hand-Eye calibration problem where the transformations between a robot and a world frame and between a robot gripper ("hand") and a camera ("eye") mounted at the robot end-effector have to be estimated.The calibration procedure is the following:a static calibration pattern is used to estimate the transformation between the target frame and the camera frame	9								
calib3d_calibrateRobotWorldHandEye_Pointer	CVAPI(ExceptionStatus) calib3d_calibrateRobotWorldH andEye_Pointer(cv::Mat** R_world2camMats, int32_t R_world2camMatsSize, cv::Mat** t_world2camMats, int32_t t_world2camMatsSize, cv::Mat** R_base2gripperMats, int32_t R_base2gripperMatsSize, cv::Mat** t_base2gripperMats, int32_t t_base2gripperMatsSize, double* R_base2world, double* t_base2world, double* R_gripper2cam, double* t_gripper2cam, int32_t method) { BEGIN_WRAP std::vector<cv::Mat> R_gripper2base; std::vector<cv::Mat> t_gripper2base;	cv::calibrateRobotWorldHandEye()	group__calib3d.html	Computes Robot-World/Hand-Eye calibration: $_{}^{w}\textrm{T}_{b}$ and $_{}^{c}\textrm{T}_{g}$.	R_world2cam : [in] t_world2cam : [in] R_base2gripper : [in] t_base2gripper : [in] R_base2world : [out] t_base2world : [out] R_gripper2cam : [out] t_gripper2cam : [out] method : [in]	The function performs the Robot-World/Hand-Eye calibration using various methods. One approach consists in estimating the rotation then the translation (separable solutions):M. Shah, Solving the robot-world/hand-eye calibration problem using the kronecker product [221]Another approach consists in estimating simultaneously the rotation and the translation (simultaneous solutions), with the following implemented method:A. Li, L. Wang, and D. Wu, Simultaneous robot-world and hand-eye calibration using dual-quaternions and kronecker product [143]The following picture describes the Robot-World/Hand-Eye calibration problem where the transformations between a robot and a world frame and between a robot gripper ("hand") and a camera ("eye") mounted at the robot end-effector have to be estimated.The calibration procedure is the following:a static calibration pattern is used to estimate the transformation between the target frame and the camera frame	17								
calib3d_convertPointsToHomogeneous_InputAr	CVAPI(ExceptionStatus) calib3d_convertPointsToHomo geneous_InputArray(cv::_Inp utArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::convertPointsToHomogen eous(*src, *dst); END_WRAP }	cv::convertPointsToHomogeneous()	group__calib3d.html	Converts points from Euclidean to homogeneous space.	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).	2								
calib3d_convertPointsToHomogeneous_array1	CVAPI(ExceptionStatus) calib3d_convertPointsToHomo geneous_array1(cv::Vec2f *src, cv::Vec3f *dst, int length) { BEGIN_WRAP const cv::Mat srcMat(length, 1, CV_64FC2, src); cv::Mat dstMat(length, 1, CV_64FC3, dst); cv::convertPointsFromHomog eneous(srcMat, dstMat); END_WRAP }	cv::convertPointsFromHomogeneous()	group__calib3d.html	Converts points from homogeneous to Euclidean space.	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,...).	2								
calib3d_convertPointsToHomogeneous_array2	CVAPI(ExceptionStatus) calib3d_convertPointsToHomo geneous_array2(cv::Vec3f *src, cv::Vec4f *dst, int length) { BEGIN_WRAP const cv::Mat srcMat(length, 1, CV_64FC3, src); cv::Mat dstMat(length, 1, CV_64FC4, dst); cv::convertPointsFromHomog eneous(srcMat, dstMat); END_WRAP }	cv::convertPointsFromHomogeneous()	group__calib3d.html	Converts points from homogeneous to Euclidean space.	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,...).	2								
calib3d_convertPointsFromHomogeneous_Inpu	CVAPI(ExceptionStatus) calib3d_convertPointsFromHo mogeneous_InputArray(cv::_I nputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::convertPointsFromHomog eneous(*src, *dst); END_WRAP }	cv::convertPointsFromHomogeneous()	group__calib3d.html	Converts points from homogeneous to Euclidean space.	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,...).	2								
calib3d_convertPointsFromHomogeneous_array	CVAPI(ExceptionStatus) calib3d_convertPointsFromHo mogeneous_array1(cv::Vec3f *src, cv::Vec2f *dst, int length) { BEGIN_WRAP const cv::Mat srcMat(length, 1, CV_64FC3, src); cv::Mat dstMat(length, 1, CV_64FC2, dst); cv::convertPointsFromHomog eneous(srcMat, dstMat); END_WRAP }	cv::convertPointsFromHomogeneous()	group__calib3d.html	Converts points from homogeneous to Euclidean space.	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,...).	2								
calib3d_convertPointsFromHomogeneous_arra	CVAPI(ExceptionStatus) calib3d_convertPointsFromHo mogeneous_array2(cv::Vec4f *src, cv::Vec3f *dst, int length) { BEGIN_WRAP const cv::Mat srcMat(length, 1, CV_64FC4, src); cv::Mat dstMat(length, 1, CV_64FC3, dst); cv::convertPointsFromHomog eneous(srcMat, dstMat); END_WRAP }	cv::convertPointsFromHomogeneous()	group__calib3d.html	Converts points from homogeneous to Euclidean space.	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,...).	2								
calib3d_convertPointsHomogeneous	CVAPI(ExceptionStatus) calib3d_convertPointsHomoge neous(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::convertPointsHomogeneo us(*src, *dst); END_WRAP }	cv::convertPointsHomogeneous()	group__calib3d.html	Converts points to/from homogeneous coordinates.	src : Input array or vector of 2D, 3D, or 4D points. dst : Output vector of 2D, 3D, or 4D points.	The function converts 2D or 3D points from/to homogeneous coordinates by calling either convertPointsToHomogeneous or convertPointsFromHomogeneous.NoteThe function is obsolete. Use one of the previous two functions instead.	2								

calib3d_findFundamentalMat_InputArray	CVAPI(ExceptionStatus) calib3d_findFundamentalMat_InputArray(cv::_InputArray *points1, cv::_InputArray *points2, int method, double param1, double param2, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::findFundamentalMat(*points1, *points2, method, param1, param2, entity(mask)); *returnValue = new cv::Mat(mat); END_WRAP }	cv::findFundamentalMat()	group__calib3d.html	Calculates a fundamental matrix from the corresponding points in two images.	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. FM_7POINT for a 7-point algorithm. ¥(N = 7¥) FM_8POINT for an 8-point algorithm. ¥(N ¥ge 8¥) FM_RANSAC for the RANSAC algorithm. ¥(N ¥ge 8¥) 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 : [out] 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 algorithmint point_count = 100;vector<Point2f> points1(point_count);vector<Point2f> points2(point_count);// initialize the points here ...for(int i = 0; i <	6									
calib3d_findFundamentalMat_arrayF64	CVAPI(ExceptionStatus) calib3d_findFundamentalMat_arrayF64(cv::Point2d *points1, int points1Size, cv::Point2d *points2, int points2Size, int method, double param1, double param2, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat points1M(points1Size, 1, CV_64FC2, points1); const cv::Mat points2M(points2Size, 1, CV_64FC2, points2); const auto mat = cv::findFundamentalMat(points1M, points2M, method, param1, param2, entity(mask)); *returnValue = new cv::Mat(mat); END_WRAP }	cv::findFundamentalMat()	group__calib3d.html	Calculates a fundamental matrix from the corresponding points in two images.	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. FM_7POINT for a 7-point algorithm. ¥(N = 7¥) FM_8POINT for an 8-point algorithm. ¥(N ¥ge 8¥) FM_RANSAC for the RANSAC algorithm. ¥(N ¥ge 8¥) 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 : [out] 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 algorithmint point_count = 100;vector<Point2f> points1(point_count);vector<Point2f> points2(point_count);// initialize the points here ...for(int i = 0; i <	6									
calib3d_findFundamentalMat_arrayF32	CVAPI(ExceptionStatus) calib3d_findFundamentalMat_arrayF32(cv::Point2f *points1, int points1Size, cv::Point2f *points2, int points2Size, int method, double param1, double param2, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat points1M(points1Size, 1, CV_32FC2, points1); const cv::Mat points2M(points2Size, 1, CV_32FC2, points2); const auto mat = cv::findFundamentalMat(points1M, points2M, method, param1, param2, entity(mask)); *returnValue = new cv::Mat(mat); END_WRAP }	cv::findFundamentalMat()	group__calib3d.html	Calculates a fundamental matrix from the corresponding points in two images.	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. FM_7POINT for a 7-point algorithm. ¥(N = 7¥) FM_8POINT for an 8-point algorithm. ¥(N ¥ge 8¥) FM_RANSAC for the RANSAC algorithm. ¥(N ¥ge 8¥) 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 : [out] 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 algorithmint point_count = 100;vector<Point2f> points1(point_count);vector<Point2f> points2(point_count);// initialize the points here ...for(int i = 0; i <	6									
calib3d_computeCorrespondEpilines_InputArray	CVAPI(ExceptionStatus) calib3d_computeCorrespondEpilines_InputArray(cv::_InputArray *points, int whichImage, cv::_InputArray *F, cv::_OutputArray *lines) { BEGIN_WRAP cv::computeCorrespondEpilines(*points, whichImage, *F, *lines); END_WRAP }	cv::computeCorrespondEpilines()	group__calib3d.html	For points in an image of a stereo pair, computes the corresponding epilines in the other image.	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¥) .	4								
calib3d_computeCorrespondEpilines_array2d	CVAPI(ExceptionStatus) calib3d_computeCorrespondEpilines_array2d(cv::Point2d *points, int pointsSize, int whichImage, double *F, cv::Point3f *lines) { BEGIN_WRAP const cv::Mat_<cv::Point2d> pointsM(pointsSize, 1, points); const cv::Mat_<double> FM(3, 3, F); cv::Mat_<cv::Point3f> linesM(pointsSize, 1, lines); cv::computeCorrespondEpilines(pointsM, whichImage, FM, linesM); END_WRAP }	cv::computeCorrespondEpilines()	group__calib3d.html	For points in an image of a stereo pair, computes the corresponding epilines in the other image.	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¥) .	4								
calib3d_computeCorrespondEpilines_array3d	CVAPI(ExceptionStatus) calib3d_computeCorrespondEpilines_array3d(cv::Point3d *points, int pointsSize, int whichImage, double *F, cv::Point3f *lines) { BEGIN_WRAP const cv::Mat_<cv::Point3d> pointsM(pointsSize, 1, points); const cv::Mat_<double> FM(3, 3, F); cv::Mat_<cv::Point3f> linesM(pointsSize, 1, lines); cv::computeCorrespondEpilines(pointsM, whichImage, FM, linesM); END_WRAP }	cv::computeCorrespondEpilines()	group__calib3d.html	For points in an image of a stereo pair, computes the corresponding epilines in the other image.	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¥) .	4								
calib3d_triangulatePoints_InputArray	CVAPI(ExceptionStatus) calib3d_triangulatePoints_InputArray(cv::_InputArray *projMatr1, cv::_InputArray *projMatr2, cv::_InputArray *projPoints1, cv::_InputArray *projPoints2, cv::_OutputArray *points4D) { BEGIN_WRAP cv::triangulatePoints(*projMatr1, *projMatr2, *projPoints1, *projPoints2, *points4D); END_WRAP }	cv::triangulatePoints()	group__calib3d.html	This function reconstructs 3-dimensional points (in homogeneous coordinates) by using their observations with a stereo camera.	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.	NoteKeep in mind that all input data should be of float type in order for this function to work. If the projection matrices from stereoRectify are used, then the returned points are represented in the first camera's rectified coordinate system.See alsoreprojectImageTo3D	5								
calib3d_triangulatePoints_array	CVAPI(ExceptionStatus) calib3d_triangulatePoints_array(double *projMatr1, double *projMatr2, cv::Point2d *projPoints1, int projPoints1Size, cv::Point2d *projPoints2, int projPoints2Size, cv::Vec4d *points4D) { BEGIN_WRAP const cv::Mat_<double> projMatr1M(3, 4, projMatr1); const cv::Mat_<double> projMatr2M(3, 4, projMatr2); const cv::Mat_<cv::Point2d> projPoints1M(projPoints1Size, 1, projPoints1); const cv::Mat_<cv::Point2d> projPoints2M(projPoints2Size, 1, projPoints2); cv::Mat_<cv::Vec4d> points4DM(1, projPoints1Size, points4D);	cv::triangulatePoints()	group__calib3d.html	This function reconstructs 3-dimensional points (in homogeneous coordinates) by using their observations with a stereo camera.	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.	NoteKeep in mind that all input data should be of float type in order for this function to work. If the projection matrices from stereoRectify are used, then the returned points are represented in the first camera's rectified coordinate system.See alsoreprojectImageTo3D	5								
calib3d_correctMatches_InputArray	CVAPI(ExceptionStatus) calib3d_correctMatches_InputArray(cv::_InputArray *F, cv::_InputArray *points1, cv::_InputArray *points2, cv::_OutputArray *newPoints1, cv::_OutputArray *newPoints2) { BEGIN_WRAP cv::correctMatches(*F, *points1, *points2, *newPoints1, *newPoints2); END_WRAP }	cv::correctMatches()	group__calib3d.html	Refines coordinates of corresponding points.	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¥) .	5								

calib3d_correctMatches_array	CVAPI(ExceptionStatus) calib3d_correctMatches_array (double *F, cv::Point2d *points1, int points1Size, cv::Point2d *points2, int points2Size, cv::Point2d *newPoints1, cv::Point2d *newPoints2) { BEGIN_WRAP cv::Mat_<double> FM(3, 3, F); cv::Mat_<cv::Point2d> points1M(points1Size, 1, points1); cv::Mat_<cv::Point2d> points2M(points2Size, 1, points2); cv::Mat_<double> points1MM = points1M.reshape(2); cv::Mat_<double> points2MM = points2M.reshape(2); cv::Mat_<cv::Point2d>	cv::correctMatches()	group__calib3d.html	Refines coordinates of corresponding points.	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¥) .	5								
calib3d_filterSpeckles	CVAPI(ExceptionStatus) calib3d_filterSpeckles(cv::_InputOutputArray *img, double newVal, int maxSpeckleSize, double maxDiff, cv::_InputOutputArray *buf) { BEGIN_WRAP cv::filterSpeckles(*img, newVal, maxSpeckleSize, maxDiff, ptr0(buf)); END_WRAP }	cv::filterSpeckles()	group__calib3d.html	Filters off small noise blobs (speckles) in the disparity map.	img : The input 16-bit signed disparity image newVal : The disparity value used to paint-off the speckles maxSpeckleSize : The maximum speckle size to consider it a speckle. Larger blobs are not affected by the algorithm maxDiff : 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.		5								
calib3d_getValidDisparityROI	CVAPI(ExceptionStatus) calib3d_getValidDisparityROI(MyCvRect roi1, MyCvRect roi2, int minDisparity, int numberOfDisparities, int SADWindowSize, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c(cv::getValidDisparityROI(cpp(roi1), cpp(roi2), minDisparity, numberOfDisparities, SADWindowSize)); END_WRAP }	cv::getValidDisparityROI()	group__calib3d.html	computes valid disparity ROI from the valid ROIs of the rectified images (that are returned by stereoRectify)			5								
calib3d_validateDisparity	CVAPI(ExceptionStatus) calib3d_validateDisparity(cv::_InputOutputArray *disparity, cv::_InputArray *cost, int minDisparity, int numberOfDisparities, int displ2MaxDisp) { BEGIN_WRAP cv::validateDisparity(*disparity, *cost, minDisparity, numberOfDisparities, displ2MaxDisp); END_WRAP }	cv::validateDisparity()	group__calib3d.html	validates disparity using the left-right check. The matrix "cost" should be computed by the stereo correspondence algorithm			5								
calib3d_reprojectImageTo3D	CVAPI(ExceptionStatus) calib3d_reprojectImageTo3D(cv::_InputArray *disparity, cv::_OutputArray *_3dImage, cv::_InputArray *Q, int handleMissingValues, int ddepth) { BEGIN_WRAP cv::reprojectImageTo3D(*disparity, *_3dImage, *Q, handleMissingValues != 0, ddepth); END_WRAP }	cv::reprojectImageTo3D()	group__calib3d.html	Reprojects a disparity image to 3D space.	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 StereoBM or 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 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 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 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 ¥¥ ¥textt{disparity} (x,y) ¥¥ z ¥end{bmatrix}.¥]See alsoTo reproject a sparse set of points {(x,y,d),...} to 3D space, use perspectiveTransform.	5								
calib3d_estimateAffine3D	CVAPI(ExceptionStatus) calib3d_estimateAffine3D(cv::_InputArray *src, cv::_InputArray *dst, cv::_OutputArray *out, cv::_OutputArray *inliers, double ransacThreshold, double confidence, int *returnValue) { BEGIN_WRAP *returnValue = cv::estimateAffine3D(*src, *dst, *out, *inliers, ransacThreshold, confidence); END_WRAP }	cv::estimateAffine3D()	group__calib3d.html	Computes an optimal affine transformation between two 3D point sets.	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.	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} ¥]The function estimates an optimal 3D affine transformation between two 3D point sets using the RANSAC algorithm.	6								
calib3d_sampsonDistance_InputArray	CVAPI(ExceptionStatus) calib3d_sampsonDistance_InputArray(cv::_InputArray *pt1, cv::_InputArray *pt2, cv::_InputArray *F, double *returnValue) { BEGIN_WRAP *returnValue = cv::sampsonDistance(*pt1, *pt2, *F); END_WRAP }	cv::sampsonDistance()	group__calib3d.html	Calculates the Sampson Distance between two points.	pt1 : first homogeneous 2d point pt2 : second homogeneous 2d point F : fundamental matrix	The function cv::sampsonDistance calculates and returns the first order approximation of the geometric error as:¥[sd(¥textit{pt1} , ¥textit{pt2})= ¥frac{(¥textit{pt2}^t ¥cdot ¥textit{F} ¥cdot ¥textit{pt1})^2}{((¥textit{F} ¥cdot ¥textit{pt1})(0))^2 + ((¥textit{F} ¥cdot ¥textit{pt1})(1))^2 + ((¥textit{F}^t ¥cdot ¥textit{pt2})(0))^2 + ((¥textit{F}^t ¥cdot ¥textit{pt2})(1))^2} ¥]The fundamental matrix may be calculated using the findFundamentalMat function. See [103] 11.4.3 for details.	3								
calib3d_sampsonDistance_Point3d	CVAPI(ExceptionStatus) calib3d_sampsonDistance_Point3d(MyCvPoint3D64f pt1, MyCvPoint3D64f pt2, MyCvPoint3D64f *F, double *returnValue) { BEGIN_WRAP std::vector<cv::Point3d> f(9); for (size_t i = 0; i < 9; i++) { f[i] = cpp(F[i]); } *returnValue = cv::sampsonDistance(cv::Mat(cpp(pt1)), cv::Mat(cpp(pt2)), f); END_WRAP }	cv::sampsonDistance()	group__calib3d.html	Calculates the Sampson Distance between two points.	pt1 : first homogeneous 2d point pt2 : second homogeneous 2d point F : fundamental matrix	The function cv::sampsonDistance calculates and returns the first order approximation of the geometric error as:¥[sd(¥textit{pt1} , ¥textit{pt2})= ¥frac{(¥textit{pt2}^t ¥cdot ¥textit{F} ¥cdot ¥textit{pt1})^2}{((¥textit{F} ¥cdot ¥textit{pt1})(0))^2 + ((¥textit{F} ¥cdot ¥textit{pt1})(1))^2 + ((¥textit{F}^t ¥cdot ¥textit{pt2})(0))^2 + ((¥textit{F}^t ¥cdot ¥textit{pt2})(1))^2} ¥]The fundamental matrix may be calculated using the findFundamentalMat function. See [103] 11.4.3 for details.	3								
calib3d_estimateAffine2D	CVAPI(ExceptionStatus) calib3d_estimateAffine2D(cv::_InputArray *from, cv::_InputArray *to, cv::_OutputArray *inliers, int method, double ransacReprojThreshold, uint64_t maxIters, double confidence, uint64_t refineIters, cv::Mat **returnValue) { BEGIN_WRAP const auto result = cv::estimateAffine2D(*from, *to, entity(inliers), method, ransacReprojThreshold, static_cast<size_t>(maxIters), confidence, static_cast<size_t>(refineIters)); *returnValue = new cv::Mat(result); END_WRAP }	cv::estimateAffine2D()															

calib3d_estimateAffinePartial2D	calib3d_estimateAffinePartial2D(cv::_InputArray *from, cv::_InputArray *to, cv::_OutputArray *inliers, int method, double ransacReprojThreshold, uint64_t maxIters, double confidence, uint64_t refineIters, cv::Mat **returnValue) { BEGIN_WRAP const auto result = cv::estimateAffinePartial2D(*from, *to, entity(inliers), method, ransacReprojThreshold, static_cast<size_t>(maxIters), confidence, static_cast<size_t>(refineIters)); *returnValue = new cv::Mat(result); END_WRAP }	cv::estimateAffinePartial2D()																
calib3d_decomposeHomographyMat	CVAPI(ExceptionStatus) calib3d_decomposeHomographyMat(cv::_InputArray *H, cv::_InputArray *K, std::vector<cv::Mat> *rotations, std::vector<cv::Mat> *translations, std::vector<cv::Mat> *normals, int *returnValue) { BEGIN_WRAP *returnValue = cv::decomposeHomographyMat(*H, *K, *rotations, *translations, *normals); END_WRAP }	cv::decomposeHomographyMat()	group__calib3d.html	Decompose a homography matrix to rotation(s), translation(s) and plane normal(s).	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 [160].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.Examples:	5								
calib3d_filterHomographyDecompByVisibleRefpoints	CVAPI(ExceptionStatus) calib3d_filterHomographyDecompByVisibleRefpoints(std::vector<cv::Mat> *rotations, std::vector<cv::Mat> *normals, cv::_InputArray *beforePoints, cv::_InputArray *afterPoints, cv::_OutputArray *possibleSolutions, cv::_InputArray *pointsMask) { BEGIN_WRAP cv::filterHomographyDecompByVisibleRefpoints(*rotations, *normals, *beforePoints, *afterPoints, *possibleSolutions, entity(pointsMask)); END_WRAP }	cv::filterHomographyDecompByVisibleRefpoints()	group__calib3d.html	Filters homography decompositions based on additional information.	rotations : Vector of rotation matrices. normals : Vector of plane normal matrices. beforePoints : Vector of (rectified) visible reference points before the homography is applied afterPoints : Vector of (rectified) visible reference points after the homography is applied possibleSolutions : Vector of int indices representing the viable solution set after filtering pointsMask : 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 [160] . 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.	6								
calib3d_undistort	CVAPI(ExceptionStatus) calib3d_undistort(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *newCameraMatrix) { BEGIN_WRAP cv::undistort(*src, *dst, *cameraMatrix, entity(distCoeffs), entity(newCameraMatrix)); END_WRAP }	cv::undistort()	group__calib3d.html	Transforms an image to compensate for lens distortion.	src : Input (distorted) image. dst : Output (corrected) image that has the same size and type as src . cameraMatrix : Input camera matrix ¥(A = ¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) . distCoeffs : Input vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, ¥tau_x, ¥tau_y]]]])¥) of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. newCameraMatrix : Camera matrix of the distorted image. By default, it is the same as cameraMatrix 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 initUndistortRectifyMap (with unity R) and remap (with bilinear interpolation). See the former function for details of the transformation being performed.Those pixels in the destination image, for which there is no correspondent pixels in the source image, are filled with zeros (black color).A particular subset of the source image that will be visible in the corrected image can be regulated by newCameraMatrix. You can use getOptimalNewCameraMatrix to compute the appropriate newCameraMatrix depending on your requirements.The camera matrix and the distortion parameters can be determined using calibrateCamera. If the resolution of images is different from the resolution used at the calibration stage, ¥(f_x, f_y, c_x¥) and ¥(c_y¥) need to be scaled accordingly, while the distortion coefficients remain the same.	5								
calib3d_initUndistortRectifyMap	CVAPI(ExceptionStatus) calib3d_initUndistortRectifyMap(cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *R, cv::_InputArray *newCameraMatrix, MyCvSize size, int m1type, cv::_OutputArray *map1, cv::_OutputArray *map2) { BEGIN_WRAP cv::initUndistortRectifyMap(*cameraMatrix, *distCoeffs, *R, *newCameraMatrix, cpp(size), m1type, *map1, *map2); END_WRAP }	cv::initUndistortRectifyMap()	group__calib3d.html	Computes the undistortion and rectification transformation map.	cameraMatrix : Input camera matrix ¥(A=¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) . distCoeffs : Input vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, ¥tau_x, ¥tau_y]]]])¥) of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. R : Optional rectification transformation in the object space (3x3 matrix). R1 or R2 , computed by stereoRectify can be passed here. If the matrix is empty, the identity transformation is assumed. In cv2nitUndistortMap R assumed to be an identity matrix. newCameraMatrix : New camera matrix ¥(A'=¥vecthreethree{f_x'}{0}{c_x'}{0}{f_y'}{c_y'}{0}{0}{1}¥). size : Undistorted image size. m1type : Type of the first output map that can be CV_32FC1, CV_32FC2 or CV_16SC2, see convertMaps map1 : The first output map. map2 : The second output map.	The function computes the joint undistortion and rectification transformation and represents the result in the form of maps for remap. The undistorted image looks like original, as if it is captured with a camera using the camera matrix =newCameraMatrix and zero distortion. In case of a monocular camera, newCameraMatrix is usually equal to cameraMatrix, or it can be computed by getOptimalNewCameraMatrix for a better control over scaling. In case of a stereo camera, newCameraMatrix is normally set to P1 or P2 computed by stereoRectify .Also, this new camera is oriented differently in the coordinate space, according to R. That, for example, helps to align two heads of a stereo camera so that the epipolar lines on both images become horizontal and have the same y-coordinate (in case of a horizontally aligned stereo camera).The function actually builds the maps for the inverse mapping algorithm that is used by remap. That is, for each pixel ¥((u, v)¥) in the destination (corrected and rectified)	8								
calib3d_initWideAngleProjMap	CVAPI(ExceptionStatus) calib3d_initWideAngleProjMap(cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, MyCvSize imageSize, int destImageWidth, int m1type, cv::_OutputArray *map1, cv::_OutputArray *map2, int projType, double alpha, float *returnValue) { BEGIN_WRAP *returnValue = cv::initWideAngleProjMap(*cameraMatrix, *distCoeffs, cpp(imageSize), destImageWidth, m1type, *map1, *map2, static_cast<cv::UndistortTypes>(projType), alpha); END_WRAP }	cv::initWideAngleProjMap()	group__calib3d.html	initializes maps for remap for wide-angle			9								
calib3d_getDefaultNewCameraMatrix	CVAPI(ExceptionStatus) calib3d_getDefaultNewCameraMatrix(cv::_InputArray *cameraMatrix, MyCvSize imgsize, int centerPrincipalPoint, cv::Mat **returnValue) { BEGIN_WRAP const auto result = cv::getDefaultNewCameraMatrix(*cameraMatrix, cpp(imgsize), centerPrincipalPoint != 0); *returnValue = new cv::Mat(result); END_WRAP }	cv::getDefaultNewCameraMatrix()	group__calib3d.html	Returns the default new camera matrix.	cameraMatrix : Input camera matrix. imgsize : Camera view image size in pixels. centerPrincipalPoint : Location of the principal point in the new camera matrix. The parameter indicates whether this location should be at the image center or not.	The function returns the camera matrix that is either an exact copy of the input cameraMatrix (when centerPrinicipalPoint=false), or the modified one (when centerPrincipalPoint=true).In the latter case, the new camera matrix will be:¥[¥begin{bmatrix} f_x && 0 && (¥texttt{imgSize.width} -1)*0.5 ¥¥ 0 && f_y && (¥texttt{imgSize.height} -1)*0.5 ¥¥ 0 && 0 && 1 ¥end{bmatrix} ,¥]where ¥(f_x¥) and ¥(f_y¥) are ¥((0,0¥) and ¥((1,1¥) elements of cameraMatrix, respectively.By default, the undistortion functions in OpenCV (see initUndistortRectifyMap, undistort) do not move the principal point. However, when you work with stereo, it is important to move the principal points in both views to the same y-coordinate (which is required by most of stereo correspondence algorithms), and may be to the same x-coordinate too. So, you can form the new camera matrix for each view where the	3								
calib3d_undistortPoints	CVAPI(ExceptionStatus) calib3d_undistortPoints(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *R, cv::_InputArray *P) { BEGIN_WRAP cv::undistortPoints(*src, *dst, *cameraMatrix, *distCoeffs, entity(R), entity(P)); END_WRAP }	cv::undistortPoints()	group__calib3d.html	Computes the ideal point coordinates from the observed point coordinates.	src : Observed point coordinates, 2xN/Nx2 1-channel or 1xN/Nx1 2-channel (CV_32FC2 or CV_64FC2) (or vector<Point2f>). dst : Output ideal point coordinates (1xN/Nx1 2-channel or vector<Point2f>) after undistortion and reverse perspective transformation. If matrix P is identity or omitted, dst will contain normalized point coordinates. cameraMatrix : Camera matrix ¥(¥vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}¥) . distCoeffs : Input vector of distortion coefficients ¥((k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, ¥tau_x, ¥tau_y]]]])¥) of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. R : Rectification transformation in the object space (3x3 matrix). R1 or R2 computed by stereoRectify can be passed here. If the matrix is empty, the identity transformation is used. P : New camera matrix (3x3) or new projection matrix (3x4) ¥(¥begin{bmatrix} {f'}_x & 0 & {c'}_x & t_x ¥¥ 0 & {f'}_y & {c'}_y & t_y ¥¥ 0 & 0 & 1 & t_z ¥end{bmatrix}¥). P1 or P2 computed by stereoRectify can be passed here. If the matrix is empty, the identity new camera matrix is used.	The function is similar to undistort and initUndistortRectifyMap but it operates on a sparse set of points instead of a raster image. Also the function performs a reverse transformation to projectPoints. In case of a 3D object, it does not reconstruct its 3D coordinates, but for a planar object, it does, up to a translation vector, if the proper R is specified.For each observed point coordinate ¥((u, v)¥) the function computes:¥[¥begin{array}{l} x^{''}_n ¥¥ y^{''}_n ¥end{array} = ¥leftarrow (x ¥¥ y^{''} ¥leftarrow (v - c_y)/f_y ¥¥ (u - c_x)/f_x ¥¥ y^{''} ¥leftarrow (v - c_y)/f_y ¥¥ (x',y') = undistort(x^{''}_n,y^{''}_n, ¥texttt{distCoeffs}) ¥¥ {[X¥,Y¥,W]} ^T ¥leftarrow R*[x' ¥, y' ¥, 1]^T ¥¥ x ¥leftarrow X/W ¥¥ y ¥leftarrow Y/W ¥¥ ¥text{only performed if P is specified:} ¥¥ u' ¥leftarrow x {f'}_x + {c'}_x ¥¥ v' ¥leftarrow y {f'}_y + {c'}_y ¥end{array} ¥]where undistort is an approximate iterative algorithm that estimates the normalized original point coordinates out of the normalized distorted point coordinates ("normalized" means that the	6								

calib3d_undistortPointsIter	CVAPI(ExceptionStatus) calib3d_undistortPointsIter(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *cameraMatrix, cv::_InputArray *distCoeffs, cv::_InputArray *R, cv::_InputArray *P, MyCvTermCriteria criteria) { BEGIN_WRAP cv::undistortPoints(*src, *dst, *cameraMatrix, *distCoeffs, entity(R), entity(P), cpp(criteria)); END_WRAP }	cv::undistortPoints()	group__calib3d.html	Computes the ideal point coordinates from the observed point coordinates.	src : Observed point coordinates, 2xN/Nx2 1-channel or 1xN/Nx1 2-channel (CV_32FC2 or CV_64FC2) (or vector<Point2f>). dst : Output ideal point coordinates (1xN/Nx1 2-channel or vector<Point2f>) after undistortion and reverse perspective transformation. If matrix P is identity or omitted, dst will contain normalized point coordinates. cameraMatrix : Camera matrix $$\vecthreethree{f_x}{0}{c_x}{0}{f_y}{c_y}{0}{0}{1}$$. distCoeffs : Input vector of distortion coefficients $$(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$$ of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. R : Rectification transformation in the object space (3x3 matrix). R1 or R2 computed by stereoRectify can be passed here. If the matrix is empty, the identity transformation is used. P : New camera matrix (3x3) or new projection matrix (3x4) $$\vecthreethree{f'_x}{0}{c'_x}{0}{f'_y}{c'_y}{0}{0}{1}$$. P1 or P2 computed by stereoRectify can be passed here. If the matrix is empty, the identity new camera matrix is used.	The function is similar to undistort and initUndistortRectifyMap but it operates on a sparse set of points instead of a raster image. Also the function performs a reverse transformation to projectPoints. In case of a 3D object, it does not reconstruct its 3D coordinates, but for a planar object, it does, up to a translation vector, if the proper R is specified.For each observed point coordinate $$(u, v)$$ the function computes: $$\begin{array}{l} x^{"} \leftarrow (u - c_x)*f_x + s*y^{"} + t_x \\ y^{"} \leftarrow (v - c_y)*f_y + t_y \\ (x', y') = undistort(x^{"},y^{"}, \texttt{distCoeffs}) \\ [X,Y,W]^T \leftarrow R*[x', y', 1]^T \\ x \leftarrow X/W \\ y \leftarrow Y/W \\ \text{only performed if P is specified:} \\ u' \leftarrow x*f'_x + c'_x \\ v' \leftarrow y*f'_y + c'_y \end{array}$$ where undistort is an approximate iterative algorithm that estimates the normalized original point coordinates out of the normalized distorted point coordinates ("normalized" means that the coordinates do not depend on the camera	7								
calib3d_recoverPose_InputArray1	CVAPI(ExceptionStatus) calib3d_recoverPose_InputArray1(cv::_InputArray *E, cv::_InputArray *points1, cv::_InputArray *points2, cv::_InputArray *cameraMatrix, cv::_OutputArray *R, cv::_OutputArray *t, cv::_InputOutputArray *mask, int *returnValue) { BEGIN_WRAP *returnValue = cv::recoverPose(*E, *points1, *points2, *cameraMatrix, *R, *t, entity(mask)); END_WRAP }	cv::recoverPose()	group__calib3d.html	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.	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 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 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 [186].This function can be used to process the output E and mask from findEssentialMat. In the scenario, points1 and points2 are the same input for findEssentialMat :// Example. Estimation of fundamental matrix using the RANSAC algorithmint 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);fragment	7								
calib3d_recoverPose_InputArray2	CVAPI(ExceptionStatus) calib3d_recoverPose_InputArray2(cv::_InputArray *E, cv::_InputArray *points1, cv::_InputArray *points2, cv::_OutputArray *R, cv::_OutputArray *t, double focal, MyCvPoint2D64f pp, cv::_InputOutputArray *mask, int *returnValue) { BEGIN_WRAP *returnValue = cv::recoverPose(*E, *points1, *points2, *R, *t, focal, cpp(pp), entity(mask)); END_WRAP }	cv::recoverPose()	group__calib3d.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	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 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: $$A = \begin{bmatrix} f & 0 & x_{pp} \\ 0 & f & y_{pp} \\ 0 & 0 & 1 \end{bmatrix}$$	8								
calib3d_recoverPose_InputArray3	CVAPI(ExceptionStatus) calib3d_recoverPose_InputArray3(cv::_InputArray *E, cv::_InputArray *points1, cv::_InputArray *points2, cv::_InputArray *cameraMatrix, double distanceThresh, cv::_OutputArray *R, cv::_OutputArray *t, cv::_InputOutputArray *mask, cv::_OutputArray *triangulatedPoints, int *returnValue) { BEGIN_WRAP *returnValue = cv::recoverPose(*E, *points1, *points2, *cameraMatrix, *R, *t, distanceThresh, entity(mask), entity(triangulatedPoints)); END_WRAP }	cv::recoverPose()	group__calib3d.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	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 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	This function differs from the one above that it outputs the triangulated 3D point that are used for the cheirality check.	9								
calib3d_findEssentialMat_InputArray1	CVAPI(ExceptionStatus) calib3d_findEssentialMat_InputArray1(cv::_InputArray *points1, cv::_InputArray *points2, cv::_InputArray *cameraMatrix, int method, double prob, double threshold, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::findEssentialMat(*points1, *points2, *cameraMatrix, method, prob, threshold, entity(mask)); *returnValue = new cv::Mat(mat); END_WRAP }	cv::findEssentialMat()	group__calib3d.html	Calculates an essential matrix from the corresponding points in two images.	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. RANSAC for the RANSAC algorithm. 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	This function estimates essential matrix based on the five-point algorithm solver in [186] . [228] 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.	7								
calib3d_findEssentialMat_InputArray2	CVAPI(ExceptionStatus) calib3d_findEssentialMat_InputArray2(cv::_InputArray *points1, cv::_InputArray *points2, double focal, MyCvPoint2D64f pp, int method, double prob, double threshold, cv::_OutputArray *mask, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::findEssentialMat(*points1, *points2, focal, cpp(pp), method, prob, threshold, entity(mask)); *returnValue = new cv::Mat(mat); END_WRAP }	cv::findEssentialMat()	group__calib3d.html	Calculates an essential matrix from the corresponding points in two images.	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. RANSAC for the RANSAC algorithm. 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	This function estimates essential matrix based on the five-point algorithm solver in [186] . [228] 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.	8								
calib3d_fisheye_projectPoints2	CVAPI(ExceptionStatus) calib3d_fisheye_projectPoints2(cv::_InputArray *objectPoints, cv::_OutputArray *imagePoints, cv::_InputArray *rvec, cv::_InputArray *tvec, cv::_InputArray *K, cv::_InputArray *D, double alpha, cv::_OutputArray *jacobian) { BEGIN_WRAP cv::fisheye::projectPoints(*ob jectPoints, *imagePoints, *rvec, *tvec, *K, *D, alpha, entity(jacobian)); END_WRAP }	cv::fisheye::projectPoints()	group__calib3d__fisheye.html	Projects points using fisheye model.	objectPoints : Array of object points, 1xN/Nx1 3-channel (or vector<Point3f>), where N is the number of points in the view. imagePoints : Output array of image points, 2xN/Nx2 1-channel or 1xN/Nx1 2-channel, or vector<Point2f>. K : Camera intrinsic matrix $$\cameramatrix{K}$$. D : Input vector of distortion coefficients $$\distortionfisheye$$. alpha : The skew coefficient. jacobian : Optional output 2Nx15 jacobian matrix of derivatives of image points with respect to components of the focal lengths, coordinates of the principal point, distortion coefficients, rotation vector, translation vector, and the skew. In the old interface different components of the jacobian are returned via different output parameters.	The function computes 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.	8								
calib3d_fisheye_distortPoints	CVAPI(ExceptionStatus) calib3d_fisheye_distortPoints(cv::_InputArray *undistorted, cv::_OutputArray *distorted, cv::_InputArray *K, cv::_InputArray *D, double alpha) { BEGIN_WRAP cv::fisheye::distortPoints(*un distorted, *distorted, *K, *D, alpha); END_WRAP }	cv::fisheye::distortPoints()	group__calib3d__fisheye.html	Distorts 2D points using fisheye model.	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 $$\distortionfisheye$$. 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 fisheye::undistortPoints you have to multiply them with $$P^{-1}$$.	5								

calib3d_fisheye_undistortPoints	CVAPI(ExceptionStatus) calib3d_fisheye_undistortPoint s(cv::_InputArray *distorted, cv::_OutputArray *undistorted, cv::_InputArray *K, cv::_InputArray *D, cv::_InputArray *R, cv::_InputArray *P) { BEGIN_WRAP cv::fisheye::undistortPoints(* distorted, *undistorted, *K, *D, entity(R), entity(P)); END_WRAP }	cv::fisheye::undistortPoints()	group__calib3d__fisheye.html	Undistorts 2D points using fisheye model.	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- channel P : New camera intrinsic matrix (3x3) or new projection matrix (3x4) undistorted : Output array of image points, 1xN/Nx1 2-channel, or vector<Point2f> .		6								
calib3d_fisheye_initUndistortRectifyMap	CVAPI(ExceptionStatus) calib3d_fisheye_initUndistortR ectifyMap(cv::_InputArray *K, cv::_InputArray *D, cv::_InputArray *R, cv::_InputArray *P, MyCvSize size, int m1type, cv::_OutputArray *map1, cv::_OutputArray *map2) { BEGIN_WRAP cv::fisheye::initUndistortRecti fyMap(*K, *D, *R, *P, cpp(size), m1type, *map1, *map2); END_WRAP }	cv::fisheye::initUndistortRectifyMap()	group__calib3d__fisheye.html	Computes undistortion and rectification maps for image transform by remap. If D is empty zero distortion is used, if R or P is empty identity matrixes are used.	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- channel P : 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.		8								
calib3d_fisheye_undistortImage	CVAPI(ExceptionStatus) calib3d_fisheye_undistortImag e(cv::_InputArray *distorted, cv::_OutputArray *undistorted, cv::_InputArray *K, cv::_InputArray *D, cv::_InputArray *Knew, MyCvSize newSize) { BEGIN_WRAP cv::fisheye::undistortImage(* distorted, *undistorted, *K, *D, entity(Knew), cpp(newSize)); END_WRAP }	cv::fisheye::undistortImage()	group__calib3d__fisheye.html	Transforms an image to compensate for fisheye lens distortion.	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 lensPictures 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	6								
calib3d_fisheye_estimateNewCameraMatrixFor	CVAPI(ExceptionStatus) calib3d_fisheye_estimateNew CameraMatrixForUndistortRec tify(cv::_InputArray *K, cv::_InputArray *D, MyCvSize image_size, cv::_InputArray *R, cv::_OutputArray *P, double balance, MyCvSize newSize, double fov_scale) { BEGIN_WRAP cv::fisheye::estimateNewCam eraMatrixForUndistortRectify(*K, *D, cpp(image_size), *R, *P, balance, cpp(newSize), fov_scale); END_WRAP }	cv::fisheye::estimateNewCameraMatrixForUndistor tRectify()	group__calib3d__fisheye.html	Estimates new camera intrinsic matrix for undistortion or rectification.	K : Camera intrinsic matrix ¥(cameramatrix{K}¥). image_size : Size of the image 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- channel P : 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 fov_scale : Divisor for new focal length.		8								
calib3d_fisheye_calibrate	calib3d_fisheye_calibrate(std::vector<cv::Mat> *objectPoints, std::vector<cv::Mat> *imagePoints, MyCvSize imageSize, cv::_InputOutputArray *K, cv::_InputOutputArray *D, std::vector<cv::Mat> *rvecs, std::vector<cv::Mat> *tvecs, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP *returnValue = cv::fisheye::calibrate(*objectPoints, *imagePoints, cpp(imageSize), *K, *D, *rvecs, *tvecs, flags, cpp(criteria)); END_WRAP }	cv::fisheye::calibrate()	group__calib3d__fisheye.html	Performs camera calibaration.	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 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		9								
calib3d_fisheye_stereoRectify	CVAPI(ExceptionStatus) calib3d_fisheye_stereoRectify(cv::_InputArray *K1, cv::_InputArray *D1, cv::_InputArray *K2, cv::_InputArray *D2, MyCvSize imageSize, cv::_InputArray *R, cv::_InputArray *tvec, cv::_OutputArray *R1, cv::_OutputArray *R2, cv::_OutputArray *P1, cv::_OutputArray *P2, cv::_OutputArray *Q, int flags, MyCvSize newImageSize, double balance, double fov_scale) { BEGIN_WRAP cv::fisheye::stereoRectify(*K 1, *D1, *K2, *D2, cpp(imageSize), *R, *tvec, *R1, *R2, *P1, *P2, *Q, flags, cpp(newImageSize), balance, fov_scale);	cv::fisheye::stereoRectify()	group__calib3d__fisheye.html	Stereo rectification for fisheye camera model.	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 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		16								
calib3d_fisheye_stereoCalibrate	CVAPI(ExceptionStatus) calib3d_fisheye_stereoCalibrat e(std::vector<cv::Mat> *objectPoints, std::vector<cv::Mat> *imagePoints1, std::vector<cv::Mat> *imagePoints2, cv::_InputOutputArray *K1, cv::_InputOutputArray *D1, cv::_InputOutputArray *K2, cv::_InputOutputArray *D2, MyCvSize imageSize, cv::_OutputArray *R, cv::_OutputArray *T, int flags, MyCvTermCriteria criteria, double *returnValue) { BEGIN_WRAP *returnValue = cv::fisheye::stereoCalibrate(	cv::fisheye::stereoCalibrate()	group__calib3d__fisheye.html	Performs stereo calibration.	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 fisheye::CALIB_USE_INTRINSIC_GUESS , 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		12								
calib3d_StereoMatcher_compute	CVAPI(ExceptionStatus) calib3d_StereoMatcher_comp ute(cv::Ptr<cv::StereoMatcher> *obj, cv::_InputArray *left, cv::_InputArray *right, cv::_OutputArray *disparity) { BEGIN_WRAP (*obj)->compute(*left, *right, *disparity); END_WRAP }	cv::StereoMatcher::compute()	classcv_1_1StereoMatcher.html	Computes disparity map for the specified stereo pair.	left : Left 8-bit single-channel image. right : Right image of the same size and the same type as the left one. disparity : Output disparity map. It has the same size as the input images. Some algorithms, like StereoBM or StereoSGBM compute 16-bit fixed-point disparity map (where each disparity value has 4 fractional bits), whereas other algorithms output 32-bit floating-point disparity map.	Implemented in cv::cuda::StereoSGM.	3								
calib3d_StereoMatcher_getMinDisparity	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getMi nDisparity(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getMinDisparity(); END_WRAP }	cv::StereoMatcher::getMinDisparity()	classcv_1_1StereoMatcher.html			1								
calib3d_StereoMatcher_setMinDisparity	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setMin Disparity(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setMinDisparity(value); END_WRAP }	cv::StereoMatcher::setMinDisparity()	classcv_1_1StereoMatcher.html			1								

calib3d_StereoMatcher_getNumDisparities	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getNumDisparities(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getNumDisparities(); END_WRAP }	cv::StereoMatcher::getNumDisparities()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_setNumDisparities	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setNumDisparities(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setNumDisparities(value); END_WRAP }	cv::StereoMatcher::setNumDisparities()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_getBlockSize	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getBlockSize(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getBlockSize(); END_WRAP }	cv::StereoMatcher::getBlockSize()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_setBlockSize	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setBlockSize(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setBlockSize(value); END_WRAP }	cv::StereoMatcher::setBlockSize()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_getSpeckleWindowSize	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getSpeckleWindowSize(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getSpeckleWindowSize(); END_WRAP }	cv::StereoMatcher::getSpeckleWindowSize()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_setSpeckleWindowSize	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setSpeckleWindowSize(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setSpeckleWindowSize(value); END_WRAP }	cv::StereoMatcher::setSpeckleWindowSize()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_getSpeckleRange	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getSpeckleRange(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getSpeckleRange(); END_WRAP }	cv::StereoMatcher::getSpeckleRange()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_setSpeckleRange	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setSpeckleRange(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setSpeckleRange(value); END_WRAP }	cv::StereoMatcher::setSpeckleRange()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_getDisp12MaxDiff	CVAPI(ExceptionStatus) calib3d_StereoMatcher_getDisp12MaxDiff(cv::Ptr<cv::StereoMatcher> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getDisp12MaxDiff(); END_WRAP }	cv::StereoMatcher::getDisp12MaxDiff()	classcv_1_1StereoMatcher.html				1										
calib3d_StereoMatcher_setDisp12MaxDiff	CVAPI(ExceptionStatus) calib3d_StereoMatcher_setDisp12MaxDiff(cv::Ptr<cv::StereoMatcher> *obj, int value) { BEGIN_WRAP (*obj)- >setDisp12MaxDiff(value); END_WRAP }	cv::StereoMatcher::setDisp12MaxDiff()	classcv_1_1StereoMatcher.html				1										
calib3d_Ptr_StereoBM_delete	CVAPI(ExceptionStatus) calib3d_Ptr_StereoBM_delete(cv::Ptr<cv::StereoBM> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::StereoBM	classcv_1_1StereoBM.html	Class for computing stereo correspondence using the block matching algorithm, introduced and contributed to OpenCV by K. Konolige.									•			
calib3d_Ptr_StereoBM_get	CVAPI(ExceptionStatus) calib3d_Ptr_StereoBM_get(cv::Ptr<cv::StereoBM> *obj, cv::StereoBM **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::StereoBM	classcv_1_1StereoBM.html	Class for computing stereo correspondence using the block matching algorithm, introduced and contributed to OpenCV by K. Konolige.										•		
calib3d_StereoBM_create	CVAPI(ExceptionStatus) calib3d_StereoBM_create(int numDisparities, int blockSize, cv::Ptr<cv::StereoBM> **returnValue) { BEGIN_WRAP const auto obj = cv::StereoBM::create(numDis parities, blockSize); *returnValue = clone(obj); END_WRAP }	cv::StereoBM::create()	classcv_1_1StereoBM.html	Creates StereoBM object.	numDisparities : the disparity search range. For each pixel algorithm will find the best disparity from 0 (default minimum disparity) to numDisparities. The search range can then be shifted by changing the minimum disparity. blockSize : the linear size of the blocks compared by the algorithm. The size should be odd (as the block is centered at the current pixel). Larger block size implies smoother, though less accurate disparity map. Smaller block size gives more detailed disparity map, but there is higher chance for algorithm to find a wrong correspondence.	The function create StereoBM object. You can then call StereoBM::compute() to compute disparity for a specific stereo pair.	2								
calib3d_StereoBM_getPreFilterType	CVAPI(ExceptionStatus) calib3d_StereoBM_getPreFilterType(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getPreFilterType(); END_WRAP }	cv::StereoBM::getPreFilterType()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setPreFilterType	CVAPI(ExceptionStatus) calib3d_StereoBM_setPreFilterType(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)- >setPreFilterType(value); END_WRAP }	cv::StereoBM::setPreFilterType()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getPreFilterSize	CVAPI(ExceptionStatus) calib3d_StereoBM_getPreFilterSize(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)- >getPreFilterSize(); END_WRAP }	cv::StereoBM::getPreFilterSize()	classcv_1_1StereoBM.html				1										

calib3d_StereoBM_setPreFilterSize	CVAPI(ExceptionStatus) calib3d_StereoBM_setPreFilterSize(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)->setPreFilterSize(value); END_WRAP }	cv::StereoBM::setPreFilterSize()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getPreFilterCap	CVAPI(ExceptionStatus) calib3d_StereoBM_getPreFilterCap(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getPreFilterCap(); END_WRAP }	cv::StereoBM::getPreFilterCap()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setPreFilterCap	CVAPI(ExceptionStatus) calib3d_StereoBM_setPreFilterCap(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)->setPreFilterCap(value); END_WRAP }	cv::StereoBM::setPreFilterCap()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getTextureThreshold	CVAPI(ExceptionStatus) calib3d_StereoBM_getTextureThreshold(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getTextureThreshold(); END_WRAP }	cv::StereoBM::getTextureThreshold()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setTextureThreshold	CVAPI(ExceptionStatus) calib3d_StereoBM_setTextureThreshold(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)->setTextureThreshold(value); END_WRAP }	cv::StereoBM::setTextureThreshold()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getUniquenessRatio	CVAPI(ExceptionStatus) calib3d_StereoBM_getUniquenessRatio(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getUniquenessRatio(); END_WRAP }	cv::StereoBM::getUniquenessRatio()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setUniquenessRatio	CVAPI(ExceptionStatus) calib3d_StereoBM_setUniquenessRatio(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)->setUniquenessRatio(value); END_WRAP }	cv::StereoBM::setUniquenessRatio()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getSmallerBlockSize	CVAPI(ExceptionStatus) calib3d_StereoBM_getSmallerBlockSize(cv::Ptr<cv::StereoBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getSmallerBlockSize(); END_WRAP }	cv::StereoBM::getSmallerBlockSize()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setSmallerBlockSize	CVAPI(ExceptionStatus) calib3d_StereoBM_setSmallerBlockSize(cv::Ptr<cv::StereoBM> *obj, int value) { BEGIN_WRAP (*obj)->setSmallerBlockSize(value); END_WRAP }	cv::StereoBM::setSmallerBlockSize()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getROI1	CVAPI(ExceptionStatus) calib3d_StereoBM_getROI1(cv::Ptr<cv::StereoBM> *obj, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c((*obj)->getROI1()); END_WRAP }	cv::StereoBM::getROI1()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setROI1	CVAPI(ExceptionStatus) calib3d_StereoBM_setROI1(cv::Ptr<cv::StereoBM> *obj, MyCvRect value) { BEGIN_WRAP (*obj)->setROI1(cpp(value)); END_WRAP }	cv::StereoBM::setROI1()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_getROI2	CVAPI(ExceptionStatus) calib3d_StereoBM_getROI2(cv::Ptr<cv::StereoBM> *obj, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c((*obj)->getROI2()); END_WRAP }	cv::StereoBM::getROI2()	classcv_1_1StereoBM.html				1										
calib3d_StereoBM_setROI2	CVAPI(ExceptionStatus) calib3d_StereoBM_setROI2(cv::Ptr<cv::StereoBM> *obj, MyCvRect value) { BEGIN_WRAP (*obj)->setROI2(cpp(value)); END_WRAP }	cv::StereoBM::setROI2()	classcv_1_1StereoBM.html				1										
calib3d_Ptr_StereoSGBM_get	CVAPI(ExceptionStatus) calib3d_Ptr_StereoSGBM_get(cv::Ptr<cv::StereoSGBM> *obj, cv::StereoSGBM **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::StereoSGBM	classcv_1_1StereoSGBM.html	The class implements the modified H. Hirschmuller algorithm [111] that differs from the original one as follows: By default, the algorithm is single-pass, which means that you consider only 5 directions instead of 8. Set mode=StereoSGBM::MODE_HH in createStereoSGBM to run the full variant of the algorithm but beware that it may consume a lot of memory. The algorithm matches blocks, not individual pixels. Though, setting blockSize=1 reduces the blocks to single pixels. Mutual information cost function is not implemented. Instead, a simpler Birchfield-Tomasi sub-pixel metric from [23] is used. Though, the color images are supported as well. Some pre- and post- processing steps from K. Konolige algorithm StereoBM are included, for example: pre-filtering (StereoBM::PREFILTER_XSOBEL type) and post-filtering (uniqueness check, quadratic interpolation and speckle filtering). Note (Python) An example illustrating the use of the			1						•		
calib3d_Ptr_StereoSGBM_delete	CVAPI(ExceptionStatus) calib3d_Ptr_StereoSGBM_delete(cv::Ptr<cv::StereoSGBM> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::StereoSGBM	classcv_1_1StereoSGBM.html	The class implements the modified H. Hirschmuller algorithm [111] that differs from the original one as follows: By default, the algorithm is single-pass, which means that you consider only 5 directions instead of 8. Set mode=StereoSGBM::MODE_HH in createStereoSGBM to run the full variant of the algorithm but beware that it may consume a lot of memory. The algorithm matches blocks, not individual pixels. Though, setting blockSize=1 reduces the blocks to single pixels. Mutual information cost function is not implemented. Instead, a simpler Birchfield-Tomasi sub-pixel metric from [23] is used. Though, the color images are supported as well. Some pre- and post- processing steps from K. Konolige algorithm StereoBM are included, for example: pre-filtering (StereoBM::PREFILTER_XSOBEL type) and post-filtering (uniqueness check, quadratic interpolation and speckle filtering). Note (Python) An example illustrating the use of the										•	

calib3d_StereoSGBM_create	CVAPI(ExceptionStatus) calib3d_StereoSGBM_create(int minDisparity, int numDisparities, int blockSize, int P1, int P2, int disp12MaxDiff, int preFilterCap, int uniquenessRatio, int speckleWindowSize, int speckleRange, int mode, cv::Ptr<cv::StereoSGBM> **returnValue) { BEGIN_WRAP const auto obj = cv::StereoSGBM::create(minDisparity, numDisparities, blockSize, P1, P2, disp12MaxDiff, preFilterCap, uniquenessRatio, speckleWindowSize, speckleRange, mode); *returnValue = new cv::Ptr<cv::StereoSGBM>(obj); END_WRAP	cv::StereoSGBM::create()	classcv_1_1StereoSGBM.html	Creates StereoSGBM object.	minDisparity : Minimum possible disparity value. Normally, it is zero but sometimes rectification algorithms can shift images, so this parameter needs to be adjusted accordingly. numDisparities : Maximum disparity minus minimum disparity. The value is always greater than zero. In the current implementation, this parameter must be divisible by 16. blockSize : Matched block size. It must be an odd number >=1 . Normally, it should be somewhere in the 3..11 range. P1 : The first parameter controlling the disparity smoothness. See below. P2 : The second parameter controlling the disparity smoothness. The larger the values are, the smoother the disparity is. P1 is the penalty on the disparity change by plus or minus 1 between neighbor pixels. P2 is the penalty on the disparity change by more than 1 between neighbor pixels. The algorithm requires P2 > P1 . See stereo_match.cpp sample where some reasonably good P1 and P2 values are shown (like 8*number_of_image_channels*blockSize*blockSize and 32*number_of_image_channels*blockSize*blockSize , respectively). disp12MaxDiff : Maximum allowed difference (in integer pixel units) in the left-right disparity check. Set	The first constructor initializes StereoSGBM with all the default parameters. So, you only have to set StereoSGBM::numDisparities at minimum. The second constructor enables you to set each parameter to a custom value.	11								
calib3d_StereoSGBM_getPreFilterCap	CVAPI(ExceptionStatus) calib3d_StereoSGBM_getPreFilterCap(cv::Ptr<cv::StereoSGBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getPreFilterCap(); END_WRAP }	cv::StereoSGBM::getPreFilterCap()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_setPreFilterCap	CVAPI(ExceptionStatus) calib3d_StereoSGBM_setPreFilterCap(cv::Ptr<cv::StereoSGBM> *obj, int value) { BEGIN_WRAP (*obj)->setPreFilterCap(value); END_WRAP }	cv::StereoSGBM::setPreFilterCap()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_getUniquenessRatio	CVAPI(ExceptionStatus) calib3d_StereoSGBM_getUniquenessRatio(cv::Ptr<cv::StereoSGBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getUniquenessRatio(); END_WRAP }	cv::StereoSGBM::getUniquenessRatio()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_setUniquenessRatio	CVAPI(ExceptionStatus) calib3d_StereoSGBM_setUniquenessRatio(cv::Ptr<cv::StereoSGBM> *obj, int value) { BEGIN_WRAP (*obj)->setUniquenessRatio(value); END_WRAP }	cv::StereoSGBM::setUniquenessRatio()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_getP1	CVAPI(ExceptionStatus) calib3d_StereoSGBM_getP1(cv::Ptr<cv::StereoSGBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getP1(); END_WRAP }	cv::StereoSGBM::getP1()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_setP1	CVAPI(ExceptionStatus) calib3d_StereoSGBM_setP1(cv::Ptr<cv::StereoSGBM> *obj, int value) { BEGIN_WRAP (*obj)->setP1(value); END_WRAP }	cv::StereoSGBM::setP1()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_getP2	CVAPI(ExceptionStatus) calib3d_StereoSGBM_getP2(cv::Ptr<cv::StereoSGBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getP2(); END_WRAP }	cv::StereoSGBM::getP2()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_setP2	CVAPI(ExceptionStatus) calib3d_StereoSGBM_setP2(cv::Ptr<cv::StereoSGBM> *obj, int value) { BEGIN_WRAP (*obj)->setP2(value); END_WRAP }	cv::StereoSGBM::setP2()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_getMode	CVAPI(ExceptionStatus) calib3d_StereoSGBM_getMode(cv::Ptr<cv::StereoSGBM> *obj, int *returnValue) { BEGIN_WRAP *returnValue = (*obj)->getMode(); END_WRAP }	cv::StereoSGBM::getMode()	classcv_1_1StereoSGBM.html				1										
calib3d_StereoSGBM_setMode	CVAPI(ExceptionStatus) calib3d_StereoSGBM_setMode(cv::Ptr<cv::StereoSGBM> *obj, int value) { BEGIN_WRAP (*obj)->setMode(value); END_WRAP }	cv::StereoSGBM::setMode()	classcv_1_1StereoSGBM.html				1										
core_setBreakOnError	CVAPI(int) core_setBreakOnError(int flag) { return cv::setBreakOnError(flag != 0) ? 1 : 0; }	cv::setBreakOnError()	group__core__utils.html	Sets/resets the break-on-error mode.			When the break-on-error mode is set, the default error handler issues a hardware exception, which can make debugging more convenient.Examples: samples/cpp/stitching_detailed.cpp.	-1									
redirectError	CVAPI(cv::ErrorCallback) redirectError(cv::ErrorCallback errCallback, void* userdata, void** prevUserdata) { return cv::redirectError(errCallback, userdata, prevUserdata); }	cv::redirectError()	group__core__utils.html	Sets the new error handler and the optional user data.	errCallback : the new error handler. If NULL, the default error handler is used. userdata : the optional user data pointer, passed to the callback. prevUserdata : the optional output parameter where the previous user data pointer is stored		The function sets the new error handler, called from cv::error().	-1									
core_glob	CVAPI(ExceptionStatus) core_glob(const char *pattern, std::vector<std::string> *result, int recursive) { BEGIN_WRAP cv::glob(pattern, *result, recursive != 0); END_WRAP }	cv::glob()	group__core__utils.html	Examples: samples/cpp/train_HOG.cpp.				3									
core_setNumThreads	CVAPI(ExceptionStatus) core_setNumThreads(int nthreads) { BEGIN_WRAP cv::setNumThreads(nthreads); END_WRAP }	cv::setNumThreads()	group__core__utils.html	OpenCV will try to set the number of threads for the next parallel region.			If threads == 0, OpenCV will disable threading optimizations and run all it's functions sequentially. Passing threads < 0 will reset threads number to system default. This function must be called outside of parallel region.OpenCV will try to run its functions with specified threads number, but some behaviour differs from framework:TBB - User-defined parallel constructions will run with the same threads number, if another is not specified. If later on user creates his own scheduler, OpenCV will use it. OpenMP - No special defined behaviour. Concurrency - If threads == 1, OpenCV will disable threading optimizations and run its functions sequentially. GCD - Supports only values <= 0. C= - No special defined behaviour. Parameters nthreadsNumber of threads used by OpenCV.	1									

core_getNumThreads	CVAPI(ExceptionStatus) core_getNumThreads(int* returnValue) { BEGIN_WRAP *returnValue = cv::getNumThreads(); END_WRAP }	cv::getNumThreads()	group__core__utils.html	Returns the number of threads used by OpenCV for parallel regions.		Always returns 1 if OpenCV is built without threading support.The exact meaning of return value depends on the threading framework used by OpenCV library:TBB - The number of threads, that OpenCV will try to use for parallel regions. If there is any tbb::thread_scheduler_init in user code conflicting with OpenCV, then function returns default number of threads used by TBB library. OpenMP - An upper bound on the number of threads that could be used to form a new team. Concurrency - The number of threads, that OpenCV will try to use for parallel regions. GCD - Unsupported; returns the GCD thread pool limit (512) for compatibility. C= - The number of threads, that OpenCV will try to use for parallel regions, if before called setNumThreads with threads > 0, otherwise returns the number of logical CPUs, available for the process. See alsosetNumThreads, getThreadNum	1									
core_getThreadNum	CVAPI(ExceptionStatus) core_getThreadNum(int* returnValue) { BEGIN_WRAP *returnValue = cv::getThreadNum(); END_WRAP }	cv::getThreadNum()	group__core__utils.html	Returns the index of the currently executed thread within the current parallel region. Always returns 0 if called outside of parallel region.		Deprecated:Current implementation doesn't corresponding to this documentation.The exact meaning of the return value depends on the threading framework used by OpenCV library:TBB - Unsupported with current 4.1 TBB release. Maybe will be supported in future. OpenMP - The thread number, within the current team, of the calling thread. Concurrency - An ID for the virtual processor that the current context is executing on (0 for master thread and unique number for others, but not necessary 1,2,3,...). GCD - System calling thread's ID. Never returns 0 inside parallel region. C= - The index of the current parallel task. See alsosetNumThreads,	1									
core_getBuildInformation	CVAPI(ExceptionStatus) core_getBuildInformation(std::string *buf) { BEGIN_WRAP const auto& str = cv::getBuildInformation(); buf->assign(str); END_WRAP }	cv::getBuildInformation()	group__core__utils.html	Returns full configuration time cmake output.		Returned value is raw cmake output including version control system revision, compiler version, compiler flags, enabled modules and third party libraries, etc. Output format depends on target architecture.	1									
core_getVersionString	CVAPI(ExceptionStatus) core_getVersionString(char *buf, int bufLength) { BEGIN_WRAP const auto& str = cv::getVersionString(); copyString(str, buf, bufLength); END_WRAP }	cv::getVersionString()	group__core__utils.html	Returns library version string.		For example "3.4.1-dev".See alsogetMajorVersion, getMinorVersion, getRevisionVersion	3									
core_getVersionMajor	CVAPI(ExceptionStatus) core_getVersionMajor(int* returnValue) { BEGIN_WRAP *returnValue = cv::getVersionMajor(); END_WRAP }	cv::getVersionMajor()	group__core__utils.html	Returns major library version.			1									
core_getVersionMinor	CVAPI(ExceptionStatus) core_getVersionMinor(int* returnValue) { BEGIN_WRAP *returnValue = cv::getVersionMinor(); END_WRAP }	cv::getVersionMinor()	group__core__utils.html	Returns minor library version.			1									
core_getVersionRevision	CVAPI(ExceptionStatus) core_getVersionRevision(int* returnValue) { BEGIN_WRAP *returnValue = cv::getVersionRevision(); END_WRAP }	cv::getVersionRevision()	group__core__utils.html	Returns revision field of the library version.			1									
core_getTickCount	CVAPI(ExceptionStatus) core_getTickCount(int64* returnValue) { BEGIN_WRAP *returnValue = cv::getTickCount(); END_WRAP }	cv::getTickCount()	group__core__utils.html	Returns the number of ticks.		The function returns the number of ticks after the certain event (for example, when the machine was turned on). It can be used to initialize RNG or to measure a function execution time by reading the tick count before and after the function call.See alsogetTickFrequency, TickMeterExamples: fid_lines.cpp, samples/cpp/facedetect.cpp, samples/cpp/image_alignment.cpp, samples/cpp/peopledetect.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/watershed.cpp, samples/tapi/hog.cpp, and	1									
core_getTickFrequency	CVAPI(ExceptionStatus) core_getTickFrequency(double* returnValue) { BEGIN_WRAP *returnValue = cv::getTickFrequency(); END_WRAP }	cv::getTickFrequency()	group__core__utils.html	Returns the number of ticks per second.		The function returns the number of ticks per second. That is, the following code computes the execution time in seconds:double t = (double)getTickCount();// do something ...t = ((double)getTickCount() - t)/getTickFrequency();fragmentSee alsogetTickCount, TickMeterExamples: fid_lines.cpp, samples/cpp/facedetect.cpp, samples/cpp/image_alignment.cpp, samples/cpp/peopledetect.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/watershed.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, samples/dnn/segmentation.cpp, samples/tapi/hog.cpp, and	1									
core_getCPUTickCount	CVAPI(ExceptionStatus) core_getCPUTickCount(int64* returnValue) { BEGIN_WRAP *returnValue = cv::getCPUTickCount(); END_WRAP }	cv::getCPUTickCount()	group__core__utils.html	Returns the number of CPU ticks.		The function returns the current number of CPU ticks on some architectures (such as x86, x64, PowerPC). On other platforms the function is equivalent to getTickCount. It can also be used for very accurate time measurements, as well as for RNG initialization. Note that in case of multi-CPU systems a thread, from which getCPUTickCount is called, can be suspended and resumed at another CPU with its own counter. So, theoretically (and practically) the subsequent calls to the function do not necessary return the monotonously increasing values. Also, since a modern CPU varies the CPU frequency depending on the load, the number of CPU clocks spent in some code cannot be directly converted to time units. Therefore, getTickCount is generally a preferable solution for measuring	1									
core_checkHardwareSupport	CVAPI(ExceptionStatus) core_checkHardwareSupport(int feature, int* returnValue) { BEGIN_WRAP *returnValue = cv::checkHardwareSupport(feature) ? 1 : 0; END_WRAP }	cv::checkHardwareSupport()	group__core__utils.html	Returns true if the specified feature is supported by the host hardware.	feature : The feature of interest, one of cv::CpuFeatures	The function returns true if the host hardware supports the specified feature. When user calls setUseOptimized(false), the subsequent calls to checkHardwareSupport() will return false until setUseOptimized(true) is called. This way user can dynamically switch on and off the optimized code in OpenCV.	1									
core_getHardwareFeatureName	CVAPI(ExceptionStatus) core_getHardwareFeatureName(int feature, std::string *buf) { BEGIN_WRAP const auto& str = cv::getHardwareFeatureName(feature); buf->assign(str); END_WRAP }	cv::getHardwareFeatureName()	group__core__utils.html	Returns feature name by ID.		Returns empty string if feature is not defined	1									
core_getCPUFeaturesLine	CVAPI(ExceptionStatus) core_getCPUFeaturesLine(std::string *buf) { BEGIN_WRAP const auto& str = cv::getCPUFeaturesLine(); buf->assign(str); END_WRAP }	cv::getCPUFeaturesLine()	group__core__utils.html	Returns list of CPU features enabled during compilation.		Returned value is a string containing space separated list of CPU features with following markers:no markers - baseline features prefix * - features enabled in dispatcher suffix ? - features enabled but not available in HWExample: SSE SSE2 SSE3 *SSE4.1 *SSE4.2 *FP16 *AVX *AVX2 *AVX512-SKX?	1									
core_getNumberOfCPUs	CVAPI(ExceptionStatus) core_getNumberOfCPUs(int* returnValue) { BEGIN_WRAP *returnValue = cv::getNumberOfCPUs(); END_WRAP }	cv::getNumberOfCPUs()	group__core__utils.html	Returns the number of logical CPUs available for the process.			1									
core_setUseOptimized	CVAPI(ExceptionStatus) core_setUseOptimized(int onoff) { BEGIN_WRAP cv::setUseOptimized(onoff != 0); END_WRAP }	cv::setUseOptimized()	group__core__utils.html	Enables or disables the optimized code.	onoff : The boolean flag specifying whether the optimized code should be used (onoff=true) or not (onoff=false).	The function can be used to dynamically turn on and off optimized dispatched code (code that uses SSE4.2, AVX/AVX2, and other instructions on the platforms that support it). It sets a global flag that is further checked by OpenCV functions. Since the flag is not checked in the inner OpenCV loops, it is only safe to call the function on the very top level in your application where you can be sure that no other OpenCV function is currently executed.By default, the optimized code is enabled unless you disable it in CMake. The current status can be retrieved using	1									

core_useOptimized	CVAPI(ExceptionStatus) core_useOptimized(int *returnValue) { BEGIN_WRAP *returnValue = cv::useOptimized() ? 1 : 0; END_WRAP }	cv::useOptimized()	group__core__utils.html	Returns the status of optimized code usage.		The function returns true if the optimized code is enabled. Otherwise, it returns false.	1								
core_format	CVAPI(ExceptionStatus) core_format(cv::_InputArray *mtx, int fmt, std::string *buf) { BEGIN_WRAP const auto formatted = cv::format(*mtx, static_cast<cv::Formatter::Fo rmatType>(fmt)); std::stringstream s; s << formatted; buf->assign(s.str()); END_WRAP }	cv::format()													
core_borderInterpolate	CVAPI(ExceptionStatus) core_borderInterpolate(int p, int len, int borderType, int* returnValue) { BEGIN_WRAP *returnValue = cv::borderInterpolate(p, len, borderType); END_WRAP }	cv::borderInterpolate()	group__core__array.html	Computes the source location of an extrapolated pixel.	p : 0-based coordinate of the extrapolated pixel along one of the axes, likely <0 or >= len len : Length of the array along the corresponding axis. borderType : Border type, one of the BorderTypes, except for BORDER_TRANSPARENT and BORDER_ISOLATED . When borderType==BORDER_CONSTANT , the function always returns -1, regardless of p and len.	The function computes and returns the coordinate of a donor pixel corresponding to the specified extrapolated pixel when using the specified extrapolation border mode. For example, if you use cv::BORDER_WRAP mode in the horizontal direction, cv::BORDER_REFLECT_101 in the vertical direction and want to compute value of the "virtual" pixel Point(-5, 100) in a floating-point image img , it looks like:float val = img.at<float>(borderInterpolate(100, img.rows, cv::BORDER_REFLECT_101), borderInterpolate(-5, img.cols, cv::BORDER_WRAP));fragmentNormally, the function is not called directly. It is used inside filtering functions and also in copyMakeBorder.See alsocopyMakeBorder	3								
core_copyMakeBorder	CVAPI(ExceptionStatus) core_copyMakeBorder(cv::_InputArray* src, cv::_OutputArray* dst, int top, int bottom, int left, int right, int borderType, MyCvScalar value) { BEGIN_WRAP cv::copyMakeBorder(*src, *dst, top, bottom, left, right, borderType, cpp(value)); END_WRAP }	cv::copyMakeBorder()	group__core__array.html	Forms a border around an image.	src : Source image. dst : Destination image of the same type as src and the size Size(src.cols+left+right, src.rows+top+bottom) . top : the top pixels bottom : the bottom pixels left : the left pixels right : Parameter specifying how many pixels in each direction from the source image rectangle to extrapolate. For example, top=1, bottom=1, left=1, right=1 mean that 1 pixel-wide border needs to be built. borderType : Border type. See borderInterpolate for details. value : Border value if borderType==BORDER_CONSTANT .	The function copies the source image into the middle of the destination image. The areas to the left, to the right, above and below the copied source image will be filled with extrapolated pixels. This is not what filtering functions based on it do (they extrapolate pixels on-fly), but what other more complex functions, including your own, may do to simplify image boundary handling.The function supports the mode when src is already in the middle of dst . In this case, the function does not copy src itself but simply constructs the border, for example:// let border be the same in all directionint border=2;// constructs a larger image to fit both the image and the borderMat gray_buf(rgb.rows + border*2, rgb.cols + border*2, rgb.depth());// select the middle part of it w/o copying dataMat gray(gray_canvas, Rect(border, border, rgb.cols, rgb.rows));// convert image from RGB to grayscalecvtColor(rgb, gray, COLOR_RGB2GRAY);// form a border in-placecopyMakeBorder(gray, gray_buf, border, border, border, border, BORDER_REPLICATE);// now do some	8								
core_add	CVAPI(ExceptionStatus) core_add(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask, int dtype) { BEGIN_WRAP cv::add(*src1, *src2, *dst, entity(mask), dtype); END_WRAP }	cv::add()	group__core__array.html	Calculates the per-element sum of two arrays or an array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array that has the same size and number of channels as the input array(s); the depth is defined by dtype or src1/src2. mask : optional operation mask - 8-bit single channel array, that specifies elements of the output array to be changed. dtype : optional depth of the output array (see the discussion below).	The function add calculates:Sum of two arrays when both input arrays have the same size and the same number of channels: $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) + \texttt{src2}(I))$$ Sum of an array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) + \texttt{src2})$$ Sum of a scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1} + \texttt{src2}(I))$$ where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed	5								
core_subtract_InputArray2	CVAPI(ExceptionStatus) core_subtract_InputArray2(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask, int dtype) { BEGIN_WRAP cv::subtract(*src1, *src2, *dst, entity(mask), dtype); END_WRAP }	cv::subtract()	group__core__array.html	Calculates the per-element difference between two arrays or array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array of the same size and the same number of channels as the input array. mask : optional operation mask; this is an 8-bit single channel array that specifies elements of the output array to be changed. dtype : optional depth of the output array	The function subtract calculates:Difference between two arrays, when both input arrays have the same size and the same number of channels: $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$ Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}) \quad \texttt{if mask}(I) \ne 0$$ Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1} - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$	5								
core_subtract_InputArrayScalar	CVAPI(ExceptionStatus) core_subtract_InputArrayScal ar(cv::_InputArray *src1, MyCvScalar src2, cv::_OutputArray *dst, cv::_InputArray *mask, int dtype) { BEGIN_WRAP cv::subtract(*src1, cpp(src2), *dst, entity(mask), dtype); END_WRAP }	cv::subtract()	group__core__array.html	Calculates the per-element difference between two arrays or array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array of the same size and the same number of channels as the input array. mask : optional operation mask; this is an 8-bit single channel array that specifies elements of the output array to be changed. dtype : optional depth of the output array	The function subtract calculates:Difference between two arrays, when both input arrays have the same size and the same number of channels: $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$ Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}) \quad \texttt{if mask}(I) \ne 0$$ Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1} - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$	5								
core_subtract_ScalarInputArray	CVAPI(ExceptionStatus) core_subtract_ScalarInputArra y(MyCvScalar src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask, int dtype) { BEGIN_WRAP cv::subtract(cpp(src1), *src2, *dst, entity(mask), dtype); END_WRAP }	cv::subtract()	group__core__array.html	Calculates the per-element difference between two arrays or array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array of the same size and the same number of channels as the input array. mask : optional operation mask; this is an 8-bit single channel array that specifies elements of the output array to be changed. dtype : optional depth of the output array	The function subtract calculates:Difference between two arrays, when both input arrays have the same size and the same number of channels: $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$ Difference between an array and a scalar, when src2 is constructed from Scalar or has the same number of elements as src1.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1}(I) - \texttt{src2}) \quad \texttt{if mask}(I) \ne 0$$ Difference between a scalar and an array, when src1 is constructed from Scalar or has the same number of elements as src2.channels(): $$\texttt{dst}(I) = \texttt{saturate} (\texttt{src1} - \texttt{src2}(I)) \quad \texttt{if mask}(I) \ne 0$$	5								
core_multiply	CVAPI(ExceptionStatus) core_multiply(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, double scale, int dtype) { BEGIN_WRAP cv::multiply(*src1, *src2, *dst, scale, dtype); END_WRAP }	cv::multiply()	group__core__array.html	Calculates the per-element scaled product of two arrays.	src1 : first input array. src2 : second input array of the same size and the same type as src1. dst : output array of the same size and type as src1. scale : optional scale factor. dtype : optional depth of the output array	The function multiply calculates the per-element product of two arrays:$$\texttt{dst} (I)= \texttt{saturate}(\texttt{src1} (I) * \texttt{scale} \texttt{src2} (I))$$There is also a MatrixExpressions -friendly variant of the first function. See Mat::mul .For a not-per-element matrix product, see gemm .NoteSaturation is not applied when the output array has the depth CV_32S. You may even get result of an incorrect sign in the case of overflow.See alsoadd, subtract, divide, scaleAdd, addWeighted, accumulate, accumulateProduct, accumulateSquare,	5								

core_divide1	CVAPI(ExceptionStatus) core_divide1(double scale, cv::_InputArray *src2, cv::_OutputArray *dst, int dtype) { BEGIN_WRAP cv::divide(scale, *src2, *dst, dtype); END_WRAP }	cv::divide()	group__core__array.html	Performs per-element division of two arrays or a scalar by an array.	src1 : first input array. src2 : second input array of the same size and type as src1. scale : scalar factor. dst : output array of the same size and type as src2. dtype : optional depth of the output array; if -1, dst will have depth src2.depth(), but in case of an array-by-array division, you can only pass -1 when src1.depth()==src2.depth().	The function cv::divide divides one array by another:¥[¥texttt{dst(I) = saturate(src1(I)*scale/src2(I))¥}or a scalar by an array when there is no src1 :¥[¥texttt{dst(I) = saturate(scale/src2(I))¥}Different channels of multi-channel arrays are processed independently.For integer types when src2(I) is zero, dst(I) will also be zero.Note3In case of floating point data there is no special defined behavior for zero src2(I) values. Regular floating-point division is used. Expect correct IEEE-754 behaviour for floating-point data (with NaN, Inf result values). Saturation is not applied when the output array has the depth CV_32S. You may even get result of an incorrect sign in the case of overflow.See alsomultiply, add, subtractExamples:	4								
core_divide2	CVAPI(ExceptionStatus) core_divide2(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, double scale, int dtype) { BEGIN_WRAP cv::divide(*src1, *src2, *dst, scale, dtype); END_WRAP }	cv::divide()	group__core__array.html	Performs per-element division of two arrays or a scalar by an array.	src1 : first input array. src2 : second input array of the same size and type as src1. scale : scalar factor. dst : output array of the same size and type as src2. dtype : optional depth of the output array; if -1, dst will have depth src2.depth(), but in case of an array-by-array division, you can only pass -1 when src1.depth()==src2.depth().	The function cv::divide divides one array by another:¥[¥texttt{dst(I) = saturate(src1(I)*scale/src2(I))¥}or a scalar by an array when there is no src1 :¥[¥texttt{dst(I) = saturate(scale/src2(I))¥}Different channels of multi-channel arrays are processed independently.For integer types when src2(I) is zero, dst(I) will also be zero.Note3In case of floating point data there is no special defined behavior for zero src2(I) values. Regular floating-point division is used. Expect correct IEEE-754 behaviour for floating-point data (with NaN, Inf result values). Saturation is not applied when the output array has the depth CV_32S. You may even get result of an incorrect sign in the case of overflow.See alsomultiply, add, subtractExamples:	5								
core_scaleAdd	CVAPI(ExceptionStatus) core_scaleAdd(cv::_InputArra y *src1, double alpha, cv::_InputArray *src2, cv::_OutputArray *dst) { BEGIN_WRAP cv::scaleAdd(*src1, alpha, *src2, *dst); END_WRAP }	cv::scaleAdd()	group__core__array.html	Calculates the sum of a scaled array and another array.	src1 : first input array. alpha : scale factor for the first array. src2 : second input array of the same size and type as src1. dst : output array of the same size and type as src1.	The function scaleAdd is one of the classical primitive linear algebra operations, known as DAXPY or SAXPY in BLAS. It calculates the sum of a scaled array and another array:¥[¥texttt{dst}(I)= ¥texttt{scale} ¥cdot ¥texttt{src1}(I) + ¥texttt{src2} (I)¥]The function can also be emulated with a matrix expression, for example:Mat A(3, 3, CV_64F);...A.row(0) = A.row(1)*2 + A.row(2);fragmentSee alsoadd, addWeighted, subtract, Mat::dot, Mat::convertTo	4								
core_addWeighted	CVAPI(ExceptionStatus) core_addWeighted(cv::_Input Array *src1, double alpha, cv::_InputArray *src2, double beta, double gamma, cv::_OutputArray *dst, int dtype) { BEGIN_WRAP cv::addWeighted(*src1, alpha, *src2, beta, gamma, *dst, dtype); END_WRAP }	cv::addWeighted()	group__core__array.html	Calculates the weighted sum of two arrays.	src1 : first input array. alpha : weight of the first array elements. src2 : second input array of the same size and channel number as src1. beta : weight of the second array elements. gamma : scalar added to each sum. dst : output array that has the same size and number of channels as the input arrays. dtype : optional depth of the output array; when both input arrays have the same depth, dtype can be set to -1, which will be equivalent to src1.depth().	The function addWeighted calculates the weighted sum of two arrays as follows:¥[¥texttt{dst} (I)= ¥texttt{saturate} (¥texttt{src1} (I)* ¥texttt{alpha} + ¥texttt{src2} (I)* ¥texttt{beta} + ¥texttt{gamma})¥]where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently. The function can be replaced with a matrix expression:dst = src1*alpha + src2*beta + gamma;fragmentNoteSaturation is not applied when the output array has the depth CV_32S. You may even get result of an incorrect sign in the case of overflow.See alsoadd, subtract, scaleAdd, Mat::convertToExamples: samples/cpp/tutorial_code/HighGUI/AddingImagesTrackbar.cpp, samples/cpp/tutorial_code/ImgTrans/Sobel_Demo.cpp, and	7								
core_convertScaleAbs	CVAPI(ExceptionStatus) core_convertScaleAbs(cv::_In putArray* src, cv::_OutputArray* dst, double alpha, double beta) { BEGIN_WRAP cv::convertScaleAbs(*src, *dst, alpha, beta); END_WRAP }	cv::convertScaleAbs()	group__core__array.html	Scales, calculates absolute values, and converts the result to 8-bit.	src : input array. dst : output array. alpha : optional scale factor. beta : optional delta added to the scaled values.	On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type:¥[¥texttt{dst} (I)= ¥texttt{saturate¥_cast<uchar>} (|¥texttt{src} (I)* ¥texttt{alpha} + ¥texttt{beta} |)¥]In case of multi-channel arrays, the function processes each channel independently. When the output is not 8-bit, the operation can be emulated by calling the Mat::convertTo method (or by using matrix expressions) and then by calculating an absolute value of the result. For example:Mat_<float> A(30,30);randu(A, Scalar(-100), Scalar(100));Mat_<float> B = A*5 + 3;B = abs(B);// Mat_<float> B = abs(A*5+3) will also do the job,// but it will allocate a temporary matrix¥fragmentSee alsoMat::convertTo, cv::abs(const Mat&)Examples: samples/cpp/laplace.cpp, and samples/cpp/tutorial_code/ImgTrans/Sobel_Demo.cpp, and	4								
core_convertFp16	CVAPI(ExceptionStatus) core_convertFp16(cv::_InputA rray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::convertFp16(*src, *dst); END_WRAP }	cv::convertFp16()	group__core__array.html	Converts an array to half precision floating number.	src : input array. dst : output array.	This function converts FP32 (single precision floating point) from/to FP16 (half precision floating point). CV_16S format is used to represent FP16 data. There are two use modes (src -> dst): CV_32F -> CV_16S and CV_16S -> CV_32F. The input array has to have type of CV_32F or CV_16S to represent the bit depth. If the input array is neither of them, the function will raise an error. The format of half precision floating point is defined in IEEE 754, page 56	2								
core_LUT	CVAPI(ExceptionStatus) core_LUT(cv::_InputArray* src, cv::_InputArray* lut, cv::_OutputArray* dst) { BEGIN_WRAP cv::LUT(*src, *lut, *dst); END_WRAP }	cv::LUT()	group__core__array.html	Performs a look-up table transform of an array.	src : input array of 8-bit elements. lut : look-up table of 256 elements; in case of multi-channel input array, the table should either have a single channel (in this case the same table is used for all channels) or the same number of channels as in the input array. dst : output array of the same size and number of channels as src, and the same depth as lut.	The function LUT fills the output array with values from the look-up table. Indices of the entries are taken from the input array. That is, the function processes each element of src as follows:¥[¥texttt{dst} (I) ¥leftarrow ¥texttt{lut[src(I + d)]}¥]where¥[d = ¥left(0,0\right)(if ¥(¥texttt{src}¥) has depth ¥(¥texttt{CV_8U}¥)¥]{128}{if ¥(¥texttt{src}¥) has depth ¥(¥texttt{CV_8S}¥)¥]See alsoconvertScaleAbs, Mat::convertTo	3								
core_sum	CVAPI(ExceptionStatus) core_sum(cv::_InputArray* src, MyCvScalar* returnValue) { BEGIN_WRAP *returnValue = c(cv::sum(*src)); END_WRAP }	cv::sum()	group__core__array.html	Calculates the sum of array elements.	src : input array that must have from 1 to 4 channels.	The function cv::sum calculates and returns the sum of array elements, independently for each channel.See alsocountNonZero, mean, meanStdDev, norm, minMaxLoc, reduce	1								
core_countNonZero	CVAPI(ExceptionStatus) core_countNonZero(cv::_Inpu tArray* src, int* returnValue) { BEGIN_WRAP *returnValue = cv::countNonZero(*src); END_WRAP }	cv::countNonZero()	group__core__array.html	Counts non-zero array elements.	src : single-channel array.	The function returns the number of non-zero elements in src :¥[¥sum _{I: ¥texttt{src} (I) ¥ne0 } 1¥]See alsomean, meanStdDev, norm, minMaxLoc, calcCovarMatrix	1								
core_findNonZero	CVAPI(ExceptionStatus) core_findNonZero(cv::_InputA rray* src, cv::_OutputArray* idx) { BEGIN_WRAP cv::findNonZero(*src, *idx); END_WRAP }	cv::findNonZero()	group__core__array.html	Returns the list of locations of non-zero pixels.	src : single-channel array idx : the output array, type of cv::Mat or std::vector<Point>, corresponding to non-zero indices in the input	Given a binary matrix (likely returned from an operation such as threshold(), compare(), >, ==, etc, return all of the non-zero indices as a cv::Mat or std::vector<cv::Point> (x,y) For example:cv::Mat binaryImage; // input, binary imagecv::Mat locations; // output, locations of non-zero pixelscv::findNonZero(binaryImage, locations);// access pixel coordinatesPoint pnt = locations.at<Point>(i);fragmentcv::Mat binaryImage; // input, binary imagevector<Point> locations; // output, locations of non-zero pixelscv::findNonZero(binaryImage, locations);// access pixel coordinatesPoint pnt = locations[i];fragment	2								
core_mean	CVAPI(ExceptionStatus) core_mean(cv::_InputArray* src, cv::_InputArray* mask, MyCvScalar* returnValue) { BEGIN_WRAP *returnValue = c(cv::mean(*src, entity(*mask))); END_WRAP }	cv::mean()	group__core__array.html	Calculates an average (mean) of array elements.	src : input array that should have from 1 to 4 channels so that the result can be stored in Scalar_ . mask : optional operation mask.	The function cv::mean calculates the mean value M of array elements, independently for each channel, and return it:¥[¥begin{array}{l} N = ¥sum _{I: ¥, ¥texttt{mask} (I) ¥ne 0} 1 ¥¥ M_c = ¥left (¥sum _{I: ¥, ¥texttt{mask} (I) ¥ne 0}{ ¥texttt{mtx} (I)_c} ¥right)/N ¥end{array}¥]When all the mask elements are 0's, the function returns Scalar::all(0)See alsocountNonZero, meanStdDev, norm, minMaxLocExamples: samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, and samples/dnn/segmentation.cpp.	2								
core_meanStdDev_OutputArray	CVAPI(ExceptionStatus) core_meanStdDev_OutputArr ay(cv::_InputArray* src, cv::_OutputArray* mean, cv::_OutputArray* stddev, cv::_InputArray* mask) { BEGIN_WRAP cv::meanStdDev(*src, *mean, *stddev, entity(mask)); END_WRAP }	cv::meanStdDev()	group__core__array.html	Calculates a mean and standard deviation of array elements.	src : input array that should have from 1 to 4 channels so that the results can be stored in Scalar_ 's. mean : output parameter: calculated mean value. stddev : output parameter: calculated standard deviation. mask : optional operation mask.	The function cv::meanStdDev calculates the mean and the standard deviation M of array elements independently for each channel and returns it via the output parameters:¥[¥begin{array}{l} N = ¥sum _{I, ¥texttt{mask} (I) ¥ne 0} 1 ¥¥ ¥texttt{mean} _c = ¥frac{¥sum_{ I: ¥, ¥texttt{mask}(I) ¥ne 0} ¥texttt{src} (I)_c}{N} ¥¥ ¥texttt{stddev} _c = ¥sqrt{¥frac{¥sum_{ I: ¥, ¥texttt{mask}(I) ¥ne 0} ¥left (¥texttt{src} (I)_c - ¥texttt{mean} _c ¥right)^2}{N}} ¥end{array}¥]When all the mask elements are 0's, the function returns mean=stddev=Scalar::all(0).NoteThe calculated standard deviation is only the diagonal of the complete normalized covariance matrix. If the full matrix is needed, you can reshape the multi-channel array M x N to the single-channel array M*N x mtx.channels() (only possible when the matrix is continuous) and then pass the matrix to calcCovarMatrix .See alsocountNonZero, mean, norm, minMaxLoc, calcCovarMatrix	4								

core_meanStdDev_Scalar	CVAPI(ExceptionStatus) core_meanStdDev_Scalar(cv::_InputArray* src, MyCvScalar* mean, MyCvScalar* stddev, cv::_InputArray* mask) { BEGIN_WRAP cv::Scalar mean0, stddev0; cv::meanStdDev(*src, mean0, stddev0, entity(mask)); *mean = c[mean0]; *stddev = c[stddev0]; END_WRAP }	cv::meanStdDev()	group__core__array.html	Calculates a mean and standard deviation of array elements.	src : input array that should have from 1 to 4 channels so that the results can be stored in Scalar_'s. mean : output parameter: calculated mean value. stddev : output parameter: calculated standard deviation. mask : optional operation mask.	The function cv::meanStdDev calculates the mean and the standard deviation M of array elements independently for each channel and returns it via the output parameters:¥[¥begin{array}{1} N = ¥sum_{I, ¥texttt{mask} (I) ¥ne 0} 1 ¥¥ ¥texttt{mean} _c = ¥frac{¥sum_{ I: ¥; ¥texttt{mask}(I) ¥ne 0} ¥texttt{src}(I)_c}{N} ¥¥ ¥texttt{stddev} _c = ¥sqrt{¥frac{¥sum_{ I: ¥; ¥texttt{mask}(I) ¥ne 0} ¥left (¥texttt{src} (I)_c - ¥texttt{mean} _c ¥right)^2}{N}} ¥end{array}¥]When all the mask elements are 0's, the function returns mean=stddev=Scalar::all(0).NoteThe calculated standard deviation is only the diagonal of the complete normalized covariance matrix. If the full matrix is needed, you can reshape the multi-channel array M x N to the single-channel array M*N x mtx.channels() (only possible when the matrix is continuous) and then pass the matrix to calcCovarMatrix .See alsocountNonZero, mean, norm, minMaxLoc, calcCovarMatrix	4								
core_norm1	CVAPI(ExceptionStatus) core_norm1(cv::_InputArray* src1, int normType, cv::_InputArray* mask, double *returnValue) { BEGIN_WRAP *returnValue = cv::norm(*src1, normType, entity(mask)); END_WRAP }	cv::norm()	group__core__array.html	Calculates the absolute norm of an array.	src1 : first input array. normType : type of the norm (see NormTypes). mask : optional operation mask; it must have the same size as src1 and CV_8UC1 type.	This version of norm calculates the absolute norm of src1. The type of norm to calculate is specified using NormTypes.As example for one array consider the function ¥(r(x)= ¥begin{pmatrix} x ¥¥ 1-x ¥end{pmatrix}, x ¥in [-1;1]¥). The ¥(L_{1}, L_{2} ¥) and ¥(L_{¥infty} ¥) norm for the sample value ¥(r(-1) = ¥begin{pmatrix} -1 ¥¥ 2 ¥end{pmatrix}¥) is calculated as follows¥begin{align*} ¥| r(-1) ¥|_{L_1} &= |-1| + |2| = 3 ¥¥ ¥| r(-1) ¥|_{L_2} &= ¥sqrt{(-1)^{2} + (2)^{2}} = ¥sqrt{5} ¥¥ ¥| r(-1) ¥|_{L_¥infty} &= ¥max{|-1|,|2|} = 2 ¥end{align*}and for ¥(r(0.5) = ¥begin{pmatrix} 0.5 ¥¥ 0.5 ¥end{pmatrix}¥) the calculation is¥begin{align*} ¥| r(0.5) ¥|_{L_1} &= |0.5| + |0.5| = 1 ¥¥ ¥| r(0.5) ¥|_{L_2} &= ¥sqrt{(0.5)^{2} + (0.5)^{2}} = ¥sqrt{0.5} ¥¥ ¥| r(0.5) ¥|_{L_¥infty} &= ¥max{|0.5|,|0.5|} = 0.5. ¥end{align*}The following graphic shows all values for the three norm functions ¥(¥| r(x) ¥|_{L_1}, ¥| r(x) ¥|_{L_2}¥) and ¥(¥| r(x) ¥|_{L_¥infty}¥). It is notable that the ¥(L_{1} ¥) norm forms	3								
core_norm2	CVAPI(ExceptionStatus) core_norm2(cv::_InputArray* src1, cv::_InputArray* src2, int normType, cv::_InputArray* mask, double* returnValue) { BEGIN_WRAP *returnValue = cv::norm(*src1, *src2, normType, entity(mask)); END_WRAP }	cv::norm()	group__core__array.html	Calculates an absolute difference norm or a relative difference norm.	src1 : first input array. src2 : second input array of the same size and the same type as src1. normType : type of the norm (see NormTypes). mask : optional operation mask; it must have the same size as src1 and CV_8UC1 type.	This version of cv::norm calculates the absolute difference norm or the relative difference norm of arrays src1 and src2. The type of norm to calculate is specified using NormTypes.	4								
core_PSNR	CVAPI(ExceptionStatus) core_PSNR(cv::_InputArray* src1, cv::_InputArray* src2, double R, double* returnValue) { BEGIN_WRAP *returnValue = cv::PSNR(*src1, *src2, R); END_WRAP }	cv::PSNR()	group__core__array.html	Computes the Peak Signal-to-Noise Ratio (PSNR) image quality metric.	src1 : first input array. src2 : second input array of the same size as src1. R : the maximum pixel value (255 by default)	This function calculates the Peak Signal-to-Noise Ratio (PSNR) image quality metric in decibels (dB), between two input arrays src1 and src2. The arrays must have the same type.The PSNR is calculated as follows:¥[¥texttt{PSNR} = 10 ¥cdot ¥log_{10}{¥left(¥frac{R^2}{MSE} ¥right) }¥]where R is the maximum integer value of depth (e.g. 255 in the case of CV_8U data) and MSE is the mean squared error between the two arrays.	3								
core_batchDistance	CVAPI(ExceptionStatus) core_batchDistance(cv::_InputArray* src1, cv::_InputArray* src2, cv::_OutputArray* dist, int dtype, cv::_OutputArray* nidx, int normType, int K, cv::_InputArray* mask, int update, int crosscheck) { BEGIN_WRAP cv::batchDistance(*src1, *src2, *dist, dtype, *nidx, normType, K, entity(mask), update, crosscheck != 0); END_WRAP }	cv::batchDistance()	group__core__array.html	naive nearest neighbor finder		see http://en.wikipedia.org/wiki/Nearest_neighbor_searchTodo:document	10								
core_normalize	CVAPI(ExceptionStatus) core_normalize(cv::_InputArray* src, cv::_InputOutputArray* dst, double alpha, double beta, int normType, int dtype, cv::_InputArray* mask) { BEGIN_WRAP cv::InputArray maskVal = entity(mask); cv::normalize(*src, *dst, alpha, beta, normType, dtype, maskVal); END_WRAP }	cv::normalize()	group__core__array.html	Normalizes the norm or value range of an array.	src : input array. dst : output array of the same size as src . alpha : norm value to normalize to or the lower range boundary in case of the range normalization. beta : upper range boundary in case of the range normalization; it is not used for the norm normalization. norm_type : normalization type (see cv::NormTypes). dtype : when negative, the output array has the same type as src; otherwise, it has the same number of channels as src and the depth =CV_MAT_DEPTH(dtype). mask : optional operation mask.	The function cv::normalize normalizes scale and shift the input array elements so that¥[¥texttt{dst} ¥	_{L_p}= ¥texttt{alpha}¥]{where p=Inf, 1 or 2} when normType=NORM_INF, NORM_L1, or NORM_L2, respectively; or so that¥[¥min _I ¥texttt{dst} (I)= ¥texttt{alpha} , ¥, ¥, ¥max _I ¥texttt{dst} (I)= ¥texttt{beta}¥]when normType=NORM_MINMAX (for dense arrays only). The optional mask specifies a sub-array to be normalized. This means that the norm or min-n-max are calculated over the sub-array, and then this sub-array is modified to be normalized. If you want to only use the mask to calculate the norm or min-max but modify the whole array, you can use norm and Mat::convertTo.In case of sparse matrices, only the non-zero values are analyzed and transformed. Because of this, the range transformation for sparse matrices is not allowed since it can shift the zero level.Possible usage with some positive example data:vector<double> positiveData = { 2.0, 8.0, 10.0	7								
core_minMaxLoc1	CVAPI(ExceptionStatus) core_minMaxLoc1(cv::_InputArray* src, double* minVal, double* maxVal) { BEGIN_WRAP cv::minMaxLoc(*src, minVal, maxVal); END_WRAP }	cv::minMaxLoc()	group__core__array.html	Finds the global minimum and maximum in an array.	src : input single-channel array. minVal : pointer to the returned minimum value; NULL is used if not required. maxVal : pointer to the returned maximum value; NULL is used if not required. minLoc : pointer to the returned minimum location (in 2D case); NULL is used if not required. maxLoc : pointer to the returned maximum location (in 2D case); NULL is used if not required. mask : optional mask used to select a sub-array.	The function cv::minMaxLoc finds the minimum and maximum element values and their positions. The extremums are searched across the whole array or, if mask is not an empty array, in the specified array region.The function do not work with multi-channel arrays. If you need to find minimum or maximum elements across all the channels, use Mat::reshape first to reinterpret the array as single-channel. Or you may extract the particular channel using either extractImageCOI , or mixChannels , or split .See alsomax, min, compare, inRange, extractImageCOI, mixChannels, split, Mat::reshapeExamples: samples/cpp/image_alignment.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, and samples/dnn/openpose.cpp.	3								
core_minMaxLoc2	CVAPI(ExceptionStatus) core_minMaxLoc2(cv::_InputArray* src, double* minVal, double* maxVal, MyCvPoint* minLoc, MyCvPoint* maxLoc, cv::_InputArray* mask) { BEGIN_WRAP cv::InputArray maskVal = entity(mask); cv::Point minLoc0, maxLoc0; cv::minMaxLoc(*src, minVal, maxVal, &minLoc0, &maxLoc0, maskVal); *minLoc = c[minLoc0]; *maxLoc = c[maxLoc0]; END_WRAP }	cv::minMaxLoc()	group__core__array.html	Finds the global minimum and maximum in an array.	src : input single-channel array. minVal : pointer to the returned minimum value; NULL is used if not required. maxVal : pointer to the returned maximum value; NULL is used if not required. minLoc : pointer to the returned minimum location (in 2D case); NULL is used if not required. maxLoc : pointer to the returned maximum location (in 2D case); NULL is used if not required. mask : optional mask used to select a sub-array.	The function cv::minMaxLoc finds the minimum and maximum element values and their positions. The extremums are searched across the whole array or, if mask is not an empty array, in the specified array region.The function do not work with multi-channel arrays. If you need to find minimum or maximum elements across all the channels, use Mat::reshape first to reinterpret the array as single-channel. Or you may extract the particular channel using either extractImageCOI , or mixChannels , or split .See alsomax, min, compare, inRange, extractImageCOI, mixChannels, split, Mat::reshapeExamples: samples/cpp/image_alignment.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, and samples/dnn/openpose.cpp.	6								
core_minMaxIdx1	CVAPI(ExceptionStatus) core_minMaxIdx1(cv::_InputArray* src, double* minVal, double* maxVal) { BEGIN_WRAP cv::minMaxIdx(*src, minVal, maxVal); END_WRAP }	cv::minMaxIdx()	group__core__array.html	Finds the global minimum and maximum in an array.	src : input single-channel array. minVal : pointer to the returned minimum value; NULL is used if not required. maxVal : pointer to the returned maximum value; NULL is used if not required. minIdx : pointer to the returned minimum location (in nD case); NULL is used if not required; Otherwise, it must point to an array of src.dims elements, the coordinates of the minimum element in each dimension are stored there sequentially. maxIdx : pointer to the returned maximum location (in nD case). NULL is used if not required. mask : specified array region	The function cv::minMaxIdx finds the minimum and maximum element values and their positions. The extremums are searched across the whole array or, if mask is not an empty array, in the specified array region. The function does not work with multi-channel arrays. If you need to find minimum or maximum elements across all the channels, use Mat::reshape first to reinterpret the array as single-channel. Or you may extract the particular channel using either extractImageCOI , or mixChannels , or split . In case of a sparse matrix, the minimum is found among non-zero elements only.NoteWhen minIdx is not NULL, it must have at least 2 elements (as well as maxIdx), even if src is a single-row or single-column matrix. In OpenCV (following MATLAB) each array has at least 2 dimensions, i.e. single-column matrix is Mx1 matrix (and therefore minIdx/maxIdx will be (i1,0)/(i2,0)) and single-row matrix is 1xN matrix (and therefore minIdx/maxIdx will	3								

core_minMaxIdx2	CVAPI(ExceptionStatus) core_minMaxIdx2(cv::_InputArray* src, double* minVal, double* maxVal, int* minIdx, int* maxIdx, cv::_InputArray* mask) { BEGIN_WRAP cv::InputArray maskVal = entity(mask); cv::minMaxIdx(*src, minVal, maxVal, minIdx, maxIdx, maskVal); END_WRAP }	cv::minMaxIdx()	group__core__array.html	Finds the global minimum and maximum in an array.	src : input single-channel array. minVal : pointer to the returned minimum value; NULL is used if not required. maxVal : pointer to the returned maximum value; NULL is used if not required. minIdx : pointer to the returned minimum location (in nD case); NULL is used if not required; Otherwise, it must point to an array of src.dims elements, the coordinates of the minimum element in each dimension are stored there sequentially. maxIdx : pointer to the returned maximum location (in nD case). NULL is used if not required. mask : specified array region	The function cv::minMaxIdx finds the minimum and maximum element values and their positions. The extremums are searched across the whole array or, if mask is not an empty array, in the specified array region. The function does not work with multi-channel arrays. If you need to find minimum or maximum elements across all the channels, use Mat::reshape first to reinterpret the array as single-channel. Or you may extract the particular channel using either extractImageCOI , or mixChannels , or split . In case of a sparse matrix, the minimum is found among non-zero elements only.NoteWhen minIdx is not NULL, it must have at least 2 elements (as well as maxIdx), even if src is a single-row or single-column matrix. In OpenCV (following MATLAB) each array has at least 2 dimensions, i.e. single-column matrix is Mx1 matrix (and therefore minIdx/maxIdx will be (i1,0)/(i2,0)) and single-row matrix is 1xN matrix (and therefore minIdx/maxIdx will be (0,i1)/(0,i2)).	6								
core_reduce	CVAPI(ExceptionStatus) core_reduce(cv::_InputArray* src, cv::_OutputArray* dst, int dim, int rtype, int dtype) { BEGIN_WRAP cv::reduce(*src, *dst, dim, rtype, dtype); END_WRAP }	cv::reduce()	group__core__array.html	Reduces a matrix to a vector.	src : input 2D matrix. dst : output vector. Its size and type is defined by dim and dtype parameters. dim : dimension index along which the matrix is reduced. 0 means that the matrix is reduced to a single row. 1 means that the matrix is reduced to a single column. rtype : reduction operation that could be one of ReduceTypes dtype : when negative, the output vector will have the same type as the input matrix, otherwise, its type will be CV_MAKE_TYPE(CV_MAT_DEPTH(dtype), src.channels()).	The function reduce reduces the matrix to a vector by treating the matrix rows/columns as a set of 1D vectors and performing the specified operation on the vectors until a single row/column is obtained. For example, the function can be used to compute horizontal and vertical projections of a raster image. In case of REDUCE_MAX and REDUCE_MIN , the output image should have the same type as the source one. In case of REDUCE_SUM and REDUCE_AVG , the output may have a larger element bit-depth to preserve accuracy. And multi-channel arrays are also supported in these two reduction modes.The following code demonstrates its usage for a single channel matrix.Mat m = (Mat_<uchar>(3,2) << 1,2,3,4,5,6); Mat col_sum, row_sum; reduce(m, col_sum, 0, REDUCE_SUM, CV_32F); reduce(m, row_sum, 1, REDUCE_SUM, CV_32F); /* m = [1, 2; 3, 4; 5, 6] col_sum = [9, 12] row_sum = [3; 7; 11] */fragmentAnd the following code demonstrates its usage for a	5								
core_merge	CVAPI(ExceptionStatus) core_merge(cv::Mat** mv, uint32_t count, cv::Mat* dst) { BEGIN_WRAP std::vector<cv::Mat> vec(static_cast<size_t>(count)); for (uint32_t i = 0; i < count; i++) vec[i] = *mv[i]; cv::merge(vec, *dst); END_WRAP }	cv::merge()	group__core__array.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	mv : input vector of matrices to be merged; all the matrices in mv must have the same size and the same depth. dst : output array of the same size and the same depth as mv[0]; The number of channels will be the total number of channels in the matrix array.		2								
core_split	CVAPI(ExceptionStatus) core_split(cv::Mat* src, std::vector<cv::Mat>* mv) { BEGIN_WRAP cv::split(*src, *mv); END_WRAP }	cv::split()	group__core__array.html	Divides a multi-channel array into several single-channel arrays.	src : input multi-channel array. mvbegin : output array; the number of arrays must match src.channels(); the arrays themselves are reallocated, if needed.	The function cv::split splits a multi-channel array into separate single-channel arrays:¥[¥textttl{mv} [c][I] = ¥textttl{src} [I]_c¥]If you need to extract a single channel or do some other sophisticated channel permutation, use mixChannels .The following example demonstrates how to split a 3-channel matrix into 3 single channel matrices.char d[] = {1,2,3,4,5,6,7,8,9,10,11,12}; Mat m(2, 2, CV_8UC3, d); Mat channels[3]; split(m, channels); /* channels[0] = [1, 4; 7, 10] channels[1] = [2, 5; 8, 11] channels[2] = [3, 6; 9, 12] */fragmentSee alsomerge, mixChannels, cvtColorExamples: samples/cpp/tutorial_code/videoio/video-write/video-write.cpp.	2								
core_mixChannels	CVAPI(ExceptionStatus) core_mixChannels(cv::Mat** src, uint32_t nsrcs, cv::Mat** dst, uint32_t ndsts, int* fromTo, uint32_t npairs) { BEGIN_WRAP std::vector<cv::Mat> srcVec(static_cast<size_t>(nsrcs)); std::vector<cv::Mat> dstVec(static_cast<size_t>(ndsts)); for (uint32_t i = 0; i < nsrcs; i++) srcVec[i] = *(src[i]); for (uint32_t i = 0; i < ndsts; i++) dstVec[i] = *(dst[i]); cv::mixChannels(srcVec, dstVec, fromTo, npairs); END_WRAP }	cv::mixChannels()	group__core__array.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	src : input array or vector of matrices; all of the matrices must have the same size and the same depth. dst : output array or vector of matrices; all the matrices must be allocated; their size and depth must be the same as in src[0]. fromTo : array of index pairs specifying which channels are copied and where; fromTo[k*2] is a 0-based index of the input channel in src, fromTo[k*2+1] is an index of the output channel in dst; the continuous channel numbering is used: the first input image channels are indexed from 0 to src[0].channels()-1, the second input image channels are indexed from src[0].channels() to src[0].channels() + src[1].channels()-1, and so on, the same scheme is used for the output image channels; as a special case, when fromTo[k*2] is negative, the corresponding output channel is filled with zero . npairs : number of index pairs in fromTo.		4								
core_extractChannel	CVAPI(ExceptionStatus) core_extractChannel(cv::_InputArray* src, cv::_OutputArray* dst, int coi) { BEGIN_WRAP cv::extractChannel(*src, *dst, coi); END_WRAP }	cv::extractChannel()	group__core__array.html	Extracts a single channel from src (coi is 0-based index)	src : input array dst : output array coi : index of channel to extract	See alsomixChannels, splitExamples: samples/dnn/colorization.cpp.	3								
core_insertChannel	CVAPI(ExceptionStatus) core_insertChannel(cv::_InputArray* src, cv::_InputOutputArray* dst, int coi) { BEGIN_WRAP cv::insertChannel(*src, *dst, coi); END_WRAP }	cv::insertChannel()	group__core__array.html	Inserts a single channel to dst (coi is 0-based index)	src : input array dst : output array coi : index of channel for insertion	See alsomixChannels, merge	3								
core_flip	CVAPI(ExceptionStatus) core_flip(cv::_InputArray* src, cv::_OutputArray* dst, int flipCode) { BEGIN_WRAP cv::flip(*src, *dst, flipCode); END_WRAP }	cv::flip()	group__core__array.html	Flips a 2D array around vertical, horizontal, or both axes.	src : input array. dst : output array of the same size and type as src. flipCode : a flag to specify how to flip the array; 0 means flipping around the x-axis and positive value (for example, 1) means flipping around y-axis. Negative value (for example, -1) means flipping around both axes.	The function cv::flip flips the array in one of three different ways (row and column indices are 0-based):¥[¥textttl{dst} _{ij} = ¥leftV{ ¥begin{array}{l l} ¥textttl{src} _{¥textttl{src.rows}-i-1,j} & if¥; ¥textttl{flipCode} = 0 ¥¥ ¥textttl{src} _{i, ¥textttl{src.cols} -j-1} & if¥; ¥textttl{flipCode} > 0 ¥¥ ¥textttl{src} _{ ¥textttl{src.rows} -i-1, ¥textttl{src.cols} -j-1} & if¥; ¥textttl{flipCode} < 0 ¥¥ ¥end{array} ¥right.¥]The example scenarios of using the function are the following: Vertical flipping of the image (flipCode == 0) to switch between top-left and bottom-left image origin. This is a typical operation in video processing on Microsoft Windows* OS. Horizontal flipping of the image with the subsequent horizontal shift and absolute difference calculation to check for a vertical-axis symmetry (flipCode > 0). Simultaneous horizontal and vertical flipping of the image with the subsequent shift and absolute difference calculation to check for a central symmetry (flipCode < 0). Reversing the order of point arrays (flipCode > 0 or flipCode == 0 flip	3								
core_rotate	CVAPI(ExceptionStatus) core_rotate(cv::_InputArray *src, cv::_OutputArray *dst, int rotateCode) { BEGIN_WRAP cv::rotate(*src, *dst, rotateCode); END_WRAP }	cv::rotate()	group__core__array.html	Rotates a 2D array in multiples of 90 degrees. The function cv::rotate rotates the array in one of three different ways: Rotate by 90 degrees clockwise (rotateCode = ROTATE_90_CLOCKWISE). Rotate by 180 degrees clockwise (rotateCode = ROTATE_180). Rotate by 270 degrees clockwise (rotateCode = ROTATE_90_COUNTERCLOCKWISE).	src : input array. dst : output array of the same type as src. The size is the same with ROTATE_180, and the rows and cols are switched for ROTATE_90_CLOCKWISE and ROTATE_90_COUNTERCLOCKWISE. rotateCode : an enum to specify how to rotate the array; see the enum RotateFlags	See alsotranspose , repeat , completeSymm, flip, RotateFlags	3								
core_repeat1	CVAPI(ExceptionStatus) core_repeat1(cv::_InputArray * src, int ny, int nx, cv::_OutputArray* dst) { BEGIN_WRAP cv::repeat(*src, ny, nx, *dst); END_WRAP }	cv::repeat()	group__core__array.html	Fills the output array with repeated copies of the input array.	src : input array to replicate. ny : Flag to specify how many times the src is repeated along the vertical axis. nx : Flag to specify how many times the src is repeated along the horizontal axis. dst : output array of the same type as src.	The function cv::repeat duplicates the input array one or more times along each of the two axes:¥[¥textttl{dst} _{ij} = ¥textttl{src} _{¥fmod src.rows, ¥, ¥fmod src.cols }¥]The second variant of the function is more convenient to use with MatrixExpressions.See alsocv::reduce	4								
core_repeat2	CVAPI(ExceptionStatus) core_repeat2(cv::Mat* src, int ny, int nx, cv::Mat** returnValue) { BEGIN_WRAP const cv::Mat ret = cv::repeat(*src, ny, nx); *returnValue = new cv::Mat(ret); END_WRAP }	cv::repeat()	group__core__array.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	src : input array to replicate. ny : Flag to specify how many times the src is repeated along the vertical axis. nx : Flag to specify how many times the src is repeated along the horizontal axis.		3								

core_hconcat1	CVAPI(ExceptionStatus) core_hconcat1(cv::Mat** src, uint32_t nsrc, cv::_OutputArray* dst) { BEGIN_WRAP std::vector<cv::Mat> srcVec(static_cast<size_t>(nsrc)); for (uint32_t i = 0; i < nsrc; i++) srcVec[i] = *(src[i]); cv::hconcat(&srcVec[0], nsrc, *dst); END_WRAP }	cv::hconcat()	group__core__array.html	Applies horizontal concatenation to given matrices.	src : input array or vector of matrices. all of the matrices must have the same number of rows and the same depth. nsrc : number of matrices in src. dst : output array. It has the same number of rows and depth as the src, and the sum of cols of the src.	The function horizontally concatenates two or more cv::Mat matrices (with the same number of rows).cv::Mat matArray[] = { cv::Mat(4, 1, CV_8UC1, cv::Scalar(1)), cv::Mat(4, 1, CV_8UC1, cv::Scalar(2)), cv::Mat(4, 1, CV_8UC1, cv::Scalar(3)),};cv::Mat out;cv::hconcat(matArray, 3, out);//out://[1, 2, 3;// 1, 2, 3;// 1, 2, 3;// 1, 2, 3]fragmentSee alsocv::vconcat(const Mat*, size_t, OutputArray), cv::vconcat(InputArrayOfArrays, OutputArray) and cv::vconcat(InputArray, InputArray, OutputArray)Examples: samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_dis	3								
core_hconcat2	CVAPI(ExceptionStatus) core_hconcat2(cv::_InputArray* src1, cv::_InputArray* src2, cv::_OutputArray* dst) { BEGIN_WRAP cv::hconcat(*src1, *src2, *dst); END_WRAP }	cv::hconcat()	group__core__array.html	Applies horizontal concatenation to given matrices.	src : input array or vector of matrices. all of the matrices must have the same number of rows and the same depth. nsrc : number of matrices in src. dst : output array. It has the same number of rows and depth as the src, and the sum of cols of the src.	The function horizontally concatenates two or more cv::Mat matrices (with the same number of rows).cv::Mat matArray[] = { cv::Mat(4, 1, CV_8UC1, cv::Scalar(1)), cv::Mat(4, 1, CV_8UC1, cv::Scalar(2)), cv::Mat(4, 1, CV_8UC1, cv::Scalar(3)),};cv::Mat out;cv::hconcat(matArray, 3, out);//out://[1, 2, 3;// 1, 2, 3;// 1, 2, 3]fragmentSee alsocv::vconcat(const Mat*, size_t, OutputArray), cv::vconcat(InputArrayOfArrays, OutputArray) and cv::vconcat(InputArray, InputArray, OutputArray)Examples: samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_dis	3								
core_vconcat1	CVAPI(ExceptionStatus) core_vconcat1(cv::Mat** src, uint32_t nsrc, cv::_OutputArray* dst) { BEGIN_WRAP std::vector<cv::Mat> srcVec(static_cast<size_t>(nsrc)); for (uint32_t i = 0; i < nsrc; i++) srcVec[i] = *(src[i]); cv::vconcat(&srcVec[0], nsrc, *dst); END_WRAP }	cv::vconcat()	group__core__array.html	Applies vertical concatenation to given matrices.	src : input array or vector of matrices. all of the matrices must have the same number of cols and the same depth. nsrc : number of matrices in src. dst : output array. It has the same number of cols and depth as the src, and the sum of rows of the src.	The function vertically concatenates two or more cv::Mat matrices (with the same number of cols).cv::Mat matArray[] = { cv::Mat(1, 4, CV_8UC1, cv::Scalar(1)), cv::Mat(1, 4, CV_8UC1, cv::Scalar(2)), cv::Mat(1, 4, CV_8UC1, cv::Scalar(3)),};cv::Mat out;cv::vconcat(matArray, 3, out);//out://[1, 1, 1, 1;// 2, 2, 2, 2;// 3, 3, 3, 3]fragmentSee alsocv::hconcat(const Mat*, size_t, OutputArray), cv::hconcat(InputArrayOfArrays, OutputArray) and cv::hconcat(InputArray, InputArray,	3								
core_vconcat2	CVAPI(ExceptionStatus) core_vconcat2(cv::_InputArray* src1, cv::_InputArray* src2, cv::_OutputArray* dst) { BEGIN_WRAP cv::vconcat(*src1, *src2, *dst); END_WRAP }	cv::vconcat()	group__core__array.html	Applies vertical concatenation to given matrices.	src : input array or vector of matrices. all of the matrices must have the same number of cols and the same depth. nsrc : number of matrices in src. dst : output array. It has the same number of cols and depth as the src, and the sum of rows of the src.	The function vertically concatenates two or more cv::Mat matrices (with the same number of cols).cv::Mat matArray[] = { cv::Mat(1, 4, CV_8UC1, cv::Scalar(1)), cv::Mat(1, 4, CV_8UC1, cv::Scalar(2)), cv::Mat(1, 4, CV_8UC1, cv::Scalar(3)),};cv::Mat out;cv::vconcat(matArray, 3, out);//out://[1, 1, 1, 1;// 2, 2, 2, 2;// 3, 3, 3, 3]fragmentSee alsocv::hconcat(const Mat*, size_t, OutputArray), cv::hconcat(InputArrayOfArrays, OutputArray) and cv::hconcat(InputArray, InputArray,	3								
core_bitwise_and	CVAPI(ExceptionStatus) core_bitwise_and(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::bitwise_and(*src1, *src2, *dst, entity(mask)); END_WRAP }	cv::bitwise_and()	group__core__array.html	computes bitwise conjunction of the two arrays (dst = src1 & src2) Calculates the per-element bit-wise conjunction of two arrays or an array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array that has the same size and type as the input arrays. mask : optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.	The function cv::bitwise_and calculates the per-element bit-wise logical conjunction for: Two arrays when src1 and src2 have the same size:¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥wedge ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥wedge ¥texttt{src2} ¥quad ¥texttt{if mask} (I) ¥neq 0¥]A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} ¥wedge ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.Examples:	4								
core_bitwise_or	CVAPI(ExceptionStatus) core_bitwise_or(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::bitwise_or(*src1, *src2, *dst, entity(mask)); END_WRAP }	cv::bitwise_or()	group__core__array.html	Calculates the per-element bit-wise disjunction of two arrays or an array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array that has the same size and type as the input arrays. mask : optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.	The function cv::bitwise_or calculates the per-element bit-wise logical disjunction for: Two arrays when src1 and src2 have the same size:¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥vee ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥vee ¥texttt{src2} ¥quad ¥texttt{if mask} (I) ¥neq 0¥]A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} ¥vee ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the second and third cases above, the scalar is first converted to the array type.	4								
core_bitwise_xor	CVAPI(ExceptionStatus) core_bitwise_xor(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::bitwise_xor(*src1, *src2, *dst, entity(mask)); END_WRAP }	cv::bitwise_xor()	group__core__array.html	Calculates the per-element bit-wise "exclusive or" operation on two arrays or an array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array that has the same size and type as the input arrays. mask : optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.	The function cv::bitwise_xor calculates the per-element bit-wise logical "exclusive-or" operation for: Two arrays when src1 and src2 have the same size:¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥oplus ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]An array and a scalar when src2 is constructed from Scalar or has the same number of elements as src1.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥oplus ¥texttt{src2} ¥quad ¥texttt{if mask} (I) ¥neq 0¥]A scalar and an array when src1 is constructed from Scalar or has the same number of elements as src2.channels():¥[¥texttt{dst} (I) = ¥texttt{src1} ¥oplus ¥texttt{src2} (I) ¥quad ¥texttt{if mask} (I) ¥neq 0¥]In case of floating-point arrays, their machine-specific bit representations (usually IEEE754-compliant) are used for the operation. In case of multi-channel arrays, each channel is processed independently. In the 2nd and 3rd cases above, the scalar is first converted to the array type.	4								
core_bitwise_not	CVAPI(ExceptionStatus) core_bitwise_not(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::bitwise_not(*src, *dst, entity(mask)); END_WRAP }	cv::bitwise_not()	group__core__array.html	Inverts every bit of an array.	src : input array. dst : output array that has the same size and type as the input array. mask : optional operation mask, 8-bit single channel array, that specifies elements of the output array to be changed.	The function cv::bitwise_not calculates per-element bit-wise inversion of the input array:¥[¥texttt{dst} (I) = ¥neg ¥texttt{src} (I)¥]In case of a floating-point input array, its machine-specific bit representation (usually IEEE754-compliant) is used for the operation. In case of multi-channel arrays, each channel is processed independently.Examples: samples/cpp/camshiftdemo.cpp.	3								
core_absdiff	CVAPI(ExceptionStatus) core_absdiff(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst) { BEGIN_WRAP cv::absdiff(*src1, *src2, *dst); END_WRAP }	cv::absdiff()	group__core__array.html	Calculates the per-element absolute difference between two arrays or between an array and a scalar.	src1 : first input array or a scalar. src2 : second input array or a scalar. dst : output array that has the same size and type as input arrays.	The function cv::absdiff calculates: Absolute difference between two arrays when they have the same size and type:¥[¥texttt{dst}(I) = ¥texttt{saturate} (	¥texttt{src1}(I) - ¥texttt{src2}(I) |)¥]Absolute difference between an array and a scalar when the second array is constructed from Scalar or has as many elements as the number of channels in src1:¥[¥texttt{dst}(I) = ¥texttt{saturate} (	¥texttt{src1}(I) - ¥texttt{src2} |)¥]Absolute difference between a scalar and an array when the first array is constructed from Scalar or has as many elements as the number of channels in src2:¥[¥texttt{dst}(I) = ¥texttt{saturate} (	¥texttt{src1} - ¥texttt{src2}(I) |)¥]where I is a multi-dimensional index of array elements. In case of multi-channel arrays, each channel is processed independently.NoteSaturation is not applied when the arrays have the depth CV_32S. You may even get a negative value in the case of overflow.See	3								
core_copyTo	CVAPI(ExceptionStatus) core_copyTo(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::copyTo(*src, *dst, entity(mask)); END_WRAP }	cv::copyTo()	group__core__array.html	This is an overloaded member function, provided for convenience (python) Copies the matrix to another one. When the operation mask is specified, if the Mat::create call shown above reallocates the matrix, the newly allocated matrix is initialized with all zeros before copying the data.	src : source matrix. dst : Destination matrix. If it does not have a proper size or type before the operation, it is reallocated. mask : Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels.		3								

core_inRange_InputArray	CVAPI(ExceptionStatus) core_inRange_InputArray(cv::_InputArray *src, cv::_InputArray *lowerb, cv::_InputArray *upperb, cv::_OutputArray *dst) { BEGIN_WRAP cv::inRange(*src, *lowerb, *upperb, *dst); END_WRAP }	cv::inRange()	group__core__array.html	Checks if array elements lie between the elements of two other arrays.	src : first input array. lowerb : inclusive lower boundary array or a scalar. upperb : inclusive upper boundary array or a scalar. dst : output array of the same size as src and CV_8U type.	The function checks the range as follows:For every element of a single-channel input array:¥[¥texttt{dst} (I)= ¥texttt{lowerb} (I)_0 ¥leq ¥texttt{src} (I)_0 ¥leq ¥texttt{upperb} (I)_0¥] For two-channel arrays:¥[¥texttt{dst} (I)= ¥texttt{lowerb} (I)_0 ¥leq ¥texttt{src} (I)_0 ¥leq ¥texttt{upperb} (I)_0 ¥land ¥texttt{lowerb} (I)_1 ¥leq ¥texttt{src} (I)_1 ¥leq ¥texttt{upperb} (I)_1¥] and so forth.That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.Examples: samples/cpp/camshiftdemo.cpp.	4								
core_inRange_Scalar	CVAPI(ExceptionStatus) core_inRange_Scalar(cv::_InputArray *src, MyCvScalar lowerb, MyCvScalar upperb, cv::_OutputArray *dst) { BEGIN_WRAP cv::inRange(*src, cpp(lowerb), cpp(upperb), *dst); END_WRAP }	cv::inRange()	group__core__array.html	Checks if array elements lie between the elements of two other arrays.	src : first input array. lowerb : inclusive lower boundary array or a scalar. upperb : inclusive upper boundary array or a scalar. dst : output array of the same size as src and CV_8U type.	The function checks the range as follows:For every element of a single-channel input array:¥[¥texttt{dst} (I)= ¥texttt{lowerb} (I)_0 ¥leq ¥texttt{src} (I)_0 ¥leq ¥texttt{upperb} (I)_0¥] For two-channel arrays:¥[¥texttt{dst} (I)= ¥texttt{lowerb} (I)_0 ¥leq ¥texttt{src} (I)_0 ¥leq ¥texttt{upperb} (I)_0 ¥land ¥texttt{lowerb} (I)_1 ¥leq ¥texttt{src} (I)_1 ¥leq ¥texttt{upperb} (I)_1¥] and so forth.That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.Examples: samples/cpp/camshiftdemo.cpp.	4								
core_compare	CVAPI(ExceptionStatus) core_compare(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, int cmpop) { BEGIN_WRAP cv::compare(*src1, *src2, *dst, cmpop); END_WRAP }	cv::compare()	group__core__array.html	Performs the per-element comparison of two arrays or an array and scalar value.	src1 : first input array or a scalar; when it is an array, it must have a single channel. src2 : second input array or a scalar; when it is an array, it must have a single channel. dst : output array of type ref CV_8U that has the same size and the same number of channels as the input arrays. cmpop : a flag, that specifies correspondence between the arrays (cv::CmpTypes)	The function compares: Elements of two arrays when src1 and src2 have the same size:¥[¥texttt{dst} (I) = ¥texttt{src1} (I) ¥, ¥texttt{cmpop}¥, ¥texttt{src2} (I)¥]Elements of src1 with a scalar src2 when src2 is constructed from Scalar or has a single element:¥[¥texttt{dst} (I) = ¥texttt{src1}(I) ¥,¥texttt{cmpop}¥, ¥texttt{src2}¥]src1 with elements of src2 when src1 is constructed from Scalar or has a single element:¥[¥texttt{dst} (I) = ¥texttt{src1} ¥,¥texttt{cmpop}¥, ¥texttt{src2} (I)¥]When the comparison result is true, the corresponding element of output array is set to 255. The comparison operations can be replaced with the equivalent matrix expressions:Mat dst1 = src1 >= src2;Mat dst2 = src1 < 8;...fragmentSee alsocheckRange, min, max, threshold	4								
core_min1	CVAPI(ExceptionStatus) core_min1(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst) { BEGIN_WRAP cv::min(*src1, *src2, *dst); END_WRAP }	cv::min()	group__core__array.html	Calculates per-element minimum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1. dst : output array of the same size and type as src1.	The function cv::min calculates the per-element minimum of two arrays:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{value})¥]See alsomax, compare, inRange, minMaxLoc	3								
core_min_MatMat	CVAPI(ExceptionStatus) core_min_MatMat(cv::Mat* src1, cv::Mat* src2, cv::Mat* dst) { BEGIN_WRAP cv::min(*src1, *src2, *dst); END_WRAP }	cv::min()	group__core__array.html	Calculates per-element minimum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1. dst : output array of the same size and type as src1.	The function cv::min calculates the per-element minimum of two arrays:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{value})¥]See alsomax, compare, inRange, minMaxLoc	3								
core_min_MatDouble	CVAPI(ExceptionStatus) core_min_MatDouble(cv::Mat * src1, double src2, cv::Mat* dst) { BEGIN_WRAP cv::min(*src1, src2, *dst); END_WRAP }	cv::min()	group__core__array.html	Calculates per-element minimum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1. dst : output array of the same size and type as src1.	The function cv::min calculates the per-element minimum of two arrays:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥min (¥texttt{src1} (I), ¥texttt{value})¥]See alsomax, compare, inRange, minMaxLoc	3								
core_max1	CVAPI(ExceptionStatus) core_max1(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst) { BEGIN_WRAP cv::max(*src1, *src2, *dst); END_WRAP }	cv::max()	group__core__array.html	Calculates per-element maximum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1 . dst : output array of the same size and type as src1.	The function cv::max calculates the per-element maximum of two arrays:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{value})¥]See alsomin, compare, inRange, minMaxLoc, MatrixExpressions	3								
core_max_MatMat	CVAPI(ExceptionStatus) core_max_MatMat(cv::Mat *src1, const cv::Mat *src2, cv::Mat *dst) { BEGIN_WRAP cv::max(*src1, *src2, *dst); END_WRAP }	cv::max()	group__core__array.html	Calculates per-element maximum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1 . dst : output array of the same size and type as src1.	The function cv::max calculates the per-element maximum of two arrays:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{value})¥]See alsomin, compare, inRange, minMaxLoc, MatrixExpressions	3								
core_max_MatDouble	CVAPI(ExceptionStatus) core_max_MatDouble(cv::Mat *src1, double src2, cv::Mat *dst) { BEGIN_WRAP cv::max(*src1, src2, *dst); END_WRAP }	cv::max()	group__core__array.html	Calculates per-element maximum of two arrays or an array and a scalar.	src1 : first input array. src2 : second input array of the same size and type as src1 . dst : output array of the same size and type as src1.	The function cv::max calculates the per-element maximum of two arrays:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{src2} (I))¥]or array and a scalar:¥[¥texttt{dst} (I)= ¥max (¥texttt{src1} (I), ¥texttt{value})¥]See alsomin, compare, inRange, minMaxLoc, MatrixExpressions	3								
core_sqrt	CVAPI(ExceptionStatus) core_sqrt(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::sqrt(*src, *dst); END_WRAP }	cv::sqrt()	group__core__array.html	Calculates a square root of array elements.	src : input floating-point array. dst : output array of the same size and type as src.	The function cv::sqrt calculates a square root of each input array element. In case of multi-channel arrays, each channel is processed independently. The accuracy is approximately the same as of the built-in std::sqrt .	2								
core_pow_Mat	CVAPI(ExceptionStatus) core_pow_Mat(cv::_InputArray *src, double power, cv::_OutputArray *dst) { BEGIN_WRAP cv::pow(*src, power, *dst); END_WRAP }	cv::pow()	group__core__array.html	Raises every array element to a power.	src : input array. power : exponent of power. dst : output array of the same size and type as src.	The function cv::pow raises every element of the input array to power :¥[¥texttt{dst} (I) = ¥fork{¥texttt{src}(I)^{power}}{if ¥(¥texttt{power}¥) is integer}{|¥texttt{src}(I)|^{power}}{otherwise}¥]So, for a non-integer power exponent, the absolute values of input array elements are used. However, it is possible to get true values for negative values using some extra operations. In the example below, computing the 5th root of array src shows:Mat mask = src < 0;pow(src, 1./5, dst);subtract(Scalar::all(0), dst, dst, mask);fragmentFor some values of power, such as integer values, 0.5 and -0.5, specialized faster algorithms are used.Special values (NaN, Inf) are not handled.See alsosqrt, exp, log,	3								
core_exp_Mat	CVAPI(ExceptionStatus) core_exp_Mat(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::exp(*src, *dst); END_WRAP }	cv::exp()	group__core__array.html	Calculates the exponent of every array element.	src : input array. dst : output array of the same size and type as src.	The function cv::exp calculates the exponent of every element of the input array:¥[¥texttt{dst} [I] = e^{ src(I) }¥]The maximum relative error is about 7e-6 for single-precision input and less than 1e-10 for double-precision input. Currently, the function converts denormalized values to zeros on output. Special values (NaN, Inf) are not handled.See alsolog , cartToPolar , polarToCart , phase , pow , sqrt ,	2								
core_log_Mat	CVAPI(ExceptionStatus) core_log_Mat(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::log(*src, *dst); END_WRAP }	cv::log()	group__core.html	For example	q : a quaternion. assumeUnit : if QUAT_ASSUME_UNIT, q assume to be a unit quaternion and this function will save some computations.	Quatd q1(1,2,3,4);cout << log(q1) << endl;fragmentExamples: samples/cpp/polar_transforms.cpp, and samples/cpp/stitching_detailed.cpp.	2								
core_polarToCart	CVAPI(ExceptionStatus) core_polarToCart(cv::_InputArray* magnitude, cv::_InputArray* angle, cv::_OutputArray* x, cv::_OutputArray* y, int angleInDegrees) { BEGIN_WRAP cv::polarToCart(*magnitude, *angle, *x, *y, angleInDegrees != 0); END_WRAP }	cv::polarToCart()	group__core__array.html	Calculates x and y coordinates of 2D vectors from their magnitude and angle.	magnitude : input floating-point array of magnitudes of 2D vectors; it can be an empty matrix (=Mat()), in this case, the function assumes that all the magnitudes are =1; if it is not empty, it must have the same size and type as angle. angle : input floating-point array of angles of 2D vectors. x : output array of x-coordinates of 2D vectors; it has the same size and type as angle. y : output array of y-coordinates of 2D vectors; it has the same size and type as angle. angleInDegrees : when true, the input angles are measured in degrees, otherwise, they are measured in radians.	The function cv::polarToCart calculates the Cartesian coordinates of each 2D vector represented by the corresponding elements of magnitude and angle:¥[¥begin{array}{l} ¥texttt{x} (I) = ¥texttt{magnitude} (I) ¥cos (¥texttt{angle} (I)) ¥¥ ¥texttt{y} (I) = ¥texttt{magnitude} (I) ¥sin (¥texttt{angle} (I)) ¥¥ ¥end{array}¥]The relative accuracy of the estimated coordinates is about 1e-6.See alsocartToPolar, magnitude, phase, exp, log, pow, sqrt	5								

core_cartToPolar	CVAPI(ExceptionStatus) core_cartToPolar(cv::_InputArray* x, cv::_InputArray* y, cv::_OutputArray* magnitude, cv::_OutputArray* angle, int angleInDegrees) { BEGIN_WRAP cv::cartToPolar(*x, *y, *magnitude, *angle, angleInDegrees != 0); END_WRAP }	cv::cartToPolar()	group__core__array.html	Calculates the magnitude and angle of 2D vectors.	x : array of x-coordinates; this must be a single-precision or double-precision floating-point array. y : array of y-coordinates, that must have the same size and same type as x. magnitude : output array of magnitudes of the same size and type as x. angle : output array of angles that has the same size and type as x; the angles are measured in radians (from 0 to 2*Pi) or in degrees (0 to 360 degrees). angleInDegrees : a flag, indicating whether the angles are measured in radians (which is by default), or in degrees.	The function cv::cartToPolar calculates either the magnitude, angle, or both for every 2D vector (x(I),y(I)):¥[¥begin{array}{l} ¥texttt{magnitude} (I)= ¥sqrt{¥texttt{x}(I)^2+¥texttt{y}(I)^2} , ¥¥ ¥texttt{angle} (I)= ¥texttt{atan2} (¥texttt{y} (I), ¥texttt{x} (I))[¥cdot180 / ¥pi] ¥end{array}¥]The angles are calculated with accuracy about 0.3 degrees. For the point (0,0), the angle is set to 0.See alsoSobel, Scharr	5								
core_phase	CVAPI(ExceptionStatus) core_phase(cv::_InputArray* x, cv::_InputArray* y, cv::_OutputArray* angle, int angleInDegrees) { BEGIN_WRAP cv::phase(*x, *y, *angle, angleInDegrees != 0); END_WRAP }	cv::phase()	group__core__array.html	Calculates the rotation angle of 2D vectors.	x : input floating-point array of x-coordinates of 2D vectors. y : input array of y-coordinates of 2D vectors; it must have the same size and the same type as x. angle : output array of vector angles; it has the same size and same type as x . angleInDegrees : when true, the function calculates the angle in degrees, otherwise, they are measured in radians.	The function cv::phase calculates the rotation angle of each 2D vector that is formed from the corresponding elements of x and y :¥[¥texttt{angle} (I) = ¥texttt{atan2} (¥texttt{y} (I), ¥texttt{x} (I))¥]The angle estimation accuracy is about 0.3 degrees. When x(I)=y(I)=0 , the corresponding angle(I) is set to 0.	4								
core_magnitude_Mat	CVAPI(ExceptionStatus) core_magnitude_Mat(cv::_InputArray* x, cv::_InputArray* y, cv::_OutputArray* magnitude) { BEGIN_WRAP cv::magnitude(*x, *y, *magnitude); END_WRAP }	cv::magnitude()	group__core__array.html	Calculates the magnitude of 2D vectors.	x : floating-point array of x-coordinates of the vectors. y : floating-point array of y-coordinates of the vectors; it must have the same size as x. magnitude : output array of the same size and type as x.	The function cv::magnitude calculates the magnitude of 2D vectors formed from the corresponding elements of x and y arrays:¥[¥texttt{dst} (I) = ¥sqrt{¥texttt{x}(I)^2 + ¥texttt{y}(I)^2}¥]See alsocartToPolar, polarToCart, phase, sqrtExamples: samples/cpp/polar_transforms.cpp.	3								
core_checkRange	CVAPI(ExceptionStatus) core_checkRange(cv::_InputArray* a, int quiet, MyCvPoint* pos, double minVal, double maxVal, int* returnValue) { BEGIN_WRAP cv::Point pos0; *returnValue = cv::checkRange(*a, quiet != 0, &pos0, minVal, maxVal); *pos = c(pos0); END_WRAP }	cv::checkRange()	group__core__array.html	Checks every element of an input array for invalid values.	a : input array. quiet : a flag, indicating whether the functions quietly return false when the array elements are out of range or they throw an exception. pos : optional output parameter, when not NULL, must be a pointer to array of src.dims elements. minVal : inclusive lower boundary of valid values range. maxVal : exclusive upper boundary of valid values range.	The function cv::checkRange checks that every array element is neither NaN nor infinite. When minVal > -DBL_MAX and maxVal < DBL_MAX, the function also checks that each value is between minVal and maxVal. In case of multi-channel arrays, each channel is processed independently. If some values are out of range, position of the first outlier is stored in pos (when pos != NULL). Then, the function either returns false (when quiet=true) or throws an exception.	5								
core_patchNaNs	CVAPI(ExceptionStatus) core_patchNaNs(cv::_InputOutputArray *a, double val) { BEGIN_WRAP cv::patchNaNs(*a, val); END_WRAP }	cv::patchNaNs()	group__core__array.html	converts NaNs to the given number	a : input/output matrix (CV_32F type). val : value to convert the NaNs		2								
core_gemm	CVAPI(ExceptionStatus) core_gemm(cv::_InputArray *src1, cv::_InputArray *src2, double alpha, cv::_InputArray *src3, double gamma, cv::_OutputArray *dst, int flags) { BEGIN_WRAP cv::gemm(*src1, *src2, alpha, *src3, gamma, *dst, flags); END_WRAP }	cv::gemm()	group__core__array.html	Performs generalized matrix multiplication.	src1 : first multiplied input matrix that could be real(CV_32FC1, CV_64FC1) or complex(CV_32FC2, CV_64FC2). src2 : second multiplied input matrix of the same type as src1. alpha : weight of the matrix product. src3 : third optional delta matrix added to the matrix product; it should have the same type as src1 and src2. beta : weight of src3. dst : output matrix; it has the proper size and the same type as input matrices. flags : operation flags (cv::GemmFlags)	The function cv::gemm performs generalized matrix multiplication similar to the gemm functions in BLAS level 3. For example, gemm(src1, src2, alpha, src3, beta, dst, GEMM_1_T + GEMM_3_T) corresponds to¥[¥texttt{dst} = ¥texttt{alpha} ¥cdot ¥texttt{src1} ^T ¥cdot ¥texttt{src2} + ¥texttt{beta} ¥cdot ¥texttt{src3} ^T¥]In case of complex (two-channel) data, performed a complex matrix multiplication.The function can be replaced with a matrix expression. For example, the above call can be replaced with:dst = alpha*src1.t() *src2 + beta*src3.t();fragmentiSee alsomulTransposed , transformExamples: samples/cpp/image_alignment.cpp.	7								
core_mulTransposed	CVAPI(ExceptionStatus) core_mulTransposed(cv::_InputArray *src, cv::_OutputArray *dst, int aTa, cv::_InputArray *delta, double scale, int dtype) { BEGIN_WRAP cv::mulTransposed(*src, *dst, aTa != 0, entity(delta), scale, dtype); END_WRAP }	cv::mulTransposed()	group__core__array.html	Calculates the product of a matrix and its transposition.	src : input single-channel matrix. Note that unlike gemm, the function can multiply not only floating-point matrices. dst : output square matrix. aTa : Flag specifying the multiplication ordering. See the description below. delta : Optional delta matrix subtracted from src before the multiplication. When the matrix is empty (delta==noArray()), it is assumed to be zero, that is, nothing is subtracted. If it has the same size as src , it is simply subtracted. Otherwise, it is "repeated" (see repeat) to cover the full src and then subtracted. Type of the delta matrix, when it is not empty, must be the same as the type of created output matrix. See the dtype parameter description below. scale : Optional scale factor for the matrix product. dtype : Optional type of the output matrix. When it is negative, the output matrix will have the same type as src . Otherwise, it will be type=CV_MAT_DEPTH(dtype) that should be either CV_32F or CV_64F .	The function cv::mulTransposed calculates the product of src and its transposition:¥[¥texttt{dst} = ¥texttt{scale} (¥texttt{src} - ¥texttt{delta})^T (¥texttt{src} - ¥texttt{delta})¥]if aTa=true , and¥[¥texttt{dst} = ¥texttt{scale} (¥texttt{src} - ¥texttt{delta}) (¥texttt{src} - ¥texttt{delta})^T¥]otherwise. The function is used to calculate the covariance matrix. With zero delta, it can be used as a faster substitute for general matrix product A*B when B=A'See alsocalcCovarMatrix, gemm, repeat, reduce	6								
core_transpose	CVAPI(ExceptionStatus) core_transpose(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::transpose(*src, *dst); END_WRAP }	cv::transpose()	group__core__array.html	Transposes a matrix.	src : input array. dst : output array of the same type as src.	The function cv::transpose transposes the matrix src :¥[¥texttt{dst} (i,j) = ¥texttt{src} (j,i)¥]NoteNo complex conjugation is done in case of a complex matrix. It should be done separately if needed.Examples: samples/cpp/train_HOG.cpp.	2								
core_transform	CVAPI(ExceptionStatus) core_transform(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *m) { BEGIN_WRAP cv::transform(*src, *dst, *m); END_WRAP }	cv::transform()	group__core__array.html	Performs the matrix transformation of every array element.	src : input array that must have as many channels (1 to 4) as m.cols or m.cols-1. dst : output array of the same size and depth as src; it has as many channels as m.rows. m : transformation 2x2 or 2x3 floating-point matrix.	The function cv::transform performs the matrix transformation of every element of the array src and stores the results in dst :¥[¥texttt{dst} (I) = ¥texttt{m} ¥cdot ¥texttt{src} (I)¥](when m.cols=src.channels()), or¥[¥texttt{dst} (I) = ¥texttt{m} ¥cdot [¥texttt{src} (I); 1]¥](when m.cols=src.channels()+1)Every element of the N -channel array src is interpreted as N -element vector that is transformed using the M x N or M x (N+1) matrix m to M-element vector - the corresponding element of the output array dst .The function may be used for geometrical transformation of N -dimensional points, arbitrary linear color space transformation (such as various kinds of RGB to YUV transforms), shuffling the image channels, and so forth.See alsoperspectiveTransform, getAffineTransform, estimateAffine2D, warpAffine, warpPerspective	3								
core_perspectiveTransform	CVAPI(ExceptionStatus) core_perspectiveTransform(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *m) { BEGIN_WRAP cv::perspectiveTransform(*src, *dst, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								
core_perspectiveTransform_Mat	CVAPI(ExceptionStatus) core_perspectiveTransform_Mat(cv::Mat *src, cv::Mat *dst, cv::Mat *m) { BEGIN_WRAP cv::perspectiveTransform(*src, *dst, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								
core_perspectiveTransform_Point2f	CVAPI(ExceptionStatus) core_perspectiveTransform_Point2f(cv::Point2f *src, int srcLength, cv::Point2f *dst, int dstLength, cv::_InputArray *m) { BEGIN_WRAP const std::vector<cv::Point2f> srcVector(src, src + srcLength); std::vector<cv::Point2f> dstVector(dst, dst + dstLength); cv::perspectiveTransform(srcVector, dstVector, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								

core_perspectiveTransform_Point2d	CVAPI(ExceptionStatus) core_perspectiveTransform_Point2d(cv::Point2d *src, int srcLength, cv::Point2d *dst, int dstLength, cv::_InputArray *m) { BEGIN_WRAP const std::vector<cv::Point2d> srcVector(src, src + srcLength); std::vector<cv::Point2d> dstVector(dst, dst + dstLength); cv::perspectiveTransform(srcVector, dstVector, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								
core_perspectiveTransform_Point3f	CVAPI(ExceptionStatus) core_perspectiveTransform_Point3f(cv::Point3f *src, int srcLength, cv::Point3f *dst, int dstLength, cv::_InputArray *m) { BEGIN_WRAP const std::vector<cv::Point3f> srcVector(src, src + srcLength); std::vector<cv::Point3f> dstVector(dst, dst + dstLength); cv::perspectiveTransform(srcVector, dstVector, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								
core_perspectiveTransform_Point3d	CVAPI(ExceptionStatus) core_perspectiveTransform_Point3d(cv::Point3d *src, int srcLength, cv::Point3d *dst, int dstLength, cv::_InputArray *m) { BEGIN_WRAP const std::vector<cv::Point3d> srcVector(src, src + srcLength); std::vector<cv::Point3d> dstVector(dst, dst + dstLength); cv::perspectiveTransform(srcVector, dstVector, *m); END_WRAP }	cv::perspectiveTransform()	group__core__array.html	Performs the perspective matrix transformation of vectors.	src : input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed. dst : output array of the same size and type as src. m : 3x3 or 4x4 floating-point transformation matrix.	The function cv::perspectiveTransform transforms every element of src by treating it as a 2D or 3D vector, in the following way:¥[(x, y, z) ¥rightarrow (x'/w, y'/w, z'/w)¥]where¥[(x', y', z', w') = ¥texttt{mat} ¥cdot ¥begin{bmatrix} x & y & z & 1 ¥end{bmatrix}¥]and¥[w = ¥fork{w'}{if ¥(w' ¥neq 0¥)}{¥infty}{otherwise}¥]Here a 3D vector transformation is shown. In case of a 2D vector transformation, the z component is omitted.NoteThe function transforms a sparse set of 2D or 3D vectors. If you want to transform an image using perspective transformation, use warpPerspective . If you have an inverse problem, that is, you want to compute the most probable perspective transformation out of several pairs of corresponding points, you can use getPerspectiveTransform or findHomography .See alsotransform, warpPerspective, getPerspectiveTransform,	3								
core_completeSymm	CVAPI(ExceptionStatus) core_completeSymm(cv::_InputOutputArray *mtx, int lowerToUpper) { BEGIN_WRAP cv::completeSymm(*mtx, lowerToUpper != 0); END_WRAP }	cv::completeSymm()	group__core__array.html	Copies the lower or the upper half of a square matrix to its another half.	m : input-output floating-point square matrix. lowerToUpper : operation flag; if true, the lower half is copied to the upper half. Otherwise, the upper half is copied to the lower half.	The function cv::completeSymm copies the lower or the upper half of a square matrix to its another half. The matrix diagonal remains unchanged:¥(¥texttt{m}_{ij}=¥texttt{m}_{ij}¥) for ¥(i > j¥) if lowerToUpper=false ¥(¥texttt{m}_{ij}=¥texttt{m}_{ji}¥) for ¥(i < j¥) if lowerToUpper=trueSee alsoflip, transpose	2								
core_setIdentity	CVAPI(ExceptionStatus) core_setIdentity(cv::_InputOutputArray *mtx, MyCvScalar s) { BEGIN_WRAP cv::setIdentity(*mtx, cpp(s)); END_WRAP }	cv::setIdentity()	group__core__array.html	Initializes a scaled identity matrix.	mtx : matrix to initialize (not necessarily square). s : value to assign to diagonal elements.	The function cv::setIdentity initializes a scaled identity matrix:¥[¥texttt{mtx}(i,j)= ¥fork{¥texttt{value}}{ if ¥(i=j¥)}{0}{otherwise}¥]The function can also be emulated using the matrix initializers and the matrix expressions:Mat A = Mat::eye(4, 3, CV_32F)*5;// A will be set to [[5, 0, 0], [0, 5, 0], [0, 0, 5], [0, 0, 0]]fragmentSee alsoMat::zeros, Mat::ones, Mat::setTo, Mat::operator=Examples: samples/cpp/kalman.cpp.	2								
core_determinant	CVAPI(ExceptionStatus) core_determinant(cv::_InputArray *mtx, double *returnValue) { BEGIN_WRAP *returnValue = cv::determinant(*mtx); END_WRAP }	cv::determinant()	group__core__array.html	Returns the determinant of a square floating-point matrix.	mtx : input matrix that must have CV_32FC1 or CV_64FC1 type and square size.	The function cv::determinant calculates and returns the determinant of the specified matrix. For small matrices (mtx.cols=mtx.rows<=3), the direct method is used. For larger matrices, the function uses LU factorization with partial pivoting.For symmetric positively-determined matrices, it is also possible to use eigen decomposition to calculate the determinant.See alsotrace, invert, solve, eigen, MatrixExpressions	1								
core_trace	CVAPI(ExceptionStatus) core_trace(cv::_InputArray *mtx, MyCvScalar *returnValue) { BEGIN_WRAP *returnValue = c(cv::trace(*mtx)); END_WRAP }	cv::trace()	group__core__array.html	Returns the trace of a matrix.	mtx : input matrix.	The function cv::trace returns the sum of the diagonal elements of the matrix mtx .¥[¥mathrm{tr} (¥texttt{mtx}) = ¥sum_i ¥texttt{mtx}_{ii}¥]	1								
core_invert	CVAPI(ExceptionStatus) core_invert(cv::_InputArray *src, cv::_OutputArray *dst, int flags, double *returnValue) { BEGIN_WRAP *returnValue = cv::invert(*src, *dst, flags); END_WRAP }	cv::invert()	group__core__array.html	Finds the inverse or pseudo-inverse of a matrix.	src : input floating-point M x N matrix. dst : output matrix of N x M size and the same type as src. flags : inversion method (cv::DecompTypes)	The function cv::invert inverts the matrix src and stores the result in dst . When the matrix src is singular or non-square, the function calculates the pseudo-inverse matrix (the dst matrix) so that norm(src*dst - I) is minimal, where I is an identity matrix.In case of the DECOMP_LU method, the function returns non-zero value if the inverse has been successfully calculated and 0 if src is singular.In case of the DECOMP_SVD method, the function returns the inverse condition number of src (the ratio of the smallest singular value to the largest singular value) and 0 if src is singular. The SVD method calculates a pseudo-inverse matrix if src is singular.Similarly to DECOMP_LU, the method DECOMP_CHOLESKY works only with non-singular square matrices that should also be symmetrical and positively defined. In this case, the function stores the inverted matrix in dst and returns non-zero. Otherwise, it returns 0.See alsosolve,	3								
core_solve	CVAPI(ExceptionStatus) core_solve(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, int flags, int *returnValue) { BEGIN_WRAP *returnValue = cv::solve(*src1, *src2, *dst, flags); END_WRAP }	cv::solve()	group__core__array.html	Solves one or more linear systems or least-squares problems.	src1 : input matrix on the left-hand side of the system. src2 : input matrix on the right-hand side of the system. dst : output solution. flags : solution (matrix inversion) method (DecompTypes)	The function cv::solve solves a linear system or least-squares problem (the latter is possible with SVD or QR methods, or by specifying the flag DECOMP_NORMAL):¥[¥texttt{dst} = ¥arg ¥min _X ¥| ¥texttt{src1} ¥cdot ¥texttt{X} - ¥texttt{src2} ¥|¥]If DECOMP_LU or DECOMP_CHOLESKY method is used, the function returns 1 if src1 (or ¥(¥texttt{src1}^T¥texttt{src1}¥)) is non-singular. Otherwise, it returns 0. In the latter case, dst is not valid. Other methods find a pseudo-solution in case of a singular left-hand side part.NoteIf you want to find a unity-norm solution of an under-defined singular system ¥(¥texttt{src1}¥cdot¥texttt{dst}=0¥) , the function solve will not do the work. Use SVD::solveZ instead.See alsoinvert, SVD, eigen	4								
core_solveLP	CVAPI(ExceptionStatus) core_solveLP(cv::_InputArray *Func, cv::_InputArray *Constr, cv::_OutputArray *z, int *returnValue) { BEGIN_WRAP *returnValue = cv::solveLP(*Func, *Constr, *z); END_WRAP }	cv::solveLP()	group__core__optim.html	Solve given (non-integer) linear programming problem using the Simplex Algorithm (Simplex Method).	Func : This row-vector corresponds to ¥(c¥) in the LP problem formulation (see above). It should contain 32- or 64-bit floating point numbers. As a convenience, column-vector may be also submitted, in the latter case it is understood to correspond to ¥(c^T¥). Constr : m-by-n+1 matrix, whose rightmost column corresponds to ¥(b¥) in formulation above and the remaining to ¥(A¥). It should contain 32- or 64-bit floating point numbers. z : The solution will be returned here as a column-vector - it corresponds to ¥(x¥) in the formulation above. It will contain 64-bit floating point numbers.	What we mean here by "linear programming problem" (or LP problem, for short) can be formulated as:¥[¥mbox{Maximize } c¥cdot x¥¥ ¥mbox{Subject to:}¥¥ Ax¥leq b¥¥ x¥geq 0¥]Where ¥(¥) is fixed 1-by-n row-vector, ¥(A¥) is fixed m-by-n matrix, ¥(b¥) is fixed m-by-1 column vector and ¥(x¥) is an arbitrary n-by-1 column vector, which satisfies the constraints.Simplex algorithm is one of many algorithms that are designed to handle this sort of problems efficiently. Although it is not optimal in theoretical sense (there exist algorithms that can solve any problem written as above in polynomial time, while simplex method degenerates to exponential time for some special cases), it is well-studied, easy to implement and is shown to work well for real-life purposes.The particular implementation is taken almost verbatim from Introduction to Algorithms, third edition by T. H. Cormen, C. E. Leiserson, R. L. Rivest and Clifford Stein. In particular, the Bland's rule http://en.wikipedia.org/wiki/Bland%27s_r	3								
core_sort	CVAPI(ExceptionStatus) core_sort(cv::_InputArray *src, cv::_OutputArray *dst, int flags) { BEGIN_WRAP cv::sort(*src, *dst, flags); END_WRAP }	cv::sort()	group__core__array.html	Sorts each row or each column of a matrix.	src : input single-channel array. dst : output array of the same size and type as src. flags : operation flags, a combination of SortFlags	The function cv::sort sorts each matrix row or each matrix column in ascending or descending order. So you should pass two operation flags to get desired behaviour. If you want to sort matrix rows or columns lexicographically, you can use STL std::sort generic function with the proper comparison predicate.See alsosortIdx, randShuffleExamples: samples/cpp/stitching_detailed.cpp.	3								

core_sort1dx	CVAPI(ExceptionStatus) core_sort1dx(cv::_InputArray *src, cv::_OutputArray *dst, int flags) { BEGIN_WRAP cv::sort1dx(*src, *dst, flags); END_WRAP }	cv::sort1dx()	group__core__array.html	Sorts each row or each column of a matrix.	src : input single-channel array. dst : output integer array of the same size as src. flags : operation flags that could be a combination of cv::SortFlags	The function cv::sort1dx sorts each matrix row or each matrix column in the ascending or descending order. So you should pass two operation flags to get desired behaviour. Instead of reordering the elements themselves, it stores the indices of sorted elements in the output array. For example:Mat A = Mat::eye(3,3,CV_32F), B;sort1dx(A, B, SORT_EVERY_ROW + SORT_ASCENDING);// B will probably contain// (because of equal elements in A some permutations are possible):// [[1, 2, 0], [0, 2, 1], [0, 1, 2]]fragmentSee alsosort, sortIdx()	3								
core_solveCubic	CVAPI(ExceptionStatus) core_solveCubic(cv::_InputArray *coeffs, cv::_OutputArray *roots, int *returnValue) { BEGIN_WRAP *returnValue = cv::solveCubic(*coeffs, *roots); END_WRAP }	cv::solveCubic()	group__core__array.html	Finds the real roots of a cubic equation.	coeffs : equation coefficients, an array of 3 or 4 elements. roots : output array of real roots that has 1 or 3 elements.	The function solveCubic finds the real roots of a cubic equation:if coeffs is a 4-element vector: $$\texttt{coeffs} [0] x^3 + \texttt{coeffs} [1] x^2 + \texttt{coeffs} [2] x + \texttt{coeffs} [3] = 0$$ if coeffs is a 3-element vector: $$x^3 + \texttt{coeffs} [0] x^2 + \texttt{coeffs} [1] x + \texttt{coeffs} [2] = 0$$The roots are stored in the roots array.	2								
core_solvePoly	CVAPI(ExceptionStatus) core_solvePoly(cv::_InputArray *coeffs, cv::_OutputArray *roots, int maxIters, double *returnValue) { BEGIN_WRAP *returnValue = cv::solvePoly(*coeffs, *roots, maxIters); END_WRAP }	cv::solvePoly()	group__core__array.html	Finds the real or complex roots of a polynomial equation.	coeffs : array of polynomial coefficients. roots : output (complex) array of roots. maxIters : maximum number of iterations the algorithm does.	The function cv::solvePoly finds real and complex roots of a polynomial equation:$$\texttt{coeffs} [n] x^{n} + \texttt{coeffs} [n-1] x^{n-1} + ... + \texttt{coeffs} [1] x + \texttt{coeffs} [0] = 0$$	3								
core_eigen	CVAPI(ExceptionStatus) core_eigen(cv::_InputArray *src, cv::_OutputArray *eigenvalues, cv::_OutputArray *eigenvectors, int *returnValue) { BEGIN_WRAP *returnValue = cv::eigen(*src, *eigenvalues, *eigenvectors) ? 1 : 0; END_WRAP }	cv::eigen()	group__core__array.html	Calculates eigenvalues and eigenvectors of a symmetric matrix.	src : input matrix that must have CV_32FC1 or CV_64FC1 type, square size and be symmetrical (src^T^ == src). eigenvalues : output vector of eigenvalues of the same type as src; the eigenvalues are stored in the descending order. eigenvectors : output matrix of eigenvectors; it has the same size and type as src; the eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding eigenvalues.	The function cv::eigen calculates just eigenvalues, or eigenvalues and eigenvectors of the symmetric matrix src:src*eigenvectors.row(i).t() = eigenvalues.at<srcType>(i)*eigenvectors.row(i).t()fragmentNoteUse cv::eigenNonSymmetric for calculation of real eigenvalues and eigenvectors of non-symmetric matrix.See alsoeigenNonSymmetric, completeSymm , PCA	3								
core_eigenNonSymmetric	CVAPI(ExceptionStatus) core_eigenNonSymmetric(cv::_InputArray *src, cv::_OutputArray *eigenvalues, cv::_OutputArray *eigenvectors) { BEGIN_WRAP cv::eigenNonSymmetric(*src, *eigenvalues, *eigenvectors); END_WRAP }	cv::eigenNonSymmetric()	group__core__array.html	Calculates eigenvalues and eigenvectors of a non-symmetric matrix (real eigenvalues only).	src : input matrix (CV_32FC1 or CV_64FC1 type). eigenvalues : output vector of eigenvalues (type is the same type as src). eigenvectors : output matrix of eigenvectors (type is the same type as src). The eigenvectors are stored as subsequent matrix rows, in the same order as the corresponding eigenvalues.	NoteAssumes real eigenvalues.The function calculates eigenvalues and eigenvectors (optional) of the square matrix src:src*eigenvectors.row(i).t() = eigenvalues.at<srcType>(i)*eigenvectors.row(i).t()fragmentSee alsoeigen	3								
core_calcCovarMatrix_Mat	CVAPI(ExceptionStatus) core_calcCovarMatrix_Mat(cv::Mat **samples, int nsamples, cv::Mat *covar, cv::Mat *mean, int flags, int ctype) { BEGIN_WRAP std::vector<cv::Mat> samplesVec(nsamples); for (int i = 0; i < nsamples; i++) samplesVec[i] = *samples[i]; cv::calcCovarMatrix(&samplesVec[0], nsamples, *covar, *mean, flags, ctype); END_WRAP }	cv::calcCovarMatrix()	group__core__array.html	Calculates the covariance matrix of a set of vectors.	samples : samples stored as separate matrices nsamples : number of samples covar : output covariance matrix of the type ctype and square size. mean : input or output (depending on the flags) array as the average value of the input vectors. flags : operation flags as a combination of CovarFlags ctype : type of the matrix; it equals 'CV_64F' by default.	The function cv::calcCovarMatrix calculates the covariance matrix and, optionally, the mean vector of the set of input vectors.See alsePCA, mulTransposed, MahalanobisTodo:InputArrayOfArrays	6								
core_calcCovarMatrix_InputArray	CVAPI(ExceptionStatus) core_calcCovarMatrix_InputArray(cv::_InputArray *samples, cv::_OutputArray *covar, cv::_InputOutputArray *mean, int flags, int ctype) { BEGIN_WRAP cv::calcCovarMatrix(*samples, *covar, *mean, flags, ctype); END_WRAP }	cv::calcCovarMatrix()	group__core__array.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	samples : samples stored as rows/columns of a single matrix. covar : output covariance matrix of the type ctype and square size. mean : input or output (depending on the flags) array as the average value of the input vectors. flags : operation flags as a combination of CovarFlags ctype : type of the matrix; it equals 'CV_64F' by default.	Noteuse CDVAR_ROWS or CDVAR_COLS flag	5								
core_PCACompute	CVAPI(ExceptionStatus) core_PCACompute(cv::_InputArray *data, cv::_InputOutputArray *mean, cv::_OutputArray *eigenvectors, int maxComponents) { BEGIN_WRAP cv::PCACompute(*data, *mean, *eigenvectors, maxComponents); END_WRAP }	cv::PCACompute()	group__core__array.html	wrap PCA::operator()			4									
core_PCACompute2	CVAPI(ExceptionStatus) core_PCACompute2(cv::_InputArray *data, cv::_InputOutputArray *mean, cv::_OutputArray *eigenvectors, cv::_OutputArray *eigenvalues, int maxComponents) { BEGIN_WRAP cv::PCACompute(*data, *mean, *eigenvectors, *eigenvalues, maxComponents); END_WRAP }	cv::PCACompute()	group__core__array.html	wrap PCA::operator()			5									
core_PCAComputeVar	CVAPI(ExceptionStatus) core_PCAComputeVar(cv::_InputArray *data, cv::_InputOutputArray *mean, cv::_OutputArray *eigenvectors, double retainedVariance) { BEGIN_WRAP cv::PCACompute(*data, *mean, *eigenvectors, retainedVariance); END_WRAP }	cv::PCACompute()	group__core__array.html	wrap PCA::operator()			4									
core_PCAComputeVar2	CVAPI(ExceptionStatus) core_PCAComputeVar2(cv::_InputArray *data, cv::_InputOutputArray *mean, cv::_OutputArray *eigenvectors, cv::_OutputArray *eigenvalues, double retainedVariance) { BEGIN_WRAP cv::PCACompute(*data, *mean, *eigenvectors, *eigenvalues, retainedVariance); END_WRAP }	cv::PCACompute()	group__core__array.html	wrap PCA::operator()			5									
core_PCAProject	CVAPI(ExceptionStatus) core_PCAProject(cv::_InputArray *data, cv::_InputArray *mean, cv::_InputArray *eigenvectors, cv::_OutputArray *result) { BEGIN_WRAP cv::PCAProject(*data, *mean, *eigenvectors, *result); END_WRAP }	cv::PCAProject()	group__core__array.html	wrap PCA::project			4									

core_PCABackProject	CVAPI(ExceptionStatus) core_PCABackProject(cv::_InputArray *data, cv::_InputArray *mean, cv::_InputArray *eigenvectors, cv::_OutputArray *result) { BEGIN_WRAP cv::PCABackProject(*data, *mean, *eigenvectors, *result); END_WRAP }	cv::PCABackProject()	group__core__array.html	wrap PCA::backProject			4								
core_SVDecomp	CVAPI(ExceptionStatus) core_SVDecomp(cv::_InputArray *src, cv::_OutputArray *w, cv::_OutputArray *u, cv::_OutputArray *vt, int flags) { BEGIN_WRAP cv::SVDecomp(*src, *w, *u, *vt, flags); END_WRAP }	cv::SVDecomp()	group__core__array.html	wrap SVD::compute		Examples: samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	5								
core_SVBackSubst	CVAPI(ExceptionStatus) core_SVBackSubst(cv::_InputArray *w, cv::_InputArray *u, cv::_InputArray *vt, cv::_InputArray *rhs, cv::_OutputArray *dst) { BEGIN_WRAP cv::SVBackSubst(*w, *u, *vt, *rhs, *dst); END_WRAP }	cv::SVBackSubst()	group__core__array.html	wrap SVD::backSubst			5								
core_Mahalanobis	CVAPI(ExceptionStatus) core_Mahalanobis(cv::_InputArray *v1, cv::_InputArray *v2, cv::_InputArray *icovar, double *returnValue) { BEGIN_WRAP *returnValue = cv::Mahalanobis(*v1, *v2, *icovar); END_WRAP }	cv::Mahalanobis()	group__core__array.html	Calculates the Mahalanobis distance between two vectors.	v1 : first 1D input vector. v2 : second 1D input vector. icovar : inverse covariance matrix.	The function cv::Mahalanobis calculates and returns the weighted distance between two vectors:¥[d(¥textit{vec1} , ¥textit{vec2})= ¥sqrt{¥sum_{i,j} {¥textit{icovar(i,j)}¥cdot(¥textit{vec1}(I)- ¥textit{vec2})(¥textit{vec1}(J)- ¥textit{vec2})}} ¥]¥The covariance matrix may be calculated using the calcCovarMatrix function and then inverted using the invert function (preferably using the DECOMP_SVD method, as the most accurate).	3								
core_dft	CVAPI(ExceptionStatus) core_dft(cv::_InputArray *src, cv::_OutputArray *dst, int flags, int nonzeroRows) { BEGIN_WRAP cv::dft(*src, *dst, flags, nonzeroRows); END_WRAP }	cv::dft()	group__core__array.html	Performs a forward or inverse Discrete Fourier transform of a 1D or 2D floating-point array.	src : input array that could be real or complex. dst : output array whose size and type depends on the flags . flags : transformation flags, representing a combination of the DftFlags nonzeroRows : when the parameter is not zero, the function assumes that only the first nonzeroRows rows of the input array (DFT_INVERSE is not set) or only the first nonzeroRows of the output array (DFT_INVERSE is set) contain non-zeros, thus, the function can handle the rest of the rows more efficiently and save some time; this technique is very useful for calculating array cross-correlation or convolution using DFT.	The function cv::dft performs one of the following:Forward the Fourier transform of a 1D vector of N elements: ¥[Y = F^{N} ¥cdot X,¥] where ¥[F^{N}_{jk}=¥exp(-2¥pi i j k/N)¥] and ¥(i=¥sqrt{-1}¥) Inverse the Fourier transform of a 1D vector of N elements: ¥[¥begin{array}{l} X= ¥left (F^{N} ¥right)^{-1} ¥cdot Y = ¥left (F^{N} ¥right)^* ¥cdot y ¥¥ X = (1/N) ¥cdot X, ¥end{array}¥] where ¥[F^*=¥left(¥textrm{Re}(F^{N})-¥textrm{Im}(F^{N})¥right)^T¥] Forward the 2D Fourier transform of a M x N matrix: ¥[Y = F^{M}) ¥cdot X ¥cdot F^{N}¥] Inverse the 2D Fourier transform of a M x N matrix: ¥[¥begin{array}{l} X= ¥left (F^{M} ¥right)^* ¥cdot Y ¥cdot ¥left (F^{N} ¥right)^* ¥¥ X = ¥frac{1}{M ¥cdot N}	4								
core_idft	CVAPI(ExceptionStatus) core_idft(cv::_InputArray *src, cv::_OutputArray *dst, int flags, int nonzeroRows) { BEGIN_WRAP cv::idft(*src, *dst, flags, nonzeroRows); END_WRAP }	cv::idft()	group__core__array.html	Calculates the inverse Discrete Fourier Transform of a 1D or 2D array.	src : input floating-point real or complex array. dst : output array whose size and type depend on the flags. flags : operation flags (see dft and DftFlags). nonzeroRows : number of dst rows to process; the rest of the rows have undefined content (see the convolution sample in dft description.	idft(src, dst, flags) is equivalent to dft(src, dst, flags | DFT_INVERSE) .NoteNone of dft and idft scales the result by default. So, you should pass DFT_SCALE to one of dft or idft explicitly to make these transforms mutually inverse.See alsodft, dct, idct, mulSpectrums, getOptimalDFTSize	4								
core_dct	CVAPI(ExceptionStatus) core_dct(cv::_InputArray *src, cv::_OutputArray *dst, int flags) { BEGIN_WRAP cv::dct(*src, *dst, flags); END_WRAP }	cv::dct()	group__core__array.html	Performs a forward or inverse discrete Cosine transform of 1D or 2D array.	src : input floating-point array. dst : output array of the same size and type as src . flags : transformation flags as a combination of cv::DftFlags (DCT_*)	The function cv::dct performs a forward or inverse discrete Cosine transform (DCT) of a 1D or 2D floating-point array:Forward Cosine transform of a 1D vector of N elements: ¥[Y = C^{N} ¥cdot X¥] where ¥[C^{N}_{jk}= ¥sqrt{¥alpha_j/N} ¥cos ¥left (¥frac{¥pi(2k+1)j}{2N} ¥right)¥] and ¥(¥alpha_0=1¥), ¥(¥alpha_j=2¥) for j > 0. Inverse Cosine transform of a 1D vector of N elements: ¥[X = ¥left (C^{N} ¥right)^{-1} ¥cdot Y = ¥left (C^{N} ¥right)^T ¥cdot Y¥] (since ¥[C^{N}¥] is an orthogonal matrix, ¥[C^{N} ¥cdot ¥left(C^{N}¥right)^T = I¥]) Forward 2D Cosine transform of M x N matrix: ¥[Y = C^{N} ¥cdot X ¥cdot ¥left (C^{N} ¥right)^T¥]	3								
core_idct	CVAPI(ExceptionStatus) core_idct(cv::_InputArray *src, cv::_OutputArray *dst, int flags) { BEGIN_WRAP cv::idct(*src, *dst, flags); END_WRAP }	cv::idct()	group__core__array.html	Calculates the inverse Discrete Cosine Transform of a 1D or 2D array.	src : input floating-point single-channel array. dst : output array of the same size and type as src. flags : operation flags.	idct(src, dst, flags) is equivalent to dct(src, dst, flags | DCT_INVERSE).See alsodct, dft, idft, getOptimalDFTSize	3								
core_mulSpectrums	CVAPI(ExceptionStatus) core_mulSpectrums(cv::_InputArray *a, cv::_InputArray *b, cv::_OutputArray *c, int flags, int conjB) { BEGIN_WRAP cv::mulSpectrums(*a, *b, *c, flags, conjB != 0); END_WRAP }	cv::mulSpectrums()	group__core__array.html	Performs the per-element multiplication of two Fourier spectrums.	a : first input array. b : second input array of the same size and type as src1 . c : output array of the same size and type as src1 . flags : operation flags; currently, the only supported flag is cv::DFT_ROWS, which indicates that each row of src1 and src2 is an independent 1D Fourier spectrum. If you do not want to use this flag, then simply add a 0 as value. conjB : optional flag that conjugates the second input array before the multiplication (true) or not (false).	The function cv::mulSpectrums performs the per-element multiplication of the two CCS-packed or complex matrices that are results of a real or complex Fourier transform.The function, together with dft and idft , may be used to calculate convolution (pass conjB=false) or correlation (pass conjB=true) of two arrays rapidly. When the arrays are complex, they are simply multiplied (per element) with an optional conjugation of the second-array elements. When the arrays are real, they are assumed to be CCS-packed (see dft for details).	5								
core_getOptimalDFTSize	CVAPI(ExceptionStatus) core_getOptimalDFTSize(int vecsize, int *returnValue) { BEGIN_WRAP *returnValue = cv::getOptimalDFTSize(vecsize); END_WRAP }	cv::getOptimalDFTSize()	group__core__array.html	Returns the optimal DFT size for a given vector size.	vecsize : vector size.	DFT performance is not a monotonic function of a vector size. Therefore, when you calculate convolution of two arrays or perform the spectral analysis of an array, it usually makes sense to pad the input data with zeros to get a bit larger array that can be transformed much faster than the original one. Arrays whose size is a power-of-two (2, 4, 8, 16, 32, ...) are the fastest to process. Though, the arrays whose size is a product of 2's, 3's, and 5's (for example, 300 = 5*5*3*2*2) are also processed quite efficiently.The function cv::getOptimalDFTSize returns the minimum number N that is greater than or equal to vecsize so that the DFT of a vector of size N can be processed efficiently. In the current implementation N = 2 ^p^ * 3 ^q^ * 5 ^r^ for some integer p, q, r.The function returns a negative number if vecsize is too large (very close to INT_MAX).While the function cannot be used directly to estimate the optimal vector size for DCT transform (since the current DCT implementation supports only even-size vectors), it can be easily processed as	1								
core_theRNG_get	CVAPI(ExceptionStatus) core_theRNG_get(uint64 *returnValue) { BEGIN_WRAP cv::RNG &rng = cv::theRNG(); *returnValue = rng.state; END_WRAP }	cv::RNG::state	classcv_1_1RNG.html							•						
core_theRNG_set	CVAPI(ExceptionStatus) core_theRNG_set(uint64 value) { BEGIN_WRAP cv::theRNG().state = value; END_WRAP }	cv::RNG::state	classcv_1_1RNG.html								•					
core_randu_InputArray	CVAPI(ExceptionStatus) core_randu_InputArray(cv::_InputOutputArray *dst, cv::_InputArray *low, cv::_InputArray *high) { BEGIN_WRAP cv::randu(*dst, *low, *high); END_WRAP }	cv::randu()	group__core__array.html	Generates a single uniformly-distributed random number or an array of random numbers.	dst : output array of random numbers; the array must be pre-allocated. low : inclusive lower boundary of the generated random numbers. high : exclusive upper boundary of the generated random numbers.	Non-template variant of the function fills the matrix dst with uniformly-distributed random numbers from the specified range:¥[¥textit{low} _c ¥leq ¥textit{dst}(I)_c < ¥textit{high} _c¥]See alsoRNG, randn, theRNGExamples: samples/cpp/cout_mat.cpp, and samples/cpp/falsecolor.cpp.	3								

core_randu_Scalar	CVAPI(ExceptionStatus) core_randu_Scalar(cv::_InputOutputArray *dst, MyCvScalar low, MyCvScalar high) { BEGIN_WRAP cv::randu(*dst, cpp(low), cpp(high)); END_WRAP }	cv::randu()	group__core__array.html	Generates a single uniformly-distributed random number or an array of random numbers.	dst : output array of random numbers; the array must be pre-allocated. low : inclusive lower boundary of the generated random numbers. high : exclusive upper boundary of the generated random numbers.	Non-template variant of the function fills the matrix dst with uniformly-distributed random numbers from the specified range:¥(¥texttt{low} _c ¥leq ¥texttt{dst}(I)_c < ¥texttt{high} _c¥)See alsoRNG, randn, theRNGExamples: samples/cpp/cout_mat.cpp, and samples/cpp/falsecolor.cpp.	3								
core_randn_InputArray	CVAPI(ExceptionStatus) core_randn_InputArray(cv::_InputOutputArray *dst, cv::_InputArray *mean, cv::_InputArray *stddev) { BEGIN_WRAP cv::randn(*dst, *mean, *stddev); END_WRAP }	cv::randn()	group__core__array.html	Fills the array with normally distributed random numbers.	dst : output array of random numbers; the array must be pre-allocated and have 1 to 4 channels. mean : mean value (expectation) of the generated random numbers. stddev : standard deviation of the generated random numbers; it can be either a vector (in which case a diagonal standard deviation matrix is assumed) or a square matrix.	The function cv::randn fills the matrix dst with normally distributed random numbers with the specified mean vector and the standard deviation matrix. The generated random numbers are clipped to fit the value range of the output array data type.See alsoRNG, randuExamples: samples/cpp/kalman.cpp.	3								
core_randn_Scalar	CVAPI(ExceptionStatus) core_randn_Scalar(cv::_InputOutputArray *dst, MyCvScalar mean, MyCvScalar stddev) { BEGIN_WRAP cv::randn(*dst, cpp(mean), cpp(stddev)); END_WRAP }	cv::randn()	group__core__array.html	Fills the array with normally distributed random numbers.	dst : output array of random numbers; the array must be pre-allocated and have 1 to 4 channels. mean : mean value (expectation) of the generated random numbers. stddev : standard deviation of the generated random numbers; it can be either a vector (in which case a diagonal standard deviation matrix is assumed) or a square matrix.	The function cv::randn fills the matrix dst with normally distributed random numbers with the specified mean vector and the standard deviation matrix. The generated random numbers are clipped to fit the value range of the output array data type.See alsoRNG, randuExamples: samples/cpp/kalman.cpp.	3								
core_randShuffle	CVAPI(ExceptionStatus) core_randShuffle(cv::_InputOutputArray *dst, double iterFactor, uint64 *rng) { BEGIN_WRAP cv::RNG rng0; cv::randShuffle(*dst, iterFactor, &rng0); *rng = rng0.state; END_WRAP }	cv::randShuffle()	group__core__array.html	Shuffles the array elements randomly.	dst : input/output numerical 1D array. iterFactor : scale factor that determines the number of random swap operations (see the details below). rng : optional random number generator used for shuffling; if it is zero, theRNG () is used instead.	The function cv::randShuffle shuffles the specified 1D array by randomly choosing pairs of elements and swapping them. The number of such swap operations will be dst.rows*dst.cols*iterFactor .See alsoRNG, sortExamples: modules/shape/samples/shape_example.cpp, and samples/cpp/kmeans.cpp.	3								
core_kmeans	CVAPI(ExceptionStatus) core_kmeans(cv::_InputArray *data, int k, cv::_InputOutputArray *bestLabels, MyCvTermCriteria criteria, int attempts, int flags, cv::_OutputArray *centers, double* returnValue) { BEGIN_WRAP *returnValue = cv::kmeans(*data, k, *bestLabels, cpp(criteria), attempts, flags, &ntity(centers)); END_WRAP }	cv::kmeans()	group__core__cluster.html	Finds centers of clusters and groups input samples around the clusters.	data : Data for clustering. An array of N-Dimensional points with float coordinates is needed. Examples of this array can be: Mat points(count, 2, CV_32F); Mat points(count, 1, CV_32FC2); Mat points(1, count, CV_32FC2); std::vector<cv::Point2f> points(sampleCount); K : Number of clusters to split the set by. bestLabels : Input/output integer array that stores the cluster indices for every sample. criteria : The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy. The accuracy is specified as criteria.epsilon. As soon as each of the cluster centers moves by less than criteria.epsilon on some iteration, the algorithm stops. attempts : Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness (see the last function parameter). flags : Flag that can take values of cv::KmeansFlags centers : Output matrix of the cluster centers, one row per each cluster center.	The function kmeans implements a k-means algorithm that finds the centers of cluster_count clusters and groups the input samples around the clusters. As an output, ¥(¥texttt{bestLabels}_i¥) contains a 0-based cluster index for the sample stored in the ¥(i¥)-¥(th¥) row of the samples matrix.Note (Python) An example on K-means clustering can be found at opencv_source_code/samples/python/kmeans.pyExamples: samples/cpp/kmeans.cpp.	7								
core_cubeRoot	CVAPI(ExceptionStatus) core_cubeRoot(float val, float* returnValue) { BEGIN_WRAP *returnValue = cv::cubeRoot(val); END_WRAP }	cv::cubeRoot()	group__core__utils.html	Computes the cube root of an argument.	val : A function argument.	The function cubeRoot computes ¥(¥sqrt[3]{¥texttt{val}}¥). Negative arguments are handled correctly. NaN and Inf are not handled. The accuracy approaches the maximum possible accuracy for single-precision data.	1								
core_fastAtan2	CVAPI(ExceptionStatus) core_fastAtan2(float y, float x, float* returnValue) { BEGIN_WRAP *returnValue = cv::fastAtan2(y, x); END_WRAP }	cv::fastAtan2()	group__core__utils.html	Calculates the angle of a 2D vector in degrees.	x : x-coordinate of the vector. y : y-coordinate of the vector.	The function fastAtan2 calculates the full-range angle of an input 2D vector. The angle is measured in degrees and varies from 0 to 360 degrees. The accuracy is about 0.3 degrees.	2								
core_Algorithm_write	CVAPI(ExceptionStatus) core_Algorithm_write(cv::Algorithm *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->write(*fs); END_WRAP }	cv::Algorithm::write()	classcv_1_1Algorithm.html	Stores algorithm parameters in a file storage.		Reimplemented in cv::FlannBasedMatcher, cv::DescriptorMatcher, cv::face::FaceRecognizer, cv::optflow::GPCForest< T >, cv::bioinspired::Retina, cv::line_descriptor::BinaryDescriptor, cv::Feature2D, cv::optflow::GPCTree, cv::bioinspired::TransientAreasSegmentationModule, cv::legacy::Tracker, cv::saliency::StaticSaliencySpectralResidual, and cv::face::BasicFaceRecognizer.	1								
core_Algorithm_read	CVAPI(ExceptionStatus) core_Algorithm_read(cv::Algorithm *obj, cv::FileNode *fn) { BEGIN_WRAP obj->read(*fn); END_WRAP }	cv::Algorithm::read()	classcv_1_1Algorithm.html	Reads algorithm parameters from a file storage.		Reimplemented in cv::FlannBasedMatcher, cv::DescriptorMatcher, cv::face::FaceRecognizer, cv::optflow::GPCForest< T >, cv::line_descriptor::BinaryDescriptor, cv::Feature2D, cv::optflow::GPCTree, cv::legacy::Tracker, cv::saliency::StaticSaliencySpectralResidual, and cv::face::BasicFaceRecognizer.	1								
core_Algorithm_empty	CVAPI(ExceptionStatus) core_Algorithm_empty(cv::Algorithm *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::Algorithm::empty()	classcv_1_1Algorithm.html	Returns true if the Algorithm is empty (e.g. in the very beginning or after unsuccessful read.		Reimplemented in cv::DescriptorMatcher, cv::face::FaceRecognizer, cv::ml::StatModel, cv::Feature2D, cv::BaseCascadeClassifier, cv::cuda::DescriptorMatcher, cv::quality::QualityPSNR, cv::quality::QualityBase, cv::face::BasicFaceRecognizer, cv::quality::QualityGMSD, cv::quality::QualitySSIM, and	1								
core_Algorithm_save	CVAPI(ExceptionStatus) core_Algorithm_save(cv::Algorithm *obj, const char *filename) { BEGIN_WRAP obj->save(filename); END_WRAP }	cv::Algorithm::save()	classcv_1_1Algorithm.html	Saves the algorithm to a file. In order to make this method work, the derived class must implement Algorithm::write(FileStorage& fs).			1								
core_Algorithm_getDefaultName	CVAPI(ExceptionStatus) core_Algorithm_getDefaultName(cv::Algorithm *obj, std::string *buf) { BEGIN_WRAP buf->assign(obj->getDefaultName()); END_WRAP }	cv::Algorithm::assign()															
core_PCA_new1	CVAPI(ExceptionStatus) core_PCA_new1(cv::PCA **returnValue) { BEGIN_WRAP *returnValue = new cv::PCA; END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two										•			
core_PCA_new2	CVAPI(ExceptionStatus) core_PCA_new2(cv::_InputArray *data, cv::_InputArray *mean, int flags, int maxComponents, cv::PCA **returnValue) { BEGIN_WRAP *returnValue = new cv::PCA(*data, *mean, flags, maxComponents); END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two matrices. The first function stores a set of vectors										•			

core_PCA_new3	CVAPI(ExceptionStatus) core_PCA_new3(cv::_InputArray *data, cv::_InputArray *mean, int flags, double retainedVariance, cv::PCA **returnValue) { BEGIN_WRAP *returnValue = new cv::PCA(*data, *mean, flags, retainedVariance); END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two									•		
core_PCA_delete	CVAPI(ExceptionStatus) core_PCA_delete(cv::PCA *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two										•	
core_PCA_operatorThis	CVAPI(ExceptionStatus) core_PCA_operatorThis(cv::PCA *obj, cv::_InputArray *data, cv::_InputArray *mean, int flags, int maxComponents) { BEGIN_WRAP (*obj)(*data, *mean, flags, maxComponents); END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two								•			
core_PCA_computeVar	CVAPI(ExceptionStatus) core_PCA_computeVar(cv::PCA *obj, cv::_InputArray *data, cv::_InputArray *mean, int flags, double retainedVariance) { BEGIN_WRAP (*obj)(*data, *mean, flags, retainedVariance); END_WRAP }	cv::PCA	classcv_1_1PCA.html	Principal Component Analysis. The class is used to calculate a special basis for a set of vectors. The basis will consist of eigenvectors of the covariance matrix calculated from the input set of vectors. The class PCA can also transform vectors to/from the new coordinate space defined by the basis. Usually, in this new coordinate system, each vector from the original set (and any linear combination of such vectors) can be quite accurately approximated by taking its first few components, corresponding to the eigenvectors of the largest eigenvalues of the covariance matrix. Geometrically it means that you calculate a projection of the vector to a subspace formed by a few eigenvectors corresponding to the dominant eigenvalues of the covariance matrix. And usually such a projection is very close to the original vector. So, you can represent the original vector from a high-dimensional space with a much shorter vector consisting of the projected vector's coordinates in the subspace. Such a transformation is also known as Karhunen-Loeve Transform, or KLT. See http://en.wikipedia.org/wiki/Principal_component_analysis The sample below is the function that takes two								•			
core_PCA_project1	CVAPI(ExceptionStatus) core_PCA_project1(cv::PCA *obj, cv::_InputArray *vec, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = obj->project(*vec); *returnValue = new cv::Mat(ret); END_WRAP }	cv::PCA::project()	classcv_1_1PCA.html	Projects vector(s) to the principal component subspace.	vec : input vector(s); must have the same dimensionality and the same layout as the input data used at PCA phase, that is, if DATA_AS_ROW are specified, then vec.cols==data.cols (vector dimensionality) and vec.rows is the number of vectors to project, and the same is true for the PCA::DATA_AS_COL case.	The methods project one or more vectors to the principal component subspace, where each vector projection is represented by coefficients in the principal component basis. The first form of the method returns the matrix that the second form writes to the result. So the first form can be used as a part of expression while the second form can be more efficient in a processing loop.Examples: samples/cpp/pca.cpp.	1								
core_PCA_project2	CVAPI(ExceptionStatus) core_PCA_project2(cv::PCA *obj, cv::_InputArray *vec, cv::_OutputArray *result) { BEGIN_WRAP obj->project(*vec, *result); END_WRAP }	cv::PCA::project()	classcv_1_1PCA.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	vec : input vector(s); must have the same dimensionality and the same layout as the input data used at PCA phase, that is, if DATA_AS_ROW are specified, then vec.cols==data.cols (vector dimensionality) and vec.rows is the number of vectors to project, and the same is true for the PCA::DATA_AS_COL case. result : output vectors; in case of PCA::DATA_AS_COL, the output matrix has as many columns as the number of input vectors, this means that result.cols==vec.cols and the number of rows match the number of principal components (for example, maxComponents parameter passed to the constructor).		2								
core_PCA_backProject1	CVAPI(ExceptionStatus) core_PCA_backProject1(cv::PCA *obj, cv::_InputArray *vec, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = obj->backProject(*vec); *returnValue = new cv::Mat(ret); END_WRAP }	cv::PCA::backProject()	classcv_1_1PCA.html	Reconstructs vectors from their PC projections.	vec : coordinates of the vectors in the principal component subspace, the layout and size are the same as of PCA::project output vectors.	The methods are inverse operations to PCA::project. They take PC coordinates of projected vectors and reconstruct the original vectors. Unless all the principal components have been retained, the reconstructed vectors are different from the originals. But typically, the difference is small if the number of components is large enough (but still much smaller than the original vector dimensionality). As a result, PCA is used.Examples: samples/cpp/pca.cpp.	1								
core_PCA_backProject2	CVAPI(ExceptionStatus) core_PCA_backProject2(cv::PCA *obj, cv::_InputArray *vec, cv::_OutputArray *result) { BEGIN_WRAP obj->backProject(*vec, *result); END_WRAP }	cv::PCA::backProject()	classcv_1_1PCA.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	vec : coordinates of the vectors in the principal component subspace, the layout and size are the same as of PCA::project output vectors. result : reconstructed vectors; the layout and size are the same as of PCA::project input vectors.		2								
core_PCA_eigenvectors	CVAPI(ExceptionStatus) core_PCA_eigenvectors(cv::PCA *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->eigenvectors); END_WRAP }	cv::PCA::eigenvectors	classcv_1_1PCA.html	eigenvectors of the covariation matrix						•					
core_PCA_eigenvalues	CVAPI(ExceptionStatus) core_PCA_eigenvalues(cv::PCA *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->eigenvalues); END_WRAP }	cv::PCA::eigenvalues	classcv_1_1PCA.html	eigenvalues of the covariation matrix						•					
core_PCA_mean	CVAPI(ExceptionStatus) core_PCA_mean(cv::PCA *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->mean); END_WRAP }	cv::PCA::mean	classcv_1_1PCA.html	mean value subtracted before the projection and added after the back projection						•					
core_PCA_write	CVAPI(ExceptionStatus) core_PCA_write(cv::PCA *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->write(*fs); END_WRAP }	cv::PCA::write()	classcv_1_1PCA.html	write PCA objects		Writes eigenvalues eigenvectors and mean to specified FileStorage	1								
core_PCA_read	CVAPI(ExceptionStatus) core_PCA_read(cv::PCA *obj, cv::FileNode *fn) { BEGIN_WRAP obj->read(*fn); END_WRAP }	cv::PCA::read()	classcv_1_1PCA.html	load PCA objects		Loads eigenvalues eigenvectors and mean from specified FileNode	1								

core_RNG_fill	CVAPI(ExceptionStatus) core_RNG_fill(uint64 *state, cv::_InputOutputArray *mat, int distType, cv::_InputArray *a, cv::_InputArray *b, int saturateRange) { BEGIN_WRAP cv::RNG rng(*state); rng.fill(*mat, distType, *a, *b, saturateRange != 0); *state = rng.state; END_WRAP }	cv::RNG::fill()	classcv_1_1RNG.html	Fills arrays with random numbers.	mat : 2D or N-dimensional matrix; currently matrices with more than 4 channels are not supported by the methods, use Mat::reshape as a possible workaround. distType : distribution type, RNG::UNIFORM or RNG::NORMAL. a : first distribution parameter; in case of the uniform distribution, this is an inclusive lower boundary, in case of the normal distribution, this is a mean value. b : second distribution parameter; in case of the uniform distribution, this is a non-inclusive upper boundary, in case of the normal distribution, this is a standard deviation (diagonal of the standard deviation matrix or the full standard deviation matrix). saturateRange : pre-saturation flag; for uniform distribution only; if true, the method will first convert a and b to the acceptable value range (according to the mat datatype) and then will generate uniformly distributed random numbers within the range [saturate(a), saturate(b)), if saturateRange=false, the method will generate uniformly distributed random numbers in the original range [a, b) and then will saturate them, it means, for example, that theRNG().fill(mat_8u, RNG::UNIFORM, -DBL_MAX, DBL_MAX) will likely produce array mostly filled with 0's and 255's, since the range (0, 255) is significantly smaller than [-DBL_MAX, DBL_MAX).	Each of the methods fills the matrix with the random values from the specified distribution. As the new numbers are generated, the RNG state is updated accordingly. In case of multiple-channel images, every channel is filled independently, which means that RNG cannot generate samples from the multi-dimensional Gaussian distribution with non-diagonal covariance matrix directly. To do that, the method generates samples from multi-dimensional standard Gaussian distribution with zero mean and identity covariation matrix, and then transforms them using transform to get samples from the specified Gaussian distribution.Examples: samples/cpp/kmeans.cpp.	5								
core_RNG_gaussian	CVAPI(ExceptionStatus) core_RNG_gaussian(uint64 *state, double sigma, double *returnValue) { BEGIN_WRAP cv::RNG rng(*state); *returnValue = rng.gaussian(sigma); *state = rng.state; END_WRAP }	cv::RNG::gaussian()	classcv_1_1RNG.html	Returns the next random number sampled from the Gaussian distribution.	sigma : standard deviation of the distribution.	The method transforms the state using the MWC algorithm and returns the next random number from the Gaussian distribution N(0,sigma) . That is, the mean value of the returned random numbers is zero and the standard deviation is the specified sigma .	1								
core_SVD_new1	CVAPI(ExceptionStatus) core_SVD_new1(cv::SVD **returnValue) { BEGIN_WRAP *returnValue = new cv::SVD; END_WRAP }	cv::SVD	classcv_1_1SVD.html	Singular Value Decomposition. Class for computing Singular Value Decomposition of a floating-point matrix. The Singular Value Decomposition is used to solve least-square problems, under-determined linear systems, invert matrices, compute condition numbers, and so on. If you want to compute a condition number of a matrix or an absolute value of its determinant, you do not need u and vt. You can pass flags=SVD::NO_UV|... . Another flag SVD::FULL_UV indicates that full-size u and vt must be computed, which is not necessary most of the time. See alsoinvert, solve, eigen, determinant								●			
core_SVD_new2	CVAPI(ExceptionStatus) core_SVD_new2(cv::_InputArray *src, int flags, cv::SVD **returnValue) { BEGIN_WRAP *returnValue = new cv::SVD(*src, flags); END_WRAP }	cv::SVD	classcv_1_1SVD.html	Singular Value Decomposition. Class for computing Singular Value Decomposition of a floating-point matrix. The Singular Value Decomposition is used to solve least-square problems, under-determined linear systems, invert matrices, compute condition numbers, and so on. If you want to compute a condition number of a matrix or an absolute value of its determinant, you do not need u and vt. You can pass flags=SVD::NO_UV|... . Another flag SVD::FULL_UV indicates that full-size u and vt must be computed, which is not necessary most of the time. See alsoinvert, solve, eigen, determinant								●			
core_SVD_delete	CVAPI(ExceptionStatus) core_SVD_delete(cv::SVD *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::SVD	classcv_1_1SVD.html	Singular Value Decomposition. Class for computing Singular Value Decomposition of a floating-point matrix. The Singular Value Decomposition is used to solve least-square problems, under-determined linear systems, invert matrices, compute condition numbers, and so on. If you want to compute a condition number of a matrix or an absolute value of its determinant, you do not need u and vt. You can pass flags=SVD::NO_UV|... . Another flag SVD::FULL_UV indicates that full-size u and vt must be computed, which is not necessary most of the time. See alsoinvert, solve, eigen, determinant									●		
core_SVD_operatorThis	CVAPI(ExceptionStatus) core_SVD_operatorThis(cv::SVD *obj, cv::_InputArray *src, int flags) { BEGIN_WRAP (*obj)(*src, flags); END_WRAP }	cv::SVD	classcv_1_1SVD.html	Singular Value Decomposition. Class for computing Singular Value Decomposition of a floating-point matrix. The Singular Value Decomposition is used to solve least-square problems, under-determined linear systems, invert matrices, compute condition numbers, and so on. If you want to compute a condition number of a matrix or an absolute value of its determinant, you do not need u and vt. You can pass flags=SVD::NO_UV|... . Another flag SVD::FULL_UV indicates that full-size u and vt must be computed, which is not necessary most of the time. See alsoinvert, solve, eigen, determinant					●						
core_SVD_backSubst	CVAPI(ExceptionStatus) core_SVD_backSubst(cv::SVD *obj, cv::_InputArray *rhs, cv::_OutputArray *dst) { BEGIN_WRAP obj->backSubst(*rhs, *dst); END_WRAP }	cv::SVD::backSubst()	classcv_1_1SVD.html	performs a singular value back substitution.	rhs : right-hand side of a linear system (u*w*v')*dst = rhs to be solved, where A has been previously decomposed. dst : found solution of the system.	The method calculates a back substitution for the specified right-hand side:¥[¥texttt{x} = ¥texttt{vt} ^T ¥cdot diag(¥texttt{w})^{-1} ¥cdot ¥texttt{u} ^T ¥cdot ¥texttt{rhs} ¥sim ¥texttt{A} ^{-1} ¥cdot ¥texttt{rhs}¥]Using this technique you can either get a very accurate solution of the convenient linear system, or the best (in the least-squares terms) pseudo-solution of an overdetermined linear system.NoteExplicit SVD with the further back substitution only makes sense if you need to solve many linear systems with the same left-hand side (for example, src). If all you need is to solve a single system (possibly with multiple rhs immediately available), simply call solve add pass DECOMP_SVD there. It does absolutely the same thing.	2								
core_SVD_static_compute1	CVAPI(ExceptionStatus) core_SVD_static_compute1(cv::_InputArray *src, cv::_OutputArray *w, cv::_OutputArray *u, cv::_OutputArray *vt, int flags) { BEGIN_WRAP cv::SVD::compute(*src, *w, *u, *vt, flags); END_WRAP }	cv::SVD::compute()	classcv_1_1SVD.html	decomposes matrix and stores the results to user-provided matrices	src : decomposed matrix. The depth has to be CV_32F or CV_64F. w : calculated singular values u : calculated left singular vectors vt : transposed matrix of right singular vectors flags : operation flags - see SVD::Flags.	The methods/functions perform SVD of matrix. Unlike SVD::SVD constructor and SVD::operator(), they store the results to the user-provided matrices:Mat A, w, u, vt;SVD::compute(A, w, u, vt);fragment	5								
core_SVD_static_compute2	CVAPI(ExceptionStatus) core_SVD_static_compute2(cv::_InputArray *src, cv::_OutputArray *w, int flags) { BEGIN_WRAP cv::SVD::compute(*src, *w, flags); END_WRAP }	cv::SVD::compute()	classcv_1_1SVD.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. computes singular values of a matrix	src : decomposed matrix. The depth has to be CV_32F or CV_64F. w : calculated singular values flags : operation flags - see SVD::Flags.		3								
core_SVD_static_backSubst	CVAPI(ExceptionStatus) core_SVD_static_backSubst(cv::_InputArray *w, cv::_InputArray *u, cv::_InputArray *vt, cv::_InputArray *rhs, cv::_OutputArray *dst) { BEGIN_WRAP cv::SVD::backSubst(*w, *u, *vt, *rhs, *dst); END_WRAP }	cv::SVD::backSubst()	classcv_1_1SVD.html	performs back substitution			5								
core_SVD_static_solveZ	CVAPI(ExceptionStatus) core_SVD_static_solveZ(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::SVD::solveZ(*src, *dst); END_WRAP }	cv::SVD::solveZ()	classcv_1_1SVD.html	solves an under-determined singular linear system	src : left-hand-side matrix. dst : found solution.	The method finds a unit-length solution x of a singular linear system A*x = 0. Depending on the rank of A, there can be no solutions, a single solution or an infinite number of solutions. In general, the algorithm solves the following problem:¥[dst = ¥arg ¥min _{x: ¥| x ¥| =1} ¥| src ¥cdot x ¥|¥]	2								
core_SVD_u	CVAPI(ExceptionStatus) core_SVD_u(cv::SVD *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->u); END_WRAP }	cv::SVD::u	classcv_1_1SVD.html							●					
core_SVD_w	CVAPI(ExceptionStatus) core_SVD_w(cv::SVD *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->w); END_WRAP }	cv::SVD::w	classcv_1_1SVD.html							●					
core_SVD_vt	CVAPI(ExceptionStatus) core_SVD_vt(cv::SVD *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->vt); END_WRAP }	cv::SVD::vt	classcv_1_1SVD.html							●					

core_LDA_new1	CVAPI(ExceptionStatus) core_LDA_new1(int num_components, cv::LDA **returnValue) { BEGIN_WRAP *returnValue = new cv::LDA(num_components); END_WRAP }	cv::LDA	classcv_1_1LDA.html	Linear Discriminant Analysis. Todo:document this class								•				
core_LDA_new2	CVAPI(ExceptionStatus) core_LDA_new2(cv::_InputAr ray *src, cv::_InputArray *labels, int num_components, cv::LDA **returnValue) { BEGIN_WRAP *returnValue = new cv::LDA(*src, *labels, num_components); END_WRAP }	cv::LDA	classcv_1_1LDA.html	Linear Discriminant Analysis. Todo:document this class								•				
core_LDA_delete	CVAPI(ExceptionStatus) core_LDA_delete(cv::LDA *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::LDA	classcv_1_1LDA.html	Linear Discriminant Analysis. Todo:document this class									•			
core_LDA_save_String	CVAPI(ExceptionStatus) core_LDA_save_String(cv::LD A *obj, const char *filename) { BEGIN_WRAP obj->save(filename); END_WRAP }	cv::LDA::save()	classcv_1_1LDA.html	Serializes this object to a given filename.			1									
core_LDA_load_String	CVAPI(ExceptionStatus) core_LDA_load_String(cv::LD A *obj, const char *filename) { BEGIN_WRAP obj->load(filename); END_WRAP }	cv::LDA::load()	classcv_1_1LDA.html	Deserializes this object from a given filename.			1									
core_LDA_save_FileStorage	CVAPI(ExceptionStatus) core_LDA_save_FileStorage(c v::LDA *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->save(*fs); END_WRAP }	cv::LDA::save()	classcv_1_1LDA.html	Serializes this object to a given filename.			1									
core_LDA_load_FileStorage	CVAPI(ExceptionStatus) core_LDA_load_FileStorage(cv ::LDA *obj, cv::FileStorage *node) { BEGIN_WRAP obj->load(*node); END_WRAP }	cv::LDA::load()	classcv_1_1LDA.html	Deserializes this object from a given filename.			1									
core_LDA_compute	CVAPI(ExceptionStatus) core_LDA_compute(cv::LDA *obj, cv::_InputArray *src, cv::_InputArray *labels) { BEGIN_WRAP obj->compute(*src, *labels); END_WRAP }	cv::LDA::compute()	classcv_1_1LDA.html	Compute the discriminants for data in src (row aligned) and labels.			2									
core_LDA_project	CVAPI(ExceptionStatus) core_LDA_project(cv::LDA *obj, cv::_InputArray *src, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = obj- >project(*src); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::project()	classcv_1_1LDA.html	Projects samples into the LDA subspace. src may be one or more row aligned samples.			1									
core_LDA_reconstruct	CVAPI(ExceptionStatus) core_LDA_reconstruct(cv::LD A *obj, cv::_InputArray *src, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = obj- >reconstruct(*src); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::reconstruct()	classcv_1_1LDA.html	Reconstructs projections from the LDA subspace. src may be one or more row aligned projections.			1									
core_LDA_eigenvectors	CVAPI(ExceptionStatus) core_LDA_eigenvectors(cv::L DA *obj, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = obj- >eigenvectors(); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::eigenvectors()	classcv_1_1LDA.html	Returns the eigenvectors of this LDA.			1									
core_LDA_eigenvalues	CVAPI(ExceptionStatus) core_LDA_eigenvalues(cv::LD A *obj, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = obj- >eigenvalues(); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::eigenvalues()	classcv_1_1LDA.html	Returns the eigenvalues of this LDA.			1									
core_LDA_subspaceProject	CVAPI(ExceptionStatus) core_LDA_subspaceProject(cv ::_InputArray *W, cv::_InputArray *mean, cv::_InputArray *src, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::LDA::subspaceProject(*W , *mean, *src); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::subspaceProject()	classcv_1_1LDA.html				3									
core_LDA_subspaceReconstruct	CVAPI(ExceptionStatus) core_LDA_subspaceReconstru ct(cv::_InputArray *W, cv::_InputArray *mean, cv::_InputArray *src, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::LDA::subspaceReconstruc t(*W, *mean, *src); *returnValue = new cv::Mat(mat); END_WRAP }	cv::LDA::subspaceReconstruct()	classcv_1_1LDA.html				3									
core_FileNode_new1	CVAPI(ExceptionStatus) core_FileNode_new1(cv::FileN ode **returnValue) { BEGIN_WRAP *returnValue = new cv::FileNode(); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.								•			
core_FileNode_delete	CVAPI(ExceptionStatus) core_FileNode_delete(cv::File Node *node) { BEGIN_WRAP delete node; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.									•		

core_FileNode_operatorThis_byString	CVAPI(ExceptionStatus) core_FileNode_operatorThis_byString(cv::FileNode *obj, const char *nodeName, cv::FileNode **returnValue) { BEGIN_WRAP const auto ret = (*obj)[nodeName]; *returnValue = new cv::FileNode(ret); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_operatorThis_byInt	CVAPI(ExceptionStatus) core_FileNode_operatorThis_byInt(cv::FileNode *obj, int i, cv::FileNode **returnValue) { BEGIN_WRAP const auto ret = (*obj)[i]; *returnValue = new cv::FileNode(ret); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_type	CVAPI(ExceptionStatus) core_FileNode_type(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->type(); END_WRAP }	cv::FileNode::type()	classcv_1_1FileNode.html	Returns type of the node.		Examples: samples/cpp/filestorage.cpp.	1									
core_FileNode_empty	CVAPI(ExceptionStatus) core_FileNode_empty(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::FileNode::empty()	classcv_1_1FileNode.html	returns true if the node is empty		Examples: samples/cpp/filestorage.cpp.	1									
core_FileNode_isNone	CVAPI(ExceptionStatus) core_FileNode_isNone(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isNone() ? 1 : 0; END_WRAP }	cv::FileNode::isNone()	classcv_1_1FileNode.html	returns true if the node is a "none" object			1									
core_FileNode_isSeq	CVAPI(ExceptionStatus) core_FileNode_isSeq(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isSeq() ? 1 : 0; END_WRAP }	cv::FileNode::isSeq()	classcv_1_1FileNode.html	returns true if the node is a sequence			1									
core_FileNode_isMap	CVAPI(ExceptionStatus) core_FileNode_isMap(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isMap() ? 1 : 0; END_WRAP }	cv::FileNode::isMap()	classcv_1_1FileNode.html	returns true if the node is a mapping			1									
core_FileNode_isInt	CVAPI(ExceptionStatus) core_FileNode_isInt(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isInt() ? 1 : 0; END_WRAP }	cv::FileNode::isInt()	classcv_1_1FileNode.html	returns true if the node is an integer			1									
core_FileNode_isReal	CVAPI(ExceptionStatus) core_FileNode_isReal(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isReal() ? 1 : 0; END_WRAP }	cv::FileNode::isReal()	classcv_1_1FileNode.html	returns true if the node is a floating-point number			1									
core_FileNode_isString	CVAPI(ExceptionStatus) core_FileNode_isString(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isString() ? 1 : 0; END_WRAP }	cv::FileNode::isString()	classcv_1_1FileNode.html	returns true if the node is a text string			1									
core_FileNode_isNamed	CVAPI(ExceptionStatus) core_FileNode_isNamed(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isNamed() ? 1 : 0; END_WRAP }	cv::FileNode::isNamed()	classcv_1_1FileNode.html	returns true if the node has a name			1									
core_FileNode_name	CVAPI(ExceptionStatus) core_FileNode_name(cv::FileNode *obj, std::string *buf) { BEGIN_WRAP buf->assign(obj->name()); END_WRAP }	cv::FileNode::name()	classcv_1_1FileNode.html	returns the node name or an empty string if the node is nameless			1									
core_FileNode_size	CVAPI(ExceptionStatus) core_FileNode_size(cv::FileNode *obj, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->size(); END_WRAP }	cv::FileNode::size()	classcv_1_1FileNode.html	returns the number of elements in the node, if it is a sequence or mapping, or 1 otherwise.			1									
core_FileNode_toInt	CVAPI(ExceptionStatus) core_FileNode_toInt(cv::FileNode *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(*obj); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_toFloat	CVAPI(ExceptionStatus) core_FileNode_toFloat(cv::FileNode *obj, float *returnValue) { BEGIN_WRAP *returnValue = static_cast<float>(*obj); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_toDouble	CVAPI(ExceptionStatus) core_FileNode_toDouble(cv::FileNode *obj, double *returnValue) { BEGIN_WRAP *returnValue = static_cast<double>(*obj); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							

core_FileNode_toString	CVAPI(ExceptionStatus) core_FileNode_toString(cv::FileNode *obj, std::string *buf) { BEGIN_WRAP buf->assign(*obj); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_toMat	CVAPI(ExceptionStatus) core_FileNode_toMat(cv::FileNode *obj, cv::Mat *m) { BEGIN_WRAP (*obj) >> (*m); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_begin	CVAPI(ExceptionStatus) core_FileNode_begin(cv::FileNode *obj, cv::FileNodeIterator **returnValue) { BEGIN_WRAP *returnValue = new cv::FileNodeIterator(obj->begin()); END_WRAP }	cv::FileNode::begin()	classcv_1_1FileNode.html	returns iterator pointing to the first node element		Examples: samples/cpp/filestorage.cpp.	1								
core_FileNode_end	CVAPI(ExceptionStatus) core_FileNode_end(cv::FileNode *obj, cv::FileNodeIterator **returnValue) { BEGIN_WRAP *returnValue = new cv::FileNodeIterator(obj->end()); END_WRAP }	cv::FileNode::end()	classcv_1_1FileNode.html	returns iterator pointing to the element following the last node element		Examples: samples/cpp/filestorage.cpp.	1								
core_FileNode_readRaw	CVAPI(ExceptionStatus) core_FileNode_readRaw(cv::FileNode *obj, const char *fmt, uchar* vec, size_t len) { BEGIN_WRAP obj->readRaw(fmt, vec, len); END_WRAP }	cv::FileNode::readRaw()	classcv_1_1FileNode.html	Reads node elements to the buffer with the specified format.	fmt : Specification of each array element. See format specification vec : Pointer to the destination array. len : Number of bytes to read (buffer size limit). If it is greater than number of remaining elements then all of them will be read.	Usually it is more convenient to use operator >> instead of this method.	3								
core_FileNode_read_int	CVAPI(ExceptionStatus) core_FileNode_read_int(cv::FileNode *node, int *value, int default_value) { BEGIN_WRAP int temp; cv::read(*node, temp, default_value); *value = temp; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_float	CVAPI(ExceptionStatus) core_FileNode_read_float(cv::FileNode *node, float *value, float default_value) { BEGIN_WRAP float temp; cv::read(*node, temp, default_value); *value = temp; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_double	CVAPI(ExceptionStatus) core_FileNode_read_double(cv::FileNode *node, double *value, double default_value) { BEGIN_WRAP double temp; cv::read(*node, temp, default_value); *value = temp; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_String	CVAPI(ExceptionStatus) core_FileNode_read_String(cv::FileNode *node, std::string *value, const char *default_value) { BEGIN_WRAP cv::String str; cv::read(*node, str, (default_value == nullptr) ? cv::String() : cv::String(default_value)); value->assign(str); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Mat	CVAPI(ExceptionStatus) core_FileNode_read_Mat(cv::FileNode *node, cv::Mat *mat, cv::Mat *default_mat) { BEGIN_WRAP cv::read(*node, *mat, entity(default_mat)); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_SparseMat	CVAPI(ExceptionStatus) core_FileNode_read_SparseMat(cv::FileNode *node, cv::SparseMat *mat, cv::SparseMat *default_mat) { BEGIN_WRAP cv::read(*node, *mat, entity(default_mat)); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							

core_FileNode_read_vectorOfKeyPoint	CVAPI(ExceptionStatus) core_FileNode_read_vectorOfKeyPoint(cv::FileNode *node, std::vector<cv::KeyPoint> *keypoints) { BEGIN_WRAP cv::read(*node, *keypoints); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_vectorOfDMatch	CVAPI(ExceptionStatus) core_FileNode_read_vectorOfDMatch(cv::FileNode *node, std::vector<cv::DMatch> *matches) { BEGIN_WRAP cv::read(*node, *matches); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Range	CVAPI(ExceptionStatus) core_FileNode_read_Range(cv::FileNode *node, MyCvSlice *returnValue) { BEGIN_WRAP cv::Range ret; (*node) >> ret; *returnValue = c(ret); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_KeyPoint	CVAPI(ExceptionStatus) core_FileNode_read_KeyPoint(cv::FileNode *node, MyKeyPoint *returnValue) { BEGIN_WRAP cv::KeyPoint ret; (*node) >> ret; *returnValue = c(ret); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_DMatch	CVAPI(ExceptionStatus) core_FileNode_read_DMatch(cv::FileNode *node, MyDMatch *returnValue) { BEGIN_WRAP cv::DMatch ret; (*node) >> ret; *returnValue = c(ret); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Point2i	CVAPI(ExceptionStatus) core_FileNode_read_Point2i(cv::FileNode *node, MyCvPoint *returnValue) { BEGIN_WRAP cv::Point2i ret; (*node) >> ret; *returnValue = MyCvPoint(ret.x, ret.y); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Point2f	CVAPI(ExceptionStatus) core_FileNode_read_Point2f(cv::FileNode *node, MyCvPoint2D32f *returnValue) { BEGIN_WRAP cv::Point2f ret; (*node) >> ret; *returnValue = MyCvPoint2D32f(ret.x, ret.y); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Point2d	CVAPI(ExceptionStatus) core_FileNode_read_Point2d(cv::FileNode *node, MyCvPoint2D64f *returnValue) { BEGIN_WRAP cv::Point2d ret; (*node) >> ret; *returnValue = MyCvPoint2D64f(ret.x, ret.y); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Point3i	CVAPI(ExceptionStatus) core_FileNode_read_Point3i(cv::FileNode *node, MyCvPoint3D32i *returnValue) { BEGIN_WRAP cv::Point3i ret; (*node) >> ret; *returnValue = MyCvPoint3D32i(ret.x, ret.y, ret.z); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Point3f	CVAPI(ExceptionStatus) core_FileNode_read_Point3f(cv::FileNode *node, MyCvPoint3D32f *returnValue) { BEGIN_WRAP cv::Point3f ret; (*node) >> ret; *returnValue = MyCvPoint3D32f(ret.x, ret.y, ret.z); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							

core_FileNode_read_Point3d	CVAPI(ExceptionStatus) core_FileNode_read_Point3d(cv::FileNode *node, MyCvPoint3D64f *returnValue) { BEGIN_WRAP cv::Point3d ret; (*node) >> ret; *returnValue = MyCvPoint3D64f(ret.x, ret.y, ret.z); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Size2i	CVAPI(ExceptionStatus) core_FileNode_read_Size2i(cv::FileNode *node, MyCvSize *returnValue) { BEGIN_WRAP cv::Size2i ret; (*node) >> ret; *returnValue = MyCvSize(ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Size2f	CVAPI(ExceptionStatus) core_FileNode_read_Size2f(cv::FileNode *node, MyCvSize2D32f *returnValue) { BEGIN_WRAP cv::Size2f ret; (*node) >> ret; *returnValue = MyCvSize2D32f(ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Size2d	CVAPI(ExceptionStatus) core_FileNode_read_Size2d(cv::FileNode *node, MyCvSize2D64f *returnValue) { BEGIN_WRAP cv::Size2d ret; (*node) >> ret; *returnValue = MyCvSize2D64f(ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Rect2i	CVAPI(ExceptionStatus) core_FileNode_read_Rect2i(cv::FileNode *node, MyCvRect *returnValue) { BEGIN_WRAP cv::Rect2i ret; (*node) >> ret; *returnValue = MyCvRect(ret.x, ret.y, ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Rect2f	CVAPI(ExceptionStatus) core_FileNode_read_Rect2f(cv::FileNode *node, MyCvRect2D32f *returnValue) { BEGIN_WRAP cv::Rect2f ret; (*node) >> ret; *returnValue = MyCvRect2D32f(ret.x, ret.y, ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Rect2d	CVAPI(ExceptionStatus) core_FileNode_read_Rect2d(cv::FileNode *node, MyCvRect2D64f *returnValue) { BEGIN_WRAP cv::Rect2d ret; (*node) >> ret; *returnValue = MyCvRect2D64f(ret.x, ret.y, ret.width, ret.height); END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Scalar	CVAPI(ExceptionStatus) core_FileNode_read_Scalar(cv::FileNode *node, MyCvScalar *returnValue) { BEGIN_WRAP cv::Scalar s; (*node) >> s; *returnValue = MyCvScalar(); returnValue->val[0] = s.val[0]; returnValue->val[1] = s.val[1]; returnValue->val[2] = s.val[2]; returnValue->val[3] = s.val[3]; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec2i	CVAPI(ExceptionStatus) core_FileNode_read_Vec2i(cv::FileNode *node, CvVec2i *returnValue) { BEGIN_WRAP cv::Vec2i v; (*node) >> v; CvVec2i ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec3i	CVAPI(ExceptionStatus) core_FileNode_read_Vec3i(cv::FileNode *node, CvVec3i *returnValue) { BEGIN_WRAP cv::Vec3i v; (*node) >> v; CvVec3i ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							

core_FileNode_read_Vec4i	CVAPI(ExceptionStatus) core_FileNode_read_Vec4i(cv::FileNode *node, CvVec4i *returnValue) { BEGIN_WRAP cv::Vec4i v; (*node) >> v; CvVec4i ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec6i	CVAPI(ExceptionStatus) core_FileNode_read_Vec6i(cv::FileNode *node, CvVec6i *returnValue) { BEGIN_WRAP cv::Vec6i v; (*node) >> v; CvVec6i ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec2d	CVAPI(ExceptionStatus) core_FileNode_read_Vec2d(cv::FileNode *node, CvVec2d *returnValue) { BEGIN_WRAP cv::Vec2d v; (*node) >> v; CvVec2d ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec3d	CVAPI(ExceptionStatus) core_FileNode_read_Vec3d(cv::FileNode *node, CvVec3d *returnValue) { BEGIN_WRAP cv::Vec3d v; (*node) >> v; CvVec3d ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec4d	CVAPI(ExceptionStatus) core_FileNode_read_Vec4d(cv::FileNode *node, CvVec4d *returnValue) { BEGIN_WRAP cv::Vec4d v; (*node) >> v; CvVec4d ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec6d	CVAPI(ExceptionStatus) core_FileNode_read_Vec6d(cv::FileNode *node, CvVec6d *returnValue) { BEGIN_WRAP cv::Vec6d v; (*node) >> v; CvVec6d ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec2f	CVAPI(ExceptionStatus) core_FileNode_read_Vec2f(cv::FileNode *node, CvVec2f *returnValue) { BEGIN_WRAP cv::Vec2f v; (*node) >> v; CvVec2f ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec3f	CVAPI(ExceptionStatus) core_FileNode_read_Vec3f(cv::FileNode *node, CvVec3f *returnValue) { BEGIN_WRAP cv::Vec3f v; (*node) >> v; CvVec3f ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec4f	CVAPI(ExceptionStatus) core_FileNode_read_Vec4f(cv::FileNode *node, CvVec4f *returnValue) { BEGIN_WRAP cv::Vec4f v; (*node) >> v; CvVec4f ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNode_read_Vec6f	CVAPI(ExceptionStatus) core_FileNode_read_Vec6f(cv::FileNode *node, CvVec6f *returnValue) { BEGIN_WRAP cv::Vec6f v; (*node) >> v; CvVec6f ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•							

core_FileNode_read_Vec2b	CVAPI(ExceptionStatus) core_FileNode_read_Vec2b(cv::FileNode *node, CvVec2b *returnValue) { BEGIN_WRAP cv::Vec2b v; (*node) >> v; CvVec2b ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec3b	CVAPI(ExceptionStatus) core_FileNode_read_Vec3b(cv::FileNode *node, CvVec3b *returnValue) { BEGIN_WRAP cv::Vec3b v; (*node) >> v; CvVec3b ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec4b	CVAPI(ExceptionStatus) core_FileNode_read_Vec4b(cv::FileNode *node, CvVec4b *returnValue) { BEGIN_WRAP cv::Vec4b v; (*node) >> v; CvVec4b ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec6b	CVAPI(ExceptionStatus) core_FileNode_read_Vec6b(cv::FileNode *node, CvVec6b *returnValue) { BEGIN_WRAP cv::Vec6b v; (*node) >> v; CvVec6b ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec2s	CVAPI(ExceptionStatus) core_FileNode_read_Vec2s(cv::FileNode *node, CvVec2s *returnValue) { BEGIN_WRAP cv::Vec2s v; (*node) >> v; CvVec2s ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec3s	CVAPI(ExceptionStatus) core_FileNode_read_Vec3s(cv::FileNode *node, CvVec3s *returnValue) { BEGIN_WRAP cv::Vec3s v; (*node) >> v; CvVec3s ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec4s	CVAPI(ExceptionStatus) core_FileNode_read_Vec4s(cv::FileNode *node, CvVec4s *returnValue) { BEGIN_WRAP cv::Vec4s v; (*node) >> v; CvVec4s ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec6s	CVAPI(ExceptionStatus) core_FileNode_read_Vec6s(cv::FileNode *node, CvVec6s *returnValue) { BEGIN_WRAP cv::Vec6s v; (*node) >> v; CvVec6s ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec2w	CVAPI(ExceptionStatus) core_FileNode_read_Vec2w(cv::FileNode *node, CvVec2w *returnValue) { BEGIN_WRAP cv::Vec2w v; (*node) >> v; CvVec2w ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							
core_FileNode_read_Vec3w	CVAPI(ExceptionStatus) core_FileNode_read_Vec3w(cv::FileNode *node, CvVec3w *returnValue) { BEGIN_WRAP cv::Vec3w v; (*node) >> v; CvVec3w ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.				•							

core_FileNode_read_Vec4w	CVAPI(ExceptionStatus) core_FileNode_read_Vec4w(cv::FileNode *node, CvVec4w *returnValue) { BEGIN_WRAP cv::Vec4w v; (*node) >> v; CvVec4w ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•								
core_FileNode_read_Vec6w	CVAPI(ExceptionStatus) core_FileNode_read_Vec6w(cv::FileNode *node, CvVec6w *returnValue) { BEGIN_WRAP cv::Vec6w v; (*node) >> v; CvVec6w ret; std::copy(std::begin(v.val), std::end(v.val), std::begin(ret.val)); *returnValue = ret; END_WRAP }	cv::FileNode	classcv_1_1FileNode.html	File Storage Node class. The node is used to store each and every element of the file storage opened for reading. When XML/YAML file is read, it is first parsed and stored in the memory as a hierarchical collection of nodes. Each node can be a "leaf" that is contain a single number or a string, or be a collection of other nodes. There can be named collections (mappings) where each element has a name and it is accessed by a name, and ordered collections (sequences) where elements do not have names but rather accessed by index. Type of the file node can be determined using FileNode::type method. Note that file nodes are only used for navigating file storages opened for reading. When a file storage is opened for writing, no data is stored in memory after it is written. Examples: samples/cpp/filestorage.cpp.					•								
core_FileNodeIterator_new1	CVAPI(ExceptionStatus) core_FileNodeIterator_new1(cv::FileNodeIterator **returnValue) { BEGIN_WRAP *returnValue = new cv::FileNodeIterator; END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.								•				
core_FileNodeIterator_delete	CVAPI(ExceptionStatus) core_FileNodeIterator_delete(cv::FileNodeIterator *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.									•			
core_FileNodeIterator_operatorAsterisk	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorAsterisk(cv::FileNodeIterator *obj, cv::FileNode **returnValue) { BEGIN_WRAP *returnValue = new cv::FileNode(**obj)); END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNodeIterator_operatorIncrement	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorIncrement(cv::FileNodeIterator *obj, int *returnValue) { BEGIN_WRAP const size_t prev_remaining = obj->remaining(); ++(*obj); *returnValue = (prev_remaining == obj->remaining()) ? 0 : 1; END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNodeIterator_operatorPlusEqual	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorPlusEqual(cv::FileNodeIterator *obj, int ofs, int *returnValue) { BEGIN_WRAP const size_t prev_remaining = obj->remaining(); (*obj) += ofs; *returnValue = (prev_remaining == obj->remaining()) ? 0 : 1; END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNodeIterator_readRaw	CVAPI(ExceptionStatus) core_FileNodeIterator_readRaw(cv::FileNodeIterator *obj, const char *fmt, uchar* vec, size_t maxCount) { BEGIN_WRAP obj->readRaw(fmt, vec, maxCount); END_WRAP }	cv::FileNodeIterator::readRaw()	classcv_1_1FileNodeIterator.html	Reads node elements to the buffer with the specified format.	fmt : Specification of each array element. See format specification vec : Pointer to the destination array. len : Number of bytes to read (buffer size limit). If it is greater than number of remaining elements then all of them will be read.	Usually it is more convenient to use operator >> instead of this method.	3									
core_FileNodeIterator_operatorEqual	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorEqual(cv::FileNodeIterator *it1, cv::FileNodeIterator *it2, int *returnValue) { BEGIN_WRAP *returnValue = ((*it1) == (*it2)) ? 1 : 0; END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNodeIterator_operatorMinus	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorMinus(cv::FileNodeIterator *it1, cv::FileNodeIterator *it2, ptrdiff_t *returnValue) { BEGIN_WRAP *returnValue = (*it1) - (*it2); END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileNodeIterator_operatorLessThan	CVAPI(ExceptionStatus) core_FileNodeIterator_operatorLessThan(cv::FileNodeIterator *it1, cv::FileNodeIterator *it2, int *returnValue) { BEGIN_WRAP *returnValue = (*it1) < (*it2); END_WRAP }	cv::FileNodeIterator	classcv_1_1FileNodeIterator.html	used to iterate through sequences and mappings. A standard STL notation, with node.begin(), node.end() denoting the beginning and the end of a sequence, stored in node. See the data reading sample in the beginning of the section. Examples: samples/cpp/filestorage.cpp.					•							
core_FileStorage_new1	CVAPI(ExceptionStatus) core_FileStorage_new1(cv::FileStorage **returnValue) { BEGIN_WRAP *returnValue = new cv::FileStorage; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.								•			
core_FileStorage_new2	CVAPI(ExceptionStatus) core_FileStorage_new2(const char *source, int flags, const char *encoding, cv::FileStorage **returnValue) { BEGIN_WRAP std::string encodingStr; if (encoding != nullptr) encodingStr = std::string(encoding); *returnValue = new cv::FileStorage(source, flags, encodingStr); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.								•			
core_FileStorage_delete	CVAPI(ExceptionStatus) core_FileStorage_delete(cv::FileStorage *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.									•		
core_FileStorage_open	CVAPI(ExceptionStatus) core_FileStorage_open(cv::FileStorage *obj, const char *filename, int flags, const char *encoding, int *returnValue) { BEGIN_WRAP std::string encodingStr; if (encoding != nullptr) encodingStr = std::string(encoding); *returnValue = obj->open(filename, flags, encodingStr) ? 1 : 0; END_WRAP }	cv::FileStorage::open()	classcv_1_1FileStorage.html	Opens a file.	filename : Name of the file to open or the text string to read the data from. Extension of the file (.xml, .yml/.yaml or .json) determines its format (XML, YAML or JSON respectively). Also you can append .gz to work with compressed files, for example myHugeMatrix.xml.gz. If both FileStorage::WRITE and FileStorage::MEMORY flags are specified, source is used just to specify the output file format (e.g. mydata.xml, .yml etc.). A file name can also contain parameters. You can use this format, "*?base64" (e.g. "file.json?base64" (case sensitive)), as an alternative to FileStorage::BASE64 flag. flags : Mode of operation. One of FileStorage::Mode encoding : Encoding of the file. Note that UTF-16 XML encoding is not supported currently and you should use 8-bit encoding instead of it.	See description of parameters in FileStorage::FileStorage. The method calls FileStorage::release before opening the file.	3									

core_FileStorage_isOpened	CVAPI(ExceptionStatus) core_FileStorage_isOpened(cv::FileStorage *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isOpened() ? 1 : 0; END_WRAP }	cv::FileStorage::isOpened()	classcv_1_1FileStorage.html	Checks whether the file is opened.		Examples: samples/cpp/filestorage.cpp.	1									
core_FileStorage_releaseAndGetString	CVAPI(ExceptionStatus) core_FileStorage_releaseAndGetString(cv::FileStorage* obj, std::string * outString) { BEGIN_WRAP outString->assign(obj->releaseAndGetString()); END_WRAP }	cv::FileStorage::releaseAndGetString()	classcv_1_1FileStorage.html	Closes the file and releases all the memory buffers.		Call this method after all I/O operations with the storage are finished. If the storage was opened for writing data and FileStorage::WRITE was specifiedExamples: samples/cpp/filestorage.cpp.	1									
core_FileStorage_getFirstTopLevelNode	CVAPI(ExceptionStatus) core_FileStorage_getFirstTopLevelNode(cv::FileStorage *obj, cv::FileNode **returnValue) { BEGIN_WRAP const auto node = obj->getFirstTopLevelNode(); *returnValue = new cv::FileNode(node); END_WRAP }	cv::FileStorage::getFirstTopLevelNode()	classcv_1_1FileStorage.html	Returns the first element of the top-level mapping.			1									
core_FileStorage_root	CVAPI(ExceptionStatus) core_FileStorage_root(cv::FileStorage *obj, int streamIdx, cv::FileNode **returnValue) { BEGIN_WRAP const auto node = obj->root(streamIdx); *returnValue = new cv::FileNode(node); END_WRAP }	cv::FileStorage::root()	classcv_1_1FileStorage.html	Returns the top-level mapping.	streamidx : Zero-based index of the stream. In most cases there is only one stream in the file. However, YAML supports multiple streams and so there can be several.		1									
core_FileStorage_indexer	CVAPI(ExceptionStatus) core_FileStorage_indexer(cv::FileStorage *obj, const char *nodeName, cv::FileNode **returnValue) { BEGIN_WRAP const auto node = (*obj)[nodeName]; *returnValue = new cv::FileNode(node); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.			1		•							
core_FileStorage_writeRaw	CVAPI(ExceptionStatus) core_FileStorage_writeRaw(cv::FileStorage *obj, const char *fmt, const uchar *vec, size_t len) { BEGIN_WRAP obj->writeRaw(fmt, vec, len); END_WRAP }	cv::FileStorage::writeRaw()	classcv_1_1FileStorage.html	Writes multiple numbers.	fmt : Specification of each array element, see format specification vec : Pointer to the written array. len : Number of the uchar elements to write.	Writes one or more numbers of the specified format to the currently written structure. Usually it is more convenient to use operator << instead of this method.	3									
core_FileStorage_writeComment	CVAPI(ExceptionStatus) core_FileStorage_writeComment(cv::FileStorage *obj, const char *comment, int append) { BEGIN_WRAP obj->writeComment(comment, append != 0); END_WRAP }	cv::FileStorage::writeComment()	classcv_1_1FileStorage.html	Writes a comment.	comment : The written comment, single-line or multi-line append : If true, the function tries to put the comment at the end of current line. Else if the comment is multi-line, or if it does not fit at the end of the current line, the comment starts a new line.	The function writes a comment into file storage. The comments are skipped when the storage is read.	2									
core_FileStorage_getDefaultObjectName	CVAPI(ExceptionStatus) core_FileStorage_getDefaultObjectName(const char *filename, std::string *buf) { BEGIN_WRAP buf->assign(cv::FileStorage::getDefaultObjectName(filename)); END_WRAP }	cv::FileStorage::getDefaultObjectName()	classcv_1_1FileStorage.html	Returns the normalized object name for the specified name of a file.	filename : Name of a file		1									
core_FileStorage_elname	CVAPI(ExceptionStatus) core_FileStorage_elname(cv::FileStorage *obj, std::string *returnValue) { BEGIN_WRAP returnValue->assign(obj->elname); END_WRAP }	cv::FileStorage::elname	classcv_1_1FileStorage.html							•						
core_FileStorage_startWriteStruct	CVAPI(ExceptionStatus) core_FileStorage_startWriteStruct(cv::FileStorage *obj, const char* name, int flags, const char *typeName) { BEGIN_WRAP obj->startWriteStruct(name, flags, typeName); END_WRAP }	cv::FileStorage::startWriteStruct()	classcv_1_1FileStorage.html	Starts to write a nested structure (sequence or a mapping).	name : name of the structure. When writing to sequences (a.k.a. "arrays"), pass an empty string. flags : type of the structure (FileNode::MAP or FileNode::SEQ (both with optional FileNode::FLOW)). typeName : optional name of the type you store. The effect of setting this depends on the storage format. I.e. if the format has a specification for storing type information, this parameter is used.		3									
core_FileStorage_endWriteStruct	CVAPI(ExceptionStatus) core_FileStorage_endWriteStruct(cv::FileStorage *obj) { BEGIN_WRAP obj->endWriteStruct(); END_WRAP }	cv::FileStorage::endWriteStruct()	classcv_1_1FileStorage.html	Finishes writing nested structure (should pair startWriteStruct())			1									
core_FileStorage_state	CVAPI(ExceptionStatus) core_FileStorage_state(cv::FileStorage *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->state; END_WRAP }	cv::FileStorage::state	classcv_1_1FileStorage.html							•						
core_FileStorage_write_int	CVAPI(ExceptionStatus) core_FileStorage_write_int(cv::FileStorage *fs, const char *name, int value) { BEGIN_WRAP cv::write(*fs, cv::String(name), value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_write_float	CVAPI(ExceptionStatus) core_FileStorage_write_float(cv::FileStorage *fs, const char *name, float value) { BEGIN_WRAP cv::write(*fs, cv::String(name), value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_write_double	CVAPI(ExceptionStatus) core_FileStorage_write_double(cv::FileStorage *fs, const char *name, double value) { BEGIN_WRAP cv::write(*fs, cv::String(name), value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_write_String	CVAPI(ExceptionStatus) core_FileStorage_write_String(cv::FileStorage *fs, const char *name, const char *value) { BEGIN_WRAP cv::write(*fs, cv::String(name), value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_write_Mat	CVAPI(ExceptionStatus) core_FileStorage_write_Mat(cv::FileStorage *fs, const char *name, const cv::Mat *value) { BEGIN_WRAP cv::write(*fs, cv::String(name), *value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							

core_FileStorage_write_SparseMat	CVAPI(ExceptionStatus) core_FileStorage_write_SparseMat(cv::FileStorage *fs, const char *name, const cv::SparseMat *value) { BEGIN_WRAP cv::write(*fs, cv::String(name), *value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_write_vectorOfKeyPoint	CVAPI(ExceptionStatus) core_FileStorage_write_vectorOfKeyPoint(cv::FileStorage *fs, const char *name, const std::vector<cv::KeyPoint> *value) { BEGIN_WRAP cv::write(*fs, cv::String(name), *value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_write_vectorOfDMatch	CVAPI(ExceptionStatus) core_FileStorage_write_vectorOfDMatch(cv::FileStorage *fs, const char *name, const std::vector<cv::DMatch> *value) { BEGIN_WRAP cv::write(*fs, cv::String(name), *value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_writeScalar_int	CVAPI(ExceptionStatus) core_FileStorage_writeScalar_int(cv::FileStorage *fs, int value) { BEGIN_WRAP cv::writeScalar(*fs, value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_writeScalar_float	CVAPI(ExceptionStatus) core_FileStorage_writeScalar_float(cv::FileStorage *fs, float value) { BEGIN_WRAP cv::writeScalar(*fs, value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_writeScalar_double	CVAPI(ExceptionStatus) core_FileStorage_writeScalar_double(cv::FileStorage *fs, double value) { BEGIN_WRAP cv::writeScalar(*fs, value); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_writeScalar_String	CVAPI(ExceptionStatus) core_FileStorage_writeScalar_String(cv::FileStorage *fs, const char *val) { BEGIN_WRAP cv::writeScalar(*fs, cv::String(val)); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_String	CVAPI(ExceptionStatus) core_FileStorage_shift_String(cv::FileStorage *fs, const char *val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_int	CVAPI(ExceptionStatus) core_FileStorage_shift_int(cv::FileStorage *fs, int val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_float	CVAPI(ExceptionStatus) core_FileStorage_shift_float(cv::FileStorage *fs, float val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_double	CVAPI(ExceptionStatus) core_FileStorage_shift_double(cv::FileStorage *fs, double val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_Mat	CVAPI(ExceptionStatus) core_FileStorage_shift_Mat(cv::FileStorage *fs, cv::Mat *val) { BEGIN_WRAP (*fs) << *val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_SparseMat	CVAPI(ExceptionStatus) core_FileStorage_shift_SparseMat(cv::FileStorage *fs, cv::SparseMat *val) { BEGIN_WRAP (*fs) << *val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_Range	CVAPI(ExceptionStatus) core_FileStorage_shift_Range(cv::FileStorage *fs, MyCvSlice val) { BEGIN_WRAP (*fs) << cpp(val); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_KeyPoint	CVAPI(ExceptionStatus) core_FileStorage_shift_KeyPoint(cv::FileStorage *fs, cv::KeyPoint val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_DMatch	CVAPI(ExceptionStatus) core_FileStorage_shift_DMatch(cv::FileStorage *fs, cv::DMatch val) { BEGIN_WRAP (*fs) << val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						
core_FileStorage_shift_vectorOfKeyPoint	CVAPI(ExceptionStatus) core_FileStorage_shift_vectorOfKeyPoint(cv::FileStorage *fs, std::vector<cv::KeyPoint> *val) { BEGIN_WRAP (*fs) << *val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•						

core_FileStorage_shift_vectorOfDMatch	CVAPI(ExceptionStatus) core_FileStorage_shift_vectorOfDMatch(cv::FileStorage *fs, std::vector<cv::DMatch> *val) { BEGIN_WRAP (*fs) << *val; END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point2i	CVAPI(ExceptionStatus) core_FileStorage_shift_Point2i(cv::FileStorage *fs, MyCvPoint val) { BEGIN_WRAP (*fs) << cv::Point2i(val.x, val.y); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point2f	CVAPI(ExceptionStatus) core_FileStorage_shift_Point2f(cv::FileStorage *fs, MyCvPoint2D32f val) { BEGIN_WRAP (*fs) << cv::Point2f(val.x, val.y); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point2d	CVAPI(ExceptionStatus) core_FileStorage_shift_Point2d(cv::FileStorage *fs, MyCvPoint2D64f val) { BEGIN_WRAP (*fs) << cv::Point2d(val.x, val.y); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point3i	CVAPI(ExceptionStatus) core_FileStorage_shift_Point3i(cv::FileStorage *fs, MyCvPoint3D32i val) { BEGIN_WRAP (*fs) << cv::Point3i(val.x, val.y, val.z); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point3f	CVAPI(ExceptionStatus) core_FileStorage_shift_Point3f(cv::FileStorage *fs, MyCvPoint3D32f val) { BEGIN_WRAP (*fs) << cv::Point3f(val.x, val.y, val.z); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Point3d	CVAPI(ExceptionStatus) core_FileStorage_shift_Point3d(cv::FileStorage *fs, MyCvPoint3D64f val) { BEGIN_WRAP (*fs) << cv::Point3d(val.x, val.y, val.z); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Size2i	CVAPI(ExceptionStatus) core_FileStorage_shift_Size2i(cv::FileStorage *fs, MyCvSize val) { BEGIN_WRAP (*fs) << cv::Size2i(val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Size2f	CVAPI(ExceptionStatus) core_FileStorage_shift_Size2f(cv::FileStorage *fs, MyCvSize2D32f val) { BEGIN_WRAP (*fs) << cv::Size2f(val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Size2d	CVAPI(ExceptionStatus) core_FileStorage_shift_Size2d(cv::FileStorage *fs, MyCvSize2D64f val) { BEGIN_WRAP (*fs) << cv::Size2d(val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Rect2i	CVAPI(ExceptionStatus) core_FileStorage_shift_Rect2i(cv::FileStorage *fs, MyCvRect val) { BEGIN_WRAP (*fs) << cv::Rect2i(val.x, val.y, val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Rect2f	CVAPI(ExceptionStatus) core_FileStorage_shift_Rect2f(cv::FileStorage *fs, MyCvRect2D32f val) { BEGIN_WRAP (*fs) << cv::Rect2f(val.x, val.y, val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Rect2d	CVAPI(ExceptionStatus) core_FileStorage_shift_Rect2d(cv::FileStorage *fs, MyCvRect2D64f val) { BEGIN_WRAP (*fs) << cv::Rect2d(val.x, val.y, val.width, val.height); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Scalar	CVAPI(ExceptionStatus) core_FileStorage_shift_Scalar(cv::FileStorage *fs, MyCvScalar val) { BEGIN_WRAP (*fs) << cpp(val); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec2i	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2i(cv::FileStorage *fs, CvVec2iv) { BEGIN_WRAP (*fs) << cv::Vec2i(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3i	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3i(cv::FileStorage *fs, CvVec3iv) { BEGIN_WRAP (*fs) << cv::Vec3i(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4i	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4i(cv::FileStorage *fs, CvVec4iv) { BEGIN_WRAP (*fs) << cv::Vec4i(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							

core_FileStorage_shift_Vec6i	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6i(cv::FileStorage *fs, CvVec6i v) { BEGIN_WRAP (*fs) < cv::Vec6i(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec2d	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2d(cv::FileStorage *fs, CvVec2d v) { BEGIN_WRAP (*fs) < cv::Vec2d(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3d	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3d(cv::FileStorage *fs, CvVec3d v) { BEGIN_WRAP (*fs) < cv::Vec3d(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4d	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4d(cv::FileStorage *fs, CvVec4d v) { BEGIN_WRAP (*fs) < cv::Vec4d(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec6d	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6d(cv::FileStorage *fs, CvVec6d v) { BEGIN_WRAP (*fs) < cv::Vec6d(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec2f	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2f(cv::FileStorage *fs, CvVec2f v) { BEGIN_WRAP (*fs) < cv::Vec2f(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3f	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3f(cv::FileStorage *fs, CvVec3f v) { BEGIN_WRAP (*fs) < cv::Vec3f(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4f	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4f(cv::FileStorage *fs, CvVec4f v) { BEGIN_WRAP (*fs) < cv::Vec4f(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec6f	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6f(cv::FileStorage *fs, CvVec6f v) { BEGIN_WRAP (*fs) < cv::Vec6f(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec2b	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2b(cv::FileStorage *fs, CvVec2b v) { BEGIN_WRAP (*fs) < cv::Vec2b(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3b	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3b(cv::FileStorage *fs, CvVec3b v) { BEGIN_WRAP (*fs) < cv::Vec3b(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4b	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4b(cv::FileStorage *fs, CvVec4b v) { BEGIN_WRAP (*fs) < cv::Vec4b(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec6b	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6b(cv::FileStorage *fs, CvVec6b v) { BEGIN_WRAP (*fs) < cv::Vec6b(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec2s	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2s(cv::FileStorage *fs, CvVec2s v) { BEGIN_WRAP (*fs) < cv::Vec2s(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3s	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3s(cv::FileStorage *fs, CvVec3s v) { BEGIN_WRAP (*fs) < cv::Vec3s(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4s	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4s(cv::FileStorage *fs, CvVec4s v) { BEGIN_WRAP (*fs) < cv::Vec4s(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec6s	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6s(cv::FileStorage *fs, CvVec6s v) { BEGIN_WRAP (*fs) < cv::Vec6s(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homograph y/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homograph y/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homograph y/pose_from_homography.cpp.					•							

core_FileStorage_shift_Vec2w	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec2w ((cv::FileStorage *fs, CvVec2w v) { BEGIN_WRAP (*fs) << cv::Vec2w(v.val[0], v.val[1]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec3w	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec3w ((cv::FileStorage *fs, CvVec3w v) { BEGIN_WRAP (*fs) << cv::Vec3w(v.val[0], v.val[1], v.val[2]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec4w	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec4w ((cv::FileStorage *fs, CvVec4w v) { BEGIN_WRAP (*fs) << cv::Vec4w(v.val[0], v.val[1], v.val[2], v.val[3]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_FileStorage_shift_Vec6w	CVAPI(ExceptionStatus) core_FileStorage_shift_Vec6w ((cv::FileStorage *fs, CvVec6w v) { BEGIN_WRAP (*fs) << cv::Vec6w(v.val[0], v.val[1], v.val[2], v.val[3], v.val[4], v.val[5]); END_WRAP }	cv::FileStorage	classcv_1_1FileStorage.html	XML/YAML/JSON file storage class that encapsulates all the information necessary for writing or reading data to/from a file. Examples: samples/cpp/filestorage.cpp, samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, and samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.					•							
core_InputArray_new_byMat	CVAPI(ExceptionStatus) core_InputArray_new_byMat(cv::Mat *mat, cv::_InputArray *returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(*mat); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				
core_InputArray_new_byUMat	CVAPI(ExceptionStatus) core_InputArray_new_byUMat ((cv::UMat* mat, cv::_InputArray** returnValue) { BEGIN_WRAP * returnValue = new cv::_InputArray(*mat); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				
core_InputArray_new_byMatExpr	CVAPI(ExceptionStatus) core_InputArray_new_byMatExpr(cv::MatExpr *expr, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(*expr); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				
core_InputArray_new_byScalar	CVAPI(ExceptionStatus) core_InputArray_new_byScalar(MyCvScalar s, cv::Scalar **handle, cv::_InputArray **returnValue) { BEGIN_WRAP *handle = new cv::Scalar(s.val[0], s.val[1], s.val[2], s.val[3]); *returnValue = new cv::_InputArray(**handle); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				
core_InputArray_new_byDouble	CVAPI(ExceptionStatus) core_InputArray_new_byDouble(double *handle, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(*handle); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				
core_InputArray_new_byVectorOfMat	CVAPI(ExceptionStatus) core_InputArray_new_byVectorOfMat(std::vector<cv::Mat> *vector, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(*vector); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•				

core_InputArray_new_byVecb	CVAPI(ExceptionStatus) core_InputArray_new_byVecb (uchar *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_new_byVecs	CVAPI(ExceptionStatus) core_InputArray_new_byVecs (short *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_new_byVecw	CVAPI(ExceptionStatus) core_InputArray_new_byVecw (ushort *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_new_byVeci	CVAPI(ExceptionStatus) core_InputArray_new_byVeci(int *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_new_byVecf	CVAPI(ExceptionStatus) core_InputArray_new_byVecf(float *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_new_byVecd	CVAPI(ExceptionStatus) core_InputArray_new_byVecd (double *vec, int n, cv::_InputArray **returnValue) { BEGIN_WRAP *returnValue = new cv::_InputArray(vec, n); END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class								•			
core_InputArray_delete	CVAPI(ExceptionStatus) core_InputArray_delete(cv::_I nputArray *ia) { BEGIN_WRAP delete ia; END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class									•		

core_InputArray_delete_withScalar	CVAPI(ExceptionStatus) core_InputArray_delete_withScalar(cv::_InputArray *ia, cv::Scalar *handle) { BEGIN_WRAP delete ia; delete handle; END_WRAP }	cv::_InputArray	classcv_1_1__InputArray.html	This is the proxy class for passing read-only input arrays into OpenCV functions. It is defined as: typedef const _InputArray& InputArray; where _InputArray is a class that can be constructed from Mat, Mat_<T>, Matx<T, m, n>, std::vector<T>, std::vector<std::vector<T> >, std::vector<Mat>, std::vector<Mat_<T> >, UMat, std::vector<UMat> or double. It can also be constructed from a matrix expression. Since this is mostly implementation-level class, and its interface may change in future versions, we do not describe it in details. There are a few key things, though, that should be kept in mind: When you see in the reference manual or in OpenCV source code a function that takes InputArray, it means that you can actually pass Mat, Matx, vector<T> etc. (see above the complete list). Optional input arguments: If some of the input arrays may be empty, pass cv::noArray() (or simply cv::Mat() as you probably did before). The class is designed solely for passing parameters. That is, normally you should not declare class members, local and global variables of this type. If you want to design your own function or a class										•		
core_InputArray_getMat	CVAPI(ExceptionStatus) core_InputArray_getMat(cv::_InputArray *ia, int idx, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(ia->getMat(idx)); END_WRAP }	cv::_InputArray::getMat()	classcv_1_1__InputArray.html	Examples: samples/cpp/pca.cpp.			1									
core_InputArray_getUMat	CVAPI(ExceptionStatus) core_InputArray_getUMat(cv::_InputArray* ia, int idx, cv::UMat** returnValue) { BEGIN_WRAP * returnValue = new cv::UMat(ia->getUMat(idx)); END_WRAP }	cv::_InputArray::getUMat()	classcv_1_1__InputArray.html				1									
core_InputArray_getMatVector	CVAPI(ExceptionStatus) core_InputArray_getMatVector(cv::_InputArray *ia, std::vector<cv::Mat> *mv) { BEGIN_WRAP ia->getMatVector(*mv); END_WRAP }	cv::_InputArray::getMatVector()	classcv_1_1__InputArray.html				1									
core_InputArray_getFlags	CVAPI(ExceptionStatus) core_InputArray_getFlags(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->getFlags(); END_WRAP }	cv::_InputArray::getFlags()	classcv_1_1__InputArray.html				1									
core_InputArray_getObj	CVAPI(ExceptionStatus) core_InputArray_getObj(cv::_InputArray *ia, void **returnValue) { BEGIN_WRAP *returnValue = ia->getObj(); END_WRAP }	cv::_InputArray::getObj()	classcv_1_1__InputArray.html				1									
core_InputArray_getSz	CVAPI(ExceptionStatus) core_InputArray_getSz(cv::_InputArray *ia, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(ia->getSz()); END_WRAP }	cv::_InputArray::getSz()	classcv_1_1__InputArray.html				1									
core_InputArray_kind	CVAPI(ExceptionStatus) core_InputArray_kind(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->kind(); END_WRAP }	cv::_InputArray::kind()	classcv_1_1__InputArray.html				1									
core_InputArray_dims	CVAPI(ExceptionStatus) core_InputArray_dims(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->dims(i); END_WRAP }	cv::_InputArray::dims()	classcv_1_1__InputArray.html				1									
core_InputArray_cols	CVAPI(ExceptionStatus) core_InputArray_cols(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->cols(i); END_WRAP }	cv::_InputArray::cols()	classcv_1_1__InputArray.html				1									
core_InputArray_rows	CVAPI(ExceptionStatus) core_InputArray_rows(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->rows(i); END_WRAP }	cv::_InputArray::rows()	classcv_1_1__InputArray.html				1									
core_InputArray_size	CVAPI(ExceptionStatus) core_InputArray_size(cv::_InputArray *ia, int i, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(ia->size(i)); END_WRAP }	cv::_InputArray::size()	classcv_1_1__InputArray.html				1									
core_InputArray_sizend	CVAPI(ExceptionStatus) core_InputArray_sizend(cv::_InputArray *ia, int* sz, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->sizend(sz, i); END_WRAP }	cv::_InputArray::sizend()	classcv_1_1__InputArray.html				2									
core_InputArray_sameSize	CVAPI(ExceptionStatus) core_InputArray_sameSize(cv::_InputArray *self, cv::_InputArray * target, int *returnValue) { BEGIN_WRAP *returnValue = self->sameSize(*target) ? 1 : 0; END_WRAP }	cv::_InputArray::sameSize()	classcv_1_1__InputArray.html				1									
core_InputArray_total	CVAPI(ExceptionStatus) core_InputArray_total(cv::_InputArray *ia, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = ia->total(i); END_WRAP }	cv::_InputArray::total()	classcv_1_1__InputArray.html				1									
core_InputArray_type	CVAPI(ExceptionStatus) core_InputArray_type(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->type(i); END_WRAP }	cv::_InputArray::type()	classcv_1_1__InputArray.html				1									
core_InputArray_depth	CVAPI(ExceptionStatus) core_InputArray_depth(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->depth(i); END_WRAP }	cv::_InputArray::depth()	classcv_1_1__InputArray.html				1									
core_InputArray_channels	CVAPI(ExceptionStatus) core_InputArray_channels(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->channels(i); END_WRAP }	cv::_InputArray::channels()	classcv_1_1__InputArray.html				1									

core_InputArray_isContinuous	CVAPI(ExceptionStatus) core_InputArray_isContinuous(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->isContinuous() ? 1 : 0; END_WRAP }	cv::_InputArray::isContinuous()	classcv_1_1__InputArray.html				1										
core_InputArray_isSubmatrix	CVAPI(ExceptionStatus) core_InputArray_isSubmatrix(cv::_InputArray *ia, int i, int *returnValue) { BEGIN_WRAP *returnValue = ia->isSubmatrix(i) ? 1 : 0; END_WRAP }	cv::_InputArray::isSubmatrix()	classcv_1_1__InputArray.html				1										
core_InputArray_empty	CVAPI(ExceptionStatus) core_InputArray_empty(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->empty() ? 1 : 0; END_WRAP }	cv::_InputArray::empty()	classcv_1_1__InputArray.html				1										
core_InputArray_copyTo1	CVAPI(ExceptionStatus) core_InputArray_copyTo1(cv::_InputArray *ia, cv::_OutputArray *arr) { BEGIN_WRAP ia->copyTo(*arr); END_WRAP }	cv::_InputArray::copyTo()	classcv_1_1__InputArray.html				1										
core_InputArray_copyTo2	CVAPI(ExceptionStatus) core_InputArray_copyTo2(cv::_InputArray *ia, cv::_OutputArray *arr, cv::_InputArray *mask) { BEGIN_WRAP ia->copyTo(*arr, *mask); END_WRAP }	cv::_InputArray::copyTo()	classcv_1_1__InputArray.html				2										
core_InputArray_offset	CVAPI(ExceptionStatus) core_InputArray_offset(cv::_InputArray *ia, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = ia->offset(i); END_WRAP }	cv::_InputArray::offset()	classcv_1_1__InputArray.html				1										
core_InputArray_step	CVAPI(ExceptionStatus) core_InputArray_step(cv::_InputArray *ia, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = ia->step(i); END_WRAP }	cv::_InputArray::step()	classcv_1_1__InputArray.html				1										
core_InputArray_isMat	CVAPI(ExceptionStatus) core_InputArray_isMat(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isMat() ? 1 : 0; END_WRAP }	cv::_InputArray::isMat()	classcv_1_1__InputArray.html				1										
core_InputArray_isUMat	CVAPI(ExceptionStatus) core_InputArray_isUMat(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isUMat() ? 1 : 0; END_WRAP }	cv::_InputArray::isUMat()	classcv_1_1__InputArray.html				1										
core_InputArray_isMatVector	CVAPI(ExceptionStatus) core_InputArray_isMatVector(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isMatVector() ? 1 : 0; END_WRAP }	cv::_InputArray::isMatVector()	classcv_1_1__InputArray.html				1										
core_InputArray_isUMatVector	CVAPI(ExceptionStatus) core_InputArray_isUMatVector(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isUMatVector() ? 1 : 0; END_WRAP }	cv::_InputArray::isUMatVector()	classcv_1_1__InputArray.html				1										
core_InputArray_isMatx	CVAPI(ExceptionStatus) core_InputArray_isMatx(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isMat() ? 1 : 0; END_WRAP }	cv::_InputArray::isMatx()	classcv_1_1__InputArray.html				1										
core_InputArray_isVector	CVAPI(ExceptionStatus) core_InputArray_isVector(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isVector() ? 1 : 0; END_WRAP }	cv::_InputArray::isVector()	classcv_1_1__InputArray.html				1										
core_InputArray_isGpuMatVector	CVAPI(ExceptionStatus) core_InputArray_isGpuMatVector(cv::_InputArray *ia, int *returnValue) { BEGIN_WRAP *returnValue = ia->isGpuMatVector() ? 1 : 0; END_WRAP }	cv::_InputArray::isGpuMatVector()	classcv_1_1__InputArray.html				1										
core_Mat_sizeof	CVAPI(uint64) core_Mat_sizeof() { return sizeof(cv::Mat); }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i,0,...,0)...,(i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_(i_0,...,i_{M.dims-1})) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_(i,j)) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size				•							
core_Mat_new1	CVAPI(ExceptionStatus) core_Mat_new1(cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat; END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i,0,...,0)...,(i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_(i_0,...,i_{M.dims-1})) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_(i,j)) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•					

core_Mat_new2	CVAPI(ExceptionStatus) core_Mat_new2(int rows, int cols, int type, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(rows, cols, type); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new3	CVAPI(ExceptionStatus) core_Mat_new3(int rows, int cols, int type, MyCvScalar scalar, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(rows, cols, type, cpp(scalar)); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new4	CVAPI(ExceptionStatus) core_Mat_new4(cv::Mat *mat, MyCvSlice rowRange, MyCvSlice colRange, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(*mat, cpp(rowRange), cpp(colRange)); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new5	CVAPI(ExceptionStatus) core_Mat_new5(cv::Mat *mat, cv::Range rowRange, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(*mat, rowRange); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new6	CVAPI(ExceptionStatus) core_Mat_new6(cv::Mat *mat, cv::Range *ranges, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(*mat, ranges); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new7	CVAPI(ExceptionStatus) core_Mat_new7(cv::Mat *mat, MyCvRect roi, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(*mat, cpp(roi)); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new8	CVAPI(ExceptionStatus) core_Mat_new8(int rows, int cols, int type, void* data, size_t step, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(rows, cols, type, data, step); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			

core_Mat_new9	CVAPI(ExceptionStatus) core_Mat_new9(int ndims, const int* sizes, int type, void* data, const size_t* steps, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(ndims, sizes, type, data, steps); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new10	CVAPI(ExceptionStatus) core_Mat_new10(int ndims, int* sizes, int type, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(ndims, sizes, type); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new11	CVAPI(ExceptionStatus) core_Mat_new11(int ndims, int* sizes, int type, MyCvScalar s, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(ndims, sizes, type, cpp(s)); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_new12	CVAPI(ExceptionStatus) core_Mat_new12(cv::Mat *mat, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(*mat); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size										•			
core_Mat_delete	CVAPI(ExceptionStatus) core_Mat_delete(cv::Mat *self) { BEGIN_WRAP delete self; END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size											•		
core_Mat_getUMat	CVAPI(ExceptionStatus) core_Mat_getUMat(cv::Mat *self, cv::AccessFlag accessFlags, cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP * returnValue = new cv::UMat(self->getUMat(accessFlags, usageFlags)); END_WRAP }	cv::Mat::getUMat()	classcv_1_1Mat.html	retrieve UMat from Mat			2								
core_Mat_row	CVAPI(ExceptionStatus) core_Mat_row(cv::Mat *self, int y, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(self->row(y)); END_WRAP }	cv::Mat::row()	classcv_1_1Mat.html	Creates a matrix header for the specified matrix row.	y : A 0-based row index.	The method makes a new header for the specified matrix row and returns it. This is an O(1) operation, regardless of the matrix size. The underlying data of the new matrix is shared with the original matrix. Here is the example of one of the classical basic matrix processing operations, axpy, used by LU and many other algorithms:inline void matrix_axpy(Mat& A, int i, int j, double alpha){ A.row(j) += A.row(j)*alpha;}fragmentNote:In the current implementation, the following code does not work as expected: Mat A;...A.row(i) = A.row(j); // will not work This happens because A.row(i) forms a temporary header that is further assigned to another header. Remember that each of these operations is O(1), that is, no data is copied. Thus, the above assignment is not true if you may have expected the j-th row to be copied to the i-th row. To achieve that, you should either turn this simple assignment into an expression or use the Mat::copyTo method: Mat A;... // works, but looks a bit strange A.row(i) = A.row(j); // this is	1								
core_Mat_col	CVAPI(ExceptionStatus) core_Mat_col(cv::Mat *self, int x, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(self->col(x)); END_WRAP }	cv::Mat::col()	classcv_1_1Mat.html	Creates a matrix header for the specified matrix column.	x : A 0-based column index.	The method makes a new header for the specified matrix column and returns it. This is an O(1) operation, regardless of the matrix size. The underlying data of the new matrix is shared with the original matrix. See also the Mat::row description.Examples: samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	1								
core_Mat_rowRange	CVAPI(ExceptionStatus) core_Mat_rowRange(cv::Mat *self, int startRow, int endRow, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(self->rowRange(startRow, endRow)); END_WRAP }	cv::Mat::rowRange()	classcv_1_1Mat.html	Creates a matrix header for the specified row span.	startrow : An inclusive 0-based start index of the row span. endrow : An exclusive 0-based ending index of the row span.	The method makes a new header for the specified row span of the matrix. Similarly to Mat::row and Mat::col , this is an O(1) operation.Examples: samples/cpp/kmeans.cpp, and samples/dnn/segmentation.cpp.	2								

core_Mat_colRange	CVAPI(ExceptionStatus) core_Mat_colRange(cv::Mat *self, int startCol, int endCol, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(self->colRange(startCol, endCol)); END_WRAP }	cv::Mat::colRange()	classcv_1_1Mat.html	Creates a matrix header for the specified column span.	startcol : An inclusive 0-based start index of the column span. endcol : An exclusive 0-based ending index of the column span.	The method makes a new header for the specified column span of the matrix. Similarly to Mat::row and Mat::col , this is an O(1) operation.	2								
core_Mat_diag	CVAPI(ExceptionStatus) core_Mat_diag(cv::Mat *self, int d, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->diag(d); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::diag()	classcv_1_1Mat.html	Extracts a diagonal from a matrix.	d : index of the diagonal, with the following values: d=0 is the main diagonal. d<0 is a diagonal from the lower half. For example, d=-1 means the diagonal is set immediately below the main one. d>0 is a diagonal from the upper half. For example, d=1 means the diagonal is set immediately above the main one. For example: Mat m = (Mat_<int>(3,3) << 1,2,3, 4,5,6, 7,8,9);Mat d0 = m.diag(0);Mat d1 = m.diag(1);Mat d_1 = m.diag(-1); The resulting matrices are d0 = [1; 5; 9]d1 = [2; 6]d_1 = [4; 8]	The method makes a new header for the specified matrix diagonal. The new matrix is represented as a single-column matrix. Similarly to Mat::row and Mat::col, this is an O(1) operation.	1								
core_Mat_diag_static	CVAPI(ExceptionStatus) core_Mat_diag_static(cv::Mat *self, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::Mat::diag(*self); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::diag()	classcv_1_1Mat.html	Extracts a diagonal from a matrix.	d : index of the diagonal, with the following values: d=0 is the main diagonal. d<0 is a diagonal from the lower half. For example, d=-1 means the diagonal is set immediately below the main one. d>0 is a diagonal from the upper half. For example, d=1 means the diagonal is set immediately above the main one. For example: Mat m = (Mat_<int>(3,3) << 1,2,3, 4,5,6, 7,8,9);Mat d0 = m.diag(0);Mat d1 = m.diag(1);Mat d_1 = m.diag(-1); The resulting matrices are d0 = [1; 5; 9]d1 = [2; 6]d_1 = [4; 8]	The method makes a new header for the specified matrix diagonal. The new matrix is represented as a single-column matrix. Similarly to Mat::row and Mat::col, this is an O(1) operation.	1								
core_Mat_clone	CVAPI(ExceptionStatus) core_Mat_clone(cv::Mat *self, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->clone(); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::clone()	classcv_1_1Mat.html	Creates a full copy of the array and the underlying data.		The method creates a full copy of the array. The original step[] is not taken into account. So, the array copy is a continuous array occupying total()*elemSize() bytes.Examples: samples/cpp/create_mask.cpp, samples/cpp/facedetect.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/ImgTrans/houghlines.cpp, and	1								
core_Mat_copyTo1	CVAPI(ExceptionStatus) core_Mat_copyTo1(cv::Mat *self, cv::_OutputArray *m) { BEGIN_WRAP self->copyTo(*m); END_WRAP }	cv::Mat::copyTo()	classcv_1_1Mat.html	Copies the matrix to another one.	m : Destination matrix. If it does not have a proper size or type before the operation, it is reallocated.	The method copies the matrix data to another matrix. Before copying the data, the method invokes :m.create(this->size(), this->type());fragmentso that the destination matrix is reallocated if needed. While m.copyTo(m); works flawlessly, the function does not handle the case of a partial overlap between the source and the destination matrices.When the operation mask is specified, if the Mat::create call shown above reallocates the matrix, the newly allocated matrix is initialized with all zeros before copying the data.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/edge.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/lkdemo.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/watershed.cpp, samples/tapi/hog.cpp, and	1								
core_Mat_copyTo2	CVAPI(ExceptionStatus) core_Mat_copyTo2(cv::Mat *self, cv::_OutputArray *m, cv::_InputArray *mask) { BEGIN_WRAP self->copyTo(*m, entity(mask)); END_WRAP }	cv::Mat::copyTo()	classcv_1_1Mat.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	m : Destination matrix. If it does not have a proper size or type before the operation, it is reallocated. mask : Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels.		2								
core_Mat_copyTo_toMat1	CVAPI(ExceptionStatus) core_Mat_copyTo_toMat1(cv::Mat *self, cv::Mat *m) { BEGIN_WRAP self->copyTo(*m); END_WRAP }	cv::Mat::copyTo()	classcv_1_1Mat.html	Copies the matrix to another one.	m : Destination matrix. If it does not have a proper size or type before the operation, it is reallocated.	The method copies the matrix data to another matrix. Before copying the data, the method invokes :m.create(this->size(), this->type());fragmentso that the destination matrix is reallocated if needed. While m.copyTo(m); works flawlessly, the function does not handle the case of a partial overlap between the source and the destination matrices.When the operation mask is specified, if the Mat::create call shown above reallocates the matrix, the newly allocated matrix is initialized with all zeros before copying the data.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/edge.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/lkdemo.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/watershed.cpp, samples/tapi/hog.cpp, and	1								
core_Mat_copyTo_toMat2	CVAPI(ExceptionStatus) core_Mat_copyTo_toMat2(cv::Mat *self, cv::Mat *m, cv::_InputArray *mask) { BEGIN_WRAP self->copyTo(*m, entity(mask)); END_WRAP }	cv::Mat::copyTo()	classcv_1_1Mat.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	m : Destination matrix. If it does not have a proper size or type before the operation, it is reallocated. mask : Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels.		2								
core_Mat_convertTo	CVAPI(ExceptionStatus) core_Mat_convertTo(cv::Mat *self, cv::_OutputArray *m, int rtype, double alpha, double beta) { BEGIN_WRAP self->convertTo(*m, rtype, alpha, beta); END_WRAP }	cv::Mat::convertTo()	classcv_1_1Mat.html	Converts an array to another data type with optional scaling.	m : output matrix; if it does not have a proper size or type before the operation, it is reallocated. rtype : desired output matrix type or, rather, the depth since the number of channels are the same as the input has; if rtype is negative, the output matrix will have the same type as the input. alpha : optional scale factor. beta : optional delta added to the scaled values.	The method converts source pixel values to the target data type. saturate_cast<> is applied at the end to avoid possible overflows:\[m(x,y) = saturate _cast<rType>(\alpha (*this)(x,y) + \beta)\]Examples: samples/cpp/demhist.cpp, samples/cpp/dftrans.cpp, samples/cpp/fitellipse.cpp, samples/cpp/pca.cpp, samples/cpp/stitching_detailed.cpp, and samples/dnn/colorization.cpp.	4								
core_Mat_assignTo	CVAPI(ExceptionStatus) core_Mat_assignTo(cv::Mat *self, cv::Mat *m, int type) { BEGIN_WRAP self->assignTo(*m, type); END_WRAP }	cv::Mat::assignTo()	classcv_1_1Mat.html	Provides a functional form of convertTo.	m : Destination array. type : Desired destination array depth (or -1 if it should be the same as the source type).	This is an internally used method called by the MatrixExpressions engine.	2								
core_Mat_setTo_Scalar	CVAPI(ExceptionStatus) core_Mat_setTo_Scalar(cv::Mat *self, MyCvScalar value, cv::Mat *mask) { BEGIN_WRAP if (mask == nullptr) self->setTo(cpp(value)); else self->setTo(cpp(value), entity(mask)); END_WRAP }	cv::Mat::setTo()	classcv_1_1Mat.html	Sets all or some of the array elements to the specified value.	value : Assigned scalar converted to the actual array type. mask : Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels	This is an advanced variant of the Mat::operator=(const Scalar& s) operator.Examples: samples/cpp/stitching_detailed.cpp, and samples/dnn/segmentation.cpp.	2								
core_Mat_setTo_InputArray	CVAPI(ExceptionStatus) core_Mat_setTo_InputArray(cv::Mat *self, cv::_InputArray *value, cv::Mat *mask) { BEGIN_WRAP self->setTo(*value, entity(mask)); END_WRAP }	cv::Mat::setTo()	classcv_1_1Mat.html	Sets all or some of the array elements to the specified value.	value : Assigned scalar converted to the actual array type. mask : Operation mask of the same size as *this. Its non-zero elements indicate which matrix elements need to be copied. The mask has to be of type CV_8U and can have 1 or multiple channels	This is an advanced variant of the Mat::operator=(const Scalar& s) operator.Examples: samples/cpp/stitching_detailed.cpp, and samples/dnn/segmentation.cpp.	2								
core_Mat_reshape1	CVAPI(ExceptionStatus) core_Mat_reshape1(cv::Mat *self, int cn, int rows, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->reshape(cn, rows); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::reshape()	classcv_1_1Mat.html	Changes the shape and/or the number of channels of a 2D matrix without copying the data.	cn : New number of channels. If the parameter is 0, the number of channels remains the same. rows : New number of rows. If the parameter is 0, the number of rows remains the same.	The method makes a new matrix header for *this elements. The new matrix may have a different size and/or different number of channels. Any combination is possible if:No extra elements are included into the new matrix and no elements are excluded. Consequently, the product rows*cols*channels() must stay the same after the transformation. No data is copied. That is, this is an O(1) operation. Consequently, if you change the number of rows, or the operation changes the indices of elements row in some other way, the matrix must be continuous. See Mat::isContinuous .For example, if there is a set of 3D points stored as an STL vector, and you want to represent the points as a 3xN matrix, do the following:std::vector<Point3f> vec;...Mat pointMat = Mat(vec); // convert vector to Mat, O(1) operation reshape(1). // make Nx3 1-channel matrix out of Nx1 3-channel. // Also, an O(1) operation t(); // finally, transpose the Nx3 matrix. // This	2								

core_Mat_reshape2	CVAPI(ExceptionStatus) core_Mat_reshape2(cv::Mat *self, int cn, int newndims, const int* newsz, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->reshape(cn, newndims, newsz); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::reshape()	classcv_1_1Mat.html	Changes the shape and/or the number of channels of a 2D matrix without copying the data.	cn : New number of channels. If the parameter is 0, the number of channels remains the same. rows : New number of rows. If the parameter is 0, the number of rows remains the same.	The method makes a new matrix header for *this elements. The new matrix may have a different size and/or different number of channels. Any combination is possible if:No extra elements are included into the new matrix and no elements are excluded. Consequently, the product rows*cols*channels() must stay the same after the transformation.No data is copied. That is, this is an O(1) operation. Consequently, if you change the number of rows, or the operation changes the indices of elements row in some other way, the matrix must be continuous. See Mat::isContinuous .For example, if there is a set of 3D points stored as an STL vector, and you want to represent the points as a 3xN matrix, do the following:std::vector<Point3f> vec;...Mat pointMat = Mat(vec). // convert vector to Mat, O(1) operation reshape(1). // make Nx3 1-channel matrix out of Nx1 3-channel. // Also, an O(1) operation t(); // finally, transpose the Nx3 matrix. // This	3									
core_Mat_t	CVAPI(ExceptionStatus) core_Mat_t(cv::Mat *self, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = self->t(); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat::t()	classcv_1_1Mat.html	Transposes a matrix.		The method performs matrix transposition by means of matrix expressions. It does not perform the actual transposition but returns a temporary matrix transposition object that can be further used as a part of more complex matrix expressions or can be assigned to a matrix:Mat A1 = A + Mat::eye(A.size(), A.type())*lambda;Mat C = A1.t()*A1; // compute (A + lambda*I)^t * (A + lambda*I)fragmentExamples: samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, and samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp.	1									
core_Mat_inv	CVAPI(ExceptionStatus) core_Mat_inv(cv::Mat *self, int method, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->inv(method); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::Mat::inv()	classcv_1_1Mat.html	Inverses a matrix.	method : Matrix inversion method. One of cv::DecompTypes	The method performs a matrix inversion by means of matrix expressions. This means that a temporary matrix inversion object is returned by the method and can be used further as a part of more complex matrix expressions or can be assigned to a matrix.Examples: samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, and samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp.	1									
core_Mat_mul	CVAPI(ExceptionStatus) core_Mat_mul(cv::Mat *self, cv::_InputArray *m, double scale, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->mul(*m, scale); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::Mat::mul()	classcv_1_1Mat.html	Performs an element-wise multiplication or division of the two matrices.	m : Another array of the same type and the same size as *this, or a matrix expression. scale : Optional scale factor.	The method returns a temporary object encoding per-element array multiplication, with optional scale. Note that this is not a matrix multiplication that corresponds to a simpler "*" operator.Example:Mat C = A.mul(5/B); // equivalent to divide(A, B, C, 5)fragment	2									
core_Mat_cross	CVAPI(ExceptionStatus) core_Mat_cross(cv::Mat *self, cv::_InputArray *m, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->cross(*m); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::cross()	classcv_1_1Mat.html	Computes a cross-product of two 3-element vectors.	m : Another cross-product operand.	The method computes a cross-product of two 3-element vectors. The vectors must be 3-element floating-point vectors of the same shape and size. The result is another 3-element vector of the same shape and type as operands.Examples: samples/cpp/tutorial_code/features2D/Homography/pose_from_homography.cpp.	1									
core_Mat_dot	CVAPI(ExceptionStatus) core_Mat_dot(cv::Mat *self, cv::_InputArray *m, double *returnValue) { BEGIN_WRAP *returnValue = self->dot(*m); END_WRAP }	cv::Mat::dot()	classcv_1_1Mat.html	Computes a dot-product of two vectors.	m : another dot-product operand.	The method computes a dot-product of two matrices. If the matrices are not single-column or single-row vectors, the top-to-bottom left-to-right scan ordering is used to treat them as 1D vectors. The vectors must have the same size and type. If the matrices have more than one channel, the dot products from all the channels are summed together.Examples: samples/cpp/tutorial_code/features2D/Homography/decompose_homography.cpp, and samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp.	1									
core_Mat_zeros1	CVAPI(ExceptionStatus) core_Mat_zeros1(int rows, int cols, int type, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cv::Mat::zeros(rows, cols, type); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat::zeros()	classcv_1_1Mat.html	Returns a zero array of the specified size and type.	rows : Number of rows. cols : Number of columns. type : Created matrix type.	The method returns a Matlab-style zero array initializer. It can be used to quickly form a constant array as a function parameter, part of a matrix expression, or as a matrix initializer:Mat A;A = Mat::zeros(3, 3, CV_32F);fragmentIn the example above, a new matrix is allocated only if A is not a 3x3 floating-point matrix. Otherwise, the existing matrix A is filled with zeros.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/contours2.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/kalman.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, and	3									
core_Mat_zeros2	CVAPI(ExceptionStatus) core_Mat_zeros2(int ndims, const int *sz, int type, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cv::Mat::zeros(ndims, sz, type); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat::zeros()	classcv_1_1Mat.html	Returns a zero array of the specified size and type.	rows : Number of rows. cols : Number of columns. type : Created matrix type.	The method returns a Matlab-style zero array initializer. It can be used to quickly form a constant array as a function parameter, part of a matrix expression, or as a matrix initializer:Mat A;A = Mat::zeros(3, 3, CV_32F);fragmentIn the example above, a new matrix is allocated only if A is not a 3x3 floating-point matrix. Otherwise, the existing matrix A is filled with zeros.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/contours2.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/kalman.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, and	3									
core_Mat_ones1	CVAPI(ExceptionStatus) core_Mat_ones1(int rows, int cols, int type, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = cv::Mat::ones(rows, cols, type); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::Mat::ones()	classcv_1_1Mat.html	Returns an array of all 1's of the specified size and type.	rows : Number of rows. cols : Number of columns. type : Created matrix type.	The method returns a Matlab-style 1's array initializer, similarly to Mat::zeros. Note that using this method you can initialize an array with an arbitrary value, using the following Matlab idiom:Mat A = Mat::ones(100, 100, CV_8U)*3; // make 100x100 matrix filled with 3.fragmentThe above operation does not form a 100x100 matrix of 1's and then multiply it by 3. Instead, it just remembers the scale factor (3 in this case) and use it when actually invoking the matrix initializer.NoteIn case of multi-channels type, only the first channel will be initialized with 1's, the others will be set to 0's.Examples:	3									
core_Mat_ones2	CVAPI(ExceptionStatus) core_Mat_ones2(int ndims, const int *sz, int type, cv::MatExpr **returnValue) { BEGIN_WRAP auto ret = cv::Mat::ones(ndims, sz, type); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::Mat::ones()	classcv_1_1Mat.html	Returns an array of all 1's of the specified size and type.	rows : Number of rows. cols : Number of columns. type : Created matrix type.	The method returns a Matlab-style 1's array initializer, similarly to Mat::zeros. Note that using this method you can initialize an array with an arbitrary value, using the following Matlab idiom:Mat A = Mat::ones(100, 100, CV_8U)*3; // make 100x100 matrix filled with 3.fragmentThe above operation does not form a 100x100 matrix of 1's and then multiply it by 3. Instead, it just remembers the scale factor (3 in this case) and use it when actually invoking the matrix initializer.NoteIn case of multi-channels type, only the first channel will be initialized with 1's, the others will be set to 0's.Examples:	3									
core_Mat_eye	CVAPI(ExceptionStatus) core_Mat_eye(int rows, int cols, int type, cv::MatExpr **returnValue) { BEGIN_WRAP const auto eye = cv::Mat::eye(rows, cols, type); *returnValue = new cv::MatExpr(eye); END_WRAP }	cv::Mat::eye()	classcv_1_1Mat.html	Returns an identity matrix of the specified size and type.	rows : Number of rows. cols : Number of columns. type : Created matrix type.	The method returns a Matlab-style identity matrix initializer, similarly to Mat::zeros. Similarly to Mat::ones, you can use a scale operation to create a scaled identity matrix efficiently:// make a 4x4 diagonal matrix with 0.1's on the diagonal.Mat A = Mat::eye(4, 4, CV_32F)*0.1;fragmentNoteIn case of multi-channels type, identity matrix will be initialized only for the first channel, the others will be set to 0'sExamples: samples/cpp/image_alignment.cpp.	3									

core_Mat_create1	CVAPI(ExceptionStatus) core_Mat_create1(cv::Mat *self, int rows, int cols, int type) { BEGIN_WRAP self->create(rows, cols, type); END_WRAP }	cv::Mat::create()	classcv_1_1Mat.html	Allocates new array data if needed.	rows : New number of rows. cols : New number of columns. type : New matrix type.	This is one of the key Mat methods. Most new-style OpenCV functions and methods that produce arrays call this method for each output array. The method uses the following algorithm:If the current array shape and the type match the new ones, return immediately. Otherwise, de-reference the previous data by calling Mat::release. Initialize the new header. Allocate the new data of total()*elemSize() bytes. Allocate the new, associated with the data, reference counter and set it to 1.Such a scheme makes the memory management robust and efficient at the same time and helps avoid extra typing for you. This means that usually there is no need to explicitly allocate output arrays. That is, instead of writing:Mat color;...Mat gray(color.rows, color.cols, color.depth());cvtColor(color, gray, COLOR_BGR2GRAY);fragmentyou can simply write:Mat color;...Mat gray;cvtColor(color, gray, COLOR_BGR2GRAY);fragmentbecause	3								
core_Mat_create2	CVAPI(ExceptionStatus) core_Mat_create2(cv::Mat *self, int ndims, const int *sizes, int type) { BEGIN_WRAP self->create(ndims, sizes, type); END_WRAP }	cv::Mat::create()	classcv_1_1Mat.html	Allocates new array data if needed.	rows : New number of rows. cols : New number of columns. type : New matrix type.	This is one of the key Mat methods. Most new-style OpenCV functions and methods that produce arrays call this method for each output array. The method uses the following algorithm:If the current array shape and the type match the new ones, return immediately. Otherwise, de-reference the previous data by calling Mat::release. Initialize the new header. Allocate the new data of total()*elemSize() bytes. Allocate the new, associated with the data, reference counter and set it to 1.Such a scheme makes the memory management robust and efficient at the same time and helps avoid extra typing for you. This means that usually there is no need to explicitly allocate output arrays. That is, instead of writing:Mat color;...Mat gray(color.rows, color.cols, color.depth());cvtColor(color, gray, COLOR_BGR2GRAY);fragmentyou can simply write:Mat color;...Mat gray;cvtColor(color, gray, COLOR_BGR2GRAY);fragmentbecause	3								
core_Mat_reserve	CVAPI(ExceptionStatus) core_Mat_reserve(cv::Mat *self, size_t sz) { BEGIN_WRAP self->reserve(sz); END_WRAP }	cv::Mat::reserve()	classcv_1_1Mat.html	Reserves space for the certain number of rows.	sz : Number of rows.	The method reserves space for sz rows. If the matrix already has enough space to store sz rows, nothing happens. If the matrix is reallocated, the first Mat::rows rows are preserved. The method emulates the corresponding method of the STL vector class.	1								
core_Mat_reserveBuffer	CVAPI(ExceptionStatus) core_Mat_reserveBuffer(cv::Mat *self, size_t sz) { BEGIN_WRAP self->reserveBuffer(sz); END_WRAP }	cv::Mat::reserveBuffer()	classcv_1_1Mat.html	Reserves space for the certain number of bytes.	sz : Number of bytes.	The method reserves space for sz bytes. If the matrix already has enough space to store sz bytes, nothing happens. If matrix has to be reallocated its previous content could be lost.	1								
core_Mat_resize1	CVAPI(ExceptionStatus) core_Mat_resize1(cv::Mat *obj, size_t sz) { BEGIN_WRAP obj->resize(sz); END_WRAP }	cv::Mat::resize()	classcv_1_1Mat.html	Changes the number of matrix rows.	sz : New number of rows.	The methods change the number of matrix rows. If the matrix is reallocated, the first min(Mat::rows, sz) rows are preserved. The methods emulate the corresponding methods of the STL vector class.	1								
core_Mat_resize2	CVAPI(ExceptionStatus) core_Mat_resize2(cv::Mat *obj, size_t sz, MyCvScalar s) { BEGIN_WRAP obj->resize(sz, cpp(s)); END_WRAP }	cv::Mat::resize()	classcv_1_1Mat.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	sz : New number of rows. s : Value assigned to the newly added elements.		2								
core_Mat_pop_back	CVAPI(ExceptionStatus) core_Mat_pop_back(cv::Mat *obj, size_t nelems) { BEGIN_WRAP obj->pop_back(nelems); END_WRAP }	cv::Mat::pop_back()	classcv_1_1Mat.html	Removes elements from the bottom of the matrix.	nelems : Number of removed rows. If it is greater than the total number of rows, an exception is thrown.	The method removes one or more rows from the bottom of the matrix.	1								
core_Mat_locateROI	CVAPI(ExceptionStatus) core_Mat_locateROI(cv::Mat *self, MyCvSize *wholeSize, MyCvPoint *ofs) { BEGIN_WRAP cv::Size wholeSize2; cv::Point ofs2; self->locateROI(wholeSize2, ofs2); *wholeSize = c(cv::Size(wholeSize2.width, wholeSize2.height)); *ofs = c(cv::Point(ofs2.x, ofs2.y)); END_WRAP }	cv::Mat::locateROI()	classcv_1_1Mat.html	Locates the matrix header within a parent matrix.	wholeSize : Output parameter that contains the size of the whole matrix containing this as a part. ofs : Output parameter that contains an offset of this inside the whole matrix.	After you extracted a submatrix from a matrix using Mat::row, Mat::col, Mat::rowRange, Mat::colRange, and others, the resultant submatrix points just to the part of the original big matrix. However, each submatrix contains information (represented by datastart and dataend fields) that helps reconstruct the original matrix size and the position of the extracted submatrix within the original matrix. The method locateROI does exactly that.	2								
core_Mat_adjustROI	CVAPI(ExceptionStatus) core_Mat_adjustROI(cv::Mat *self, int dtop, int dbottom, int dleft, int dright, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = self->adjustROI(dtop, dbottom, dleft, dright); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat::adjustROI()	classcv_1_1Mat.html	Adjusts a submatrix size and position within the parent matrix.	dtop : Shift of the top submatrix boundary upwards. dbottom : Shift of the bottom submatrix boundary downwards. dleft : Shift of the left submatrix boundary to the left. dright : Shift of the right submatrix boundary to the right.	The method is complimentary to Mat::locateROI . The typical use of these functions is to determine the submatrix position within the parent matrix and then shift the position somehow. Typically, it can be required for filtering operations when pixels outside of the ROI should be taken into account. When all the method parameters are positive, the ROI needs to grow in all directions by the specified amount, for example:A.adjustROI(2, 2, 2, 2);fragmentIn this example, the matrix size is increased by 4 elements in each direction. The matrix is shifted by 2 elements to the left and 2 elements up, which brings in all the necessary pixels for the filtering with the 5x5 kernel.adjustROI forces the adjusted ROI to be inside of the parent matrix that is boundaries of the adjusted ROI are constrained by boundaries of the parent matrix. For example, if the submatrix A is located in the first row of a parent matrix and you called A.adjustROI(2, 2, 2, 2) then A will not be increased in the upward direction.The function is used internally by the OpenCV filtering functions, like	4								
core_Mat_subMat1	CVAPI(ExceptionStatus) core_Mat_subMat1(cv::Mat *self, int rowStart, int rowEnd, int colStart, int colEnd, cv::Mat** returnValue) { BEGIN_WRAP const cv::Range rowRange(rowStart, rowEnd); const cv::Range colRange(colStart, colEnd); const auto ret = (*self)(rowRange, colRange); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size.			•							
core_Mat_subMat2	CVAPI(ExceptionStatus) core_Mat_subMat2(cv::Mat *self, int nRanges, MyCvSlice *ranges, cv::Mat** returnValue) { BEGIN_WRAP std::vector<cv::Range> rangesVec(nRanges); for (auto i = 0; i < nRanges; i++) { rangesVec[i] = (cpp(ranges[i])); } const auto ret = (*self)(rangesVec[0]); *returnValue = new cv::Mat(ret); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size.			•							

core_Mat_isContinuous	CVAPI(ExceptionStatus) core_Mat_isContinuous(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->isContinuous() ? 1 : 0; END_WRAP }	cv::Mat::isContinuous()	classcv_1_1Mat.html	Reports whether the matrix is continuous or not.		The method returns true if the matrix elements are stored continuously without gaps at the end of each row. Otherwise, it returns false. Obviously, 1x1 or 1xN matrices are always continuous. Matrices created with Mat::create are always continuous. But if you extract a part of the matrix using Mat::col, Mat::diag, and so on, or constructed a matrix header for externally allocated data, such matrices may no longer have this property.The continuity flag is stored as a bit in the Mat::flags field and is computed automatically when you construct a matrix header. Thus, the continuity check is a very fast operation, though theoretically it could be done as follows:// alternative implementation of Mat::isContinuous()bool myCheckMatContinuity(const Mat& m){ //return (m.flags & Mat::CONTINUOUS_FLAG) != 0; return m.rows == 1 || m.step == m.cols*m.elemSize();}fragmentThe method is used in quite a few of OpenCV functions. The point is that element-wise	1								
core_Mat_isSubmatrix	CVAPI(ExceptionStatus) core_Mat_isSubmatrix(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->isSubmatrix() ? 1 : 0; END_WRAP }	cv::Mat::isSubmatrix()	classcv_1_1Mat.html	returns true if the matrix is a submatrix of another matrix			1								
core_Mat_elemSize	CVAPI(ExceptionStatus) core_Mat_elemSize(cv::Mat *self, size_t *returnValue) { BEGIN_WRAP *returnValue = self->elemSize(); END_WRAP }	cv::Mat::elemSize()	classcv_1_1Mat.html	Returns the matrix element size in bytes.		The method returns the matrix element size in bytes. For example, if the matrix type is CV_16SC3 , the method returns 3*sizeof(short) or 6.	1								
core_Mat_elemSize1	CVAPI(ExceptionStatus) core_Mat_elemSize1(cv::Mat *self, size_t *returnValue) { BEGIN_WRAP *returnValue = self->elemSize1(); END_WRAP }	cv::Mat::elemSize1()	classcv_1_1Mat.html	Returns the size of each matrix element channel in bytes.		The method returns the matrix element channel size in bytes, that is, it ignores the number of channels. For example, if the matrix type is CV_16SC3 , the method returns sizeof(short) or 2.	1								
core_Mat_type	CVAPI(ExceptionStatus) core_Mat_type(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->type(); END_WRAP }	cv::Mat::type()	classcv_1_1Mat.html	Returns the type of a matrix element.		The method returns a matrix element type. This is an identifier compatible with the CvMat type system, like CV_16SC3 or 16-bit signed 3-channel array, and so on.Examples: samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/train_HOG.cpp, and samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp.	1								
core_Mat_depth	CVAPI(ExceptionStatus) core_Mat_depth(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->depth(); END_WRAP }	cv::Mat::depth()	classcv_1_1Mat.html	Returns the depth of a matrix element.		The method returns the identifier of the matrix element depth (the type of each individual channel). For example, for a 16-bit signed element array, the method returns CV_16S . A complete list of matrix types contains the following values:CV_8U - 8-bit unsigned integers (0..255) CV_8S - 8-bit signed integers (-128..127) CV_16U - 16-bit unsigned integers (0..65535) CV_16S - 16-bit signed integers (-32768..32767) CV_32S - 32-bit signed integers (-2147483648..2147483647) CV_32F - 32-bit floating-point numbers (-FLT_MAX..FLT_MAX, INF, NAN) CV_64F - 64-bit floating-point numbers (-DBL_MAX..DBL_MAX, INF, NAN)Examples:	1								
core_Mat_channels	CVAPI(ExceptionStatus) core_Mat_channels(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->channels(); END_WRAP }	cv::Mat::channels()	classcv_1_1Mat.html	Returns the number of matrix channels.		The method returns the number of matrix channels.Examples: samples/cpp/pca.cpp.	1								
core_Mat_empty	CVAPI(ExceptionStatus) core_Mat_empty(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->empty() ? 1 : 0; END_WRAP }	cv::Mat::empty()	classcv_1_1Mat.html	Returns true if the array has no elements.		The method returns true if Mat::total() is 0 or if Mat::data is NULL. Because of pop_back() and resize() methods M.total() == 0 does not imply that M.data == NULL.Examples: fld_lines.cpp, samples/cpp/facedetect.cpp, samples/cpp/fitellipse.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/lkdemo.cpp, samples/cpp/squares.cpp, samples/cpp/stitching.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/HighGUI/AddingImagesTrackbar.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_2.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/ImgTrans/copy	1								
core_Mat_total1	CVAPI(ExceptionStatus) core_Mat_total1(cv::Mat *self, size_t *returnValue) { BEGIN_WRAP *returnValue = self->total(); END_WRAP }	cv::Mat::total()	classcv_1_1Mat.html	Returns the total number of array elements.		The method returns the number of array elements (a number of pixels if the array represents an image).Examples: samples/cpp/train_HOG.cpp.	1								
core_Mat_total2	CVAPI(ExceptionStatus) core_Mat_total2(cv::Mat *self, int startDim, int endDim, size_t *returnValue) { BEGIN_WRAP *returnValue = self->total(startDim, endDim); END_WRAP }	cv::Mat::total()	classcv_1_1Mat.html	Returns the total number of array elements.		The method returns the number of array elements (a number of pixels if the array represents an image).Examples: samples/cpp/train_HOG.cpp.	2								
core_Mat_checkVector	CVAPI(ExceptionStatus) core_Mat_checkVector(cv::Mat *self, int elemChannels, int depth, int requireContinuous, int *returnValue) { BEGIN_WRAP *returnValue = self->checkVector(elemChannels, depth, requireContinuous != 0); END_WRAP }	cv::Mat::checkVector()	classcv_1_1Mat.html	The following code demonstrates its usage for a 2-d matrix:	elemChannels : Number of channels or number of columns the matrix should have. For a 2-D matrix, when the matrix has only 1 column, then it should have elemChannels channels; When the matrix has only 1 channel, then it should have elemChannels columns. For a 3-D matrix, it should have only one channel. Furthermore, if the number of planes is not one, then the number of rows within every plane has to be 1; if the number of rows within every plane is not 1, then the number of planes has to be 1. depth : The depth the matrix should have. Set it to -1 when any depth is fine. requireContinuous : Set it to true to require the matrix to be continuous	cv::Mat mat(2, 1, CV_32FC2); int n = mat.checkVector(2); CV_Assert(n == 20); // mat has 20 elements mat.create(20, 2, CV_32FC1); n = mat.checkVector(1); CV_Assert(n == -1); // mat is neither a column nor a row vector n = mat.checkVector(2); CV_Assert(n == 20); // the 2 columns are considered as 1 elementfragmentThe following code demonstrates its usage for a 3-d matrix:int dims[] = {1, 3, 5}; // 1 plane, every plane has 3 rows and 5 columns mat.create(3, dims, CV_32FC1); // for 3-d mat, it MUST have only 1 channel n = mat.checkVector(5); // the 5 columns are considered as 1 element CV_Assert(n == 3); int dims2[] = {3, 1, 5}; // 3 planes, every plane has 1 row and 5 columns mat.create(3, dims2, CV_32FC1); n = mat.checkVector(5); // the 5 columns are considered as 1 element CV_Assert(n	3								
core_Mat_ptr1d	CVAPI(ExceptionStatus) core_Mat_ptr1d(cv::Mat *self, int i0, uchar **returnValue) { BEGIN_WRAP *returnValue = self->ptr(i0); END_WRAP }	cv::Mat::ptr()	classcv_1_1Mat.html	Returns a pointer to the specified matrix row.	i0 : A 0-based row index.	The methods return uchar* or typed pointer to the specified matrix row. See the sample in Mat::isContinuous to know how to use these methods.Examples: samples/cpp/image_alignment.cpp, samples/cpp/train_HOG.cpp, samples/dnn/colorization.cpp, samples/dnn/openpose.cpp, and samples/dnn/segmentation.cpp.	1								
core_Mat_ptr2d	CVAPI(ExceptionStatus) core_Mat_ptr2d(cv::Mat *self, int i0, int i1, uchar **returnValue) { BEGIN_WRAP *returnValue = self->ptr(i0, i1); END_WRAP }	cv::Mat::ptr()	classcv_1_1Mat.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	row : Index along the dimension 0 col : Index along the dimension 1		2								
core_Mat_ptr3d	CVAPI(ExceptionStatus) core_Mat_ptr3d(cv::Mat *self, int i0, int i1, int i2, uchar **returnValue) { BEGIN_WRAP *returnValue = self->ptr(i0, i1, i2); END_WRAP }	cv::Mat::ptr()	classcv_1_1Mat.html	Returns a pointer to the specified matrix row.	i0 : A 0-based row index.	The methods return uchar* or typed pointer to the specified matrix row. See the sample in Mat::isContinuous to know how to use these methods.Examples: samples/cpp/image_alignment.cpp, samples/cpp/train_HOG.cpp, samples/dnn/colorization.cpp, samples/dnn/openpose.cpp, and samples/dnn/segmentation.cpp.	3								
core_Mat_ptrnd	CVAPI(ExceptionStatus) core_Mat_ptrnd(cv::Mat *self, int *idx, uchar **returnValue) { BEGIN_WRAP *returnValue = self->ptr(idx); END_WRAP }	cv::Mat::ptr()	classcv_1_1Mat.html	Returns a pointer to the specified matrix row.	i0 : A 0-based row index.	The methods return uchar* or typed pointer to the specified matrix row. See the sample in Mat::isContinuous to know how to use these methods.Examples: samples/cpp/image_alignment.cpp, samples/cpp/train_HOG.cpp, samples/dnn/colorization.cpp, samples/dnn/openpose.cpp, and samples/dnn/segmentation.cpp.	1								

core_Mat_flags	CVAPI(ExceptionStatus) core_Mat_flags(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->flags; END_WRAP }	cv::Mat::flags	classcv_1_1Mat.html	includes several bit-fields: 		the magic signature continuity flag depth number of channels				•						
core_Mat_dims	CVAPI(ExceptionStatus) core_Mat_dims(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->dims; END_WRAP }	cv::Mat::dims	classcv_1_1Mat.html	the matrix dimensionality, >= 2						•						
core_Mat_rows	CVAPI(ExceptionStatus) core_Mat_rows(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->rows; END_WRAP }	cv::Mat::rows	classcv_1_1Mat.html	the number of rows and columns or (-1, -1) when the matrix has more than 2 dimensions		Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/convexhull.cpp, samples/cpp/demhist.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/kalman.cpp, samples/cpp/kmeans.cpp, samples/cpp/minarea.cpp, samples/cpp/squares.cpp, samples/cpp/stitching.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/ImgTrans/copyMakeBorder_demo.cpp, samples/cpp/tutorial_code/ImgTrans/houghcircles.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/tutorial_code/ml/introduction_to_svm/introduction_to_svm.cpp				•						
core_Mat_cols	CVAPI(ExceptionStatus) core_Mat_cols(cv::Mat *self, int *returnValue) { BEGIN_WRAP *returnValue = self->cols; END_WRAP }	cv::Mat::cols	classcv_1_1Mat.html	Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/convexhull.cpp, samples/cpp/demhist.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/kalman.cpp, samples/cpp/kmeans.cpp, samples/cpp/minarea.cpp, samples/cpp/squares.cpp, samples/cpp/stitching.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/ImgTrans/copyMakeBorder_demo.cpp, samples/cpp/warpPerspective_demo.cpp, samples/cpp/watershed.cpp, samples/dnn/object_detection.cpp, and						•					
core_Mat_data	CVAPI(ExceptionStatus) core_Mat_data(cv::Mat *self, uchar **returnValue) { BEGIN_WRAP *returnValue = self->data; END_WRAP }	cv::Mat::data	classcv_1_1Mat.html	pointer to the data		Examples: samples/dnn/segmentation.cpp.				•						
core_Mat_datastart	CVAPI(ExceptionStatus) core_Mat_datastart(cv::Mat *self, const uchar **returnValue) { BEGIN_WRAP *returnValue = self->datastart; END_WRAP }	cv::Mat::datastart	classcv_1_1Mat.html	helper fields used in locateROI and adjustROI						•						
core_Mat_dataend	CVAPI(ExceptionStatus) core_Mat_dataend(cv::Mat *self, const uchar **returnValue) { BEGIN_WRAP *returnValue = self->dataend; END_WRAP }	cv::Mat::dataend	classcv_1_1Mat.html							•						
core_Mat_datalimit	CVAPI(ExceptionStatus) core_Mat_datalimit(cv::Mat *self, const uchar **returnValue) { BEGIN_WRAP *returnValue = self->datalimit; END_WRAP }	cv::Mat::datalimit	classcv_1_1Mat.html							•						
core_Mat_size	CVAPI(uint64) core_Mat_sizeof() { return sizeof(cv::Mat); }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i,j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•						
core_Mat_sizeAt	CVAPI(ExceptionStatus) core_Mat_sizeAt(cv::Mat *self, int i, int *returnValue) { BEGIN_WRAP *returnValue = self->size[i]; END_WRAP }	cv::Mat::size	classcv_1_1Mat.html	Examples: fld_lines.cpp, samples/cpp/camshiftdemo.cpp, samples/cpp/connected_components.cpp, samples/cpp/create_mask.cpp, samples/cpp/grabcut.cpp, samples/cpp/image_alignment.cpp, samples/cpp/polar_transforms.cpp, samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/tutorial_code/features2D/Homography/homography_from_camera_displacement.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp, samples/cpp/watershed.cpp, samples/dnn/colorization.cpp, samples/dnn/openpose.cpp, and samples/dnn/segmentation.cpp.						•					
core_Mat_step1	CVAPI(ExceptionStatus) core_Mat_step1(cv::Mat *self, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = self->step1(i); END_WRAP }	cv::Mat::step1()	classcv_1_1Mat.html	Returns a normalized step.		The method returns a matrix step divided by Mat::elemSize1() . It can be useful to quickly access an arbitrary matrix element.	1									
core_Mat_step	CVAPI(ExceptionStatus) core_Mat_step1(cv::Mat *self, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = self->step1(i); END_WRAP }	cv::Mat::step1()	classcv_1_1Mat.html	Returns a normalized step.		The method returns a matrix step divided by Mat::elemSize1() . It can be useful to quickly access an arbitrary matrix element.	1									
core_Mat_stepAt	CVAPI(ExceptionStatus) core_Mat_stepAt(cv::Mat *self, int i, size_t *returnValue) { BEGIN_WRAP *returnValue = self->step[i]; END_WRAP }	cv::Mat::step	classcv_1_1Mat.html							•						

core_abs_Mat	CVAPI(ExceptionStatus) core_abs_Mat(cv::Mat *m, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = cv::abs(*m); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						
core_Mat_setMatData	CVAPI(ExceptionStatus) core_Mat_setMatData(cv::Ma t *obj, uchar *vals, int *returnValue) { BEGIN_WRAP *returnValue = internal_Mat_set(obj, vals) ? 1 : 0; END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						
core_Mat_getMatData	CVAPI(ExceptionStatus) core_Mat_getMatData(cv::Ma t *obj, uchar *vals, int *returnValue) { BEGIN_WRAP *returnValue = internal_Mat_get(obj, vals) ? 1 : 0; END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						
core_Mat_push_back_Mat	CVAPI(ExceptionStatus) core_Mat_push_back_Mat(cv: :Mat *self, cv::Mat *m) { BEGIN_WRAP self->push_back(*m); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_char	CVAPI(ExceptionStatus) core_Mat_push_back_char(cv: :Mat *self, char v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_uchar	CVAPI(ExceptionStatus) core_Mat_push_back_uchar(c v::Mat *self, uchar v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_short	CVAPI(ExceptionStatus) core_Mat_push_back_short(cv ::Mat *self, short v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_ushort	CVAPI(ExceptionStatus) core_Mat_push_back_ushort(c v::Mat *self, ushort v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_int	CVAPI(ExceptionStatus) core_Mat_push_back_int(cv:: Mat *self, int v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_float	CVAPI(ExceptionStatus) core_Mat_push_back_float(cv ::Mat *self, float v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_double	CVAPI(ExceptionStatus) core_Mat_push_back_double(cv::Mat *self, double v) { BEGIN_WRAP self->push_back(v); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2b	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2b(c v::Mat *self, CvVec2b v) { BEGIN_WRAP self- >push_back(cv::Vec2b(v.val[0], END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec3b	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3b(c v::Mat *self, CvVec3b v) { BEGIN_WRAP self- >push_back(cv::Vec3b(v.val[0], END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4b	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4b(c v::Mat *self, CvVec4b v) { BEGIN_WRAP self- >push_back(cv::Vec4b(v.val[0], END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6b	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6b(c v::Mat *self, CvVec6b v) { BEGIN_WRAP self- >push_back(cv::Vec6b(v.val[0], END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2s	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2s(c v::Mat *self, CvVec2s v) { BEGIN_WRAP self- >push_back(cv::Vec2s(v.val[0], END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									

core_Mat_push_back_Vec3s	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3s(cv::Mat *self, CvVec3s v) { BEGIN_WRAP self->push_back(cv::Vec3s(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4s	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4s(cv::Mat *self, CvVec4s v) { BEGIN_WRAP self->push_back(cv::Vec4s(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6s	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6s(cv::Mat *self, CvVec6s v) { BEGIN_WRAP self->push_back(cv::Vec6s(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2w	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2w(cv::Mat *self, CvVec2w v) { BEGIN_WRAP self->push_back(cv::Vec2w(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec3w	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3w(cv::Mat *self, CvVec3w v) { BEGIN_WRAP self->push_back(cv::Vec3w(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4w	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4w(cv::Mat *self, CvVec4w v) { BEGIN_WRAP self->push_back(cv::Vec4w(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6w	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6w(cv::Mat *self, CvVec6w v) { BEGIN_WRAP self->push_back(cv::Vec6w(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2i	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2i(cv::Mat *self, CvVec2i v) { BEGIN_WRAP self->push_back(cv::Vec2i(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec3i	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3i(cv::Mat *self, CvVec3i v) { BEGIN_WRAP self->push_back(cv::Vec3i(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4i	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4i(cv::Mat *self, CvVec4i v) { BEGIN_WRAP self->push_back(cv::Vec4i(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6i	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6i(cv::Mat *self, CvVec6i v) { BEGIN_WRAP self->push_back(cv::Vec6i(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2f	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2f(cv::Mat *self, CvVec2f v) { BEGIN_WRAP self->push_back(cv::Vec2f(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec3f	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3f(cv::Mat *self, CvVec3f v) { BEGIN_WRAP self->push_back(cv::Vec3f(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4f	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4f(cv::Mat *self, CvVec4f v) { BEGIN_WRAP self->push_back(cv::Vec4f(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6f	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6f(cv::Mat *self, CvVec6f v) { BEGIN_WRAP self->push_back(cv::Vec6f(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec2d	CVAPI(ExceptionStatus) core_Mat_push_back_Vec2d(cv::Mat *self, CvVec2d v) { BEGIN_WRAP self->push_back(cv::Vec2d(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec3d	CVAPI(ExceptionStatus) core_Mat_push_back_Vec3d(cv::Mat *self, CvVec3d v) { BEGIN_WRAP self->push_back(cv::Vec3d(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec4d	CVAPI(ExceptionStatus) core_Mat_push_back_Vec4d(cv::Mat *self, CvVec4d v) { BEGIN_WRAP self->push_back(cv::Vec4d(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									
core_Mat_push_back_Vec6d	CVAPI(ExceptionStatus) core_Mat_push_back_Vec6d(cv::Mat *self, CvVec6d v) { BEGIN_WRAP self->push_back(cv::Vec6d(v.val)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1									

core_Mat_push_back_Point	CVAPI(ExceptionStatus) core_Mat_push_back_Point(cv::Mat *self, MyCvPoint v) { BEGIN_WRAP self->push_back(cv::Point(v.x, v.y)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Point2f	CVAPI(ExceptionStatus) core_Mat_push_back_Point2f(cv::Mat *self, MyCvPoint2D32f v) { BEGIN_WRAP self->push_back(cv::Point2f(v.x, v.y)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Point2d	CVAPI(ExceptionStatus) core_Mat_push_back_Point2d(cv::Mat *self, MyCvPoint2D64f v) { BEGIN_WRAP self->push_back(cv::Point2d(v.x, v.y)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Point3i	CVAPI(ExceptionStatus) core_Mat_push_back_Point3i(cv::Mat *self, MyCvPoint3D32i v) { BEGIN_WRAP self->push_back(cv::Point3i(v.x, v.y, v.z)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Point3f	CVAPI(ExceptionStatus) core_Mat_push_back_Point3f(cv::Mat *self, MyCvPoint3D32f v) { BEGIN_WRAP self->push_back(cv::Point3f(v.x, v.y, v.z)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Point3d	CVAPI(ExceptionStatus) core_Mat_push_back_Point3d(cv::Mat *self, MyCvPoint3D64f v) { BEGIN_WRAP self->push_back(cv::Point3d(v.x, v.y, v.z)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Size	CVAPI(ExceptionStatus) core_Mat_push_back_Size(cv::Mat *self, MyCvSize v) { BEGIN_WRAP self->push_back(cv::Size(v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Size2f	CVAPI(ExceptionStatus) core_Mat_push_back_Size2f(cv::Mat *self, MyCvSize2D32f v) { BEGIN_WRAP self->push_back(cv::Size2f(v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Size2d	CVAPI(ExceptionStatus) core_Mat_push_back_Size2d(cv::Mat *self, MyCvSize2D64f v) { BEGIN_WRAP self->push_back(cv::Size2d(v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Rect	CVAPI(ExceptionStatus) core_Mat_push_back_Rect(cv::Mat *self, MyCvRect v) { BEGIN_WRAP self->push_back(cv::Rect(v.x, v.y, v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Rect2f	CVAPI(ExceptionStatus) core_Mat_push_back_Rect2f(cv::Mat *self, MyCvRect2D32f v) { BEGIN_WRAP self->push_back(cv::Rect2f(v.x, v.y, v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_push_back_Rect2d	CVAPI(ExceptionStatus) core_Mat_push_back_Rect2d(cv::Mat *self, MyCvRect2D64f v) { BEGIN_WRAP self->push_back(cv::Rect2d(v.x, v.y, v.width, v.height)); END_WRAP }	cv::Mat::push_back()	classcv_1_1Mat.html	Adds elements to the bottom of the matrix.	elem : Added element(s).	The methods add one or more elements to the bottom of the matrix. They emulate the corresponding method of the STL vector class. When elem is Mat , its type and the number of columns must be the same as in the container matrix.	1										
core_Mat_forEach_uchar	CVAPI(ExceptionStatus) core_Mat_forEach_uchar(cv::Mat *m, Mat_foreach_uchar proc) { BEGIN_WRAP const Functor<Mat_foreach_uchar, uchar> functor(proc); m->forEach<uchar>(functor); END_WRAP }	cv::Mat::forEach<uchar>()																
core_Mat_forEach_Vec2b	CVAPI(ExceptionStatus) core_Mat_forEach_Vec2b(cv::Mat *m, Mat_foreach_Vec2b proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec2b, cv::Vec2b> functor(proc); m->forEach<cv::Vec2b>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec2b>()																
core_Mat_forEach_Vec3b	CVAPI(ExceptionStatus) core_Mat_forEach_Vec3b(cv::Mat *m, Mat_foreach_Vec3b proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec3b, cv::Vec3b> functor(proc); m->forEach<cv::Vec3b>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec3b>()																
core_Mat_forEach_Vec4b	CVAPI(ExceptionStatus) core_Mat_forEach_Vec4b(cv::Mat *m, Mat_foreach_Vec4b proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec4b, cv::Vec4b> functor(proc); m->forEach<cv::Vec4b>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec4b>()																
core_Mat_forEach_Vec6b	CVAPI(ExceptionStatus) core_Mat_forEach_Vec6b(cv::Mat *m, Mat_foreach_Vec6b proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec6b, cv::Vec6b> functor(proc); m->forEach<cv::Vec6b>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec6b>()																

core_Mat_forEach_short	CVAPI(ExceptionStatus) core_Mat_forEach_short(cv::Mat *m, Mat_foreach_short proc) { BEGIN_WRAP const Functor<Mat_foreach_short, short> functor(proc); m->forEach<short>(functor); END_WRAP }	cv::Mat::forEach<short>()														
core_Mat_forEach_Vec2s	CVAPI(ExceptionStatus) core_Mat_forEach_Vec2s(cv::Mat *m, Mat_foreach_Vec2s proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec2s, cv::Vec2s> functor(proc); m->forEach<cv::Vec2s>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec2s>()														
core_Mat_forEach_Vec3s	CVAPI(ExceptionStatus) core_Mat_forEach_Vec3s(cv::Mat *m, Mat_foreach_Vec3s proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec3s, cv::Vec3s> functor(proc); m->forEach<cv::Vec3s>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec3s>()														
core_Mat_forEach_Vec4s	CVAPI(ExceptionStatus) core_Mat_forEach_Vec4s(cv::Mat *m, Mat_foreach_Vec4s proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec4s, cv::Vec4s> functor(proc); m->forEach<cv::Vec4s>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec4s>()														
core_Mat_forEach_Vec6s	CVAPI(ExceptionStatus) core_Mat_forEach_Vec6s(cv::Mat *m, Mat_foreach_Vec6s proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec6s, cv::Vec6s> functor(proc); m->forEach<cv::Vec6s>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec6s>()														
core_Mat_forEach_int	CVAPI(ExceptionStatus) core_Mat_forEach_int(cv::Mat *m, Mat_foreach_int proc) { BEGIN_WRAP const Functor<Mat_foreach_int, int> functor(proc); m->forEach<int>(functor); END_WRAP }	cv::Mat::forEach<int>()														
core_Mat_forEach_Vec2i	CVAPI(ExceptionStatus) core_Mat_forEach_Vec2i(cv::Mat *m, Mat_foreach_Vec2i proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec2i, cv::Vec2i> functor(proc); m->forEach<cv::Vec2i>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec2i>()														
core_Mat_forEach_Vec3i	CVAPI(ExceptionStatus) core_Mat_forEach_Vec3i(cv::Mat *m, Mat_foreach_Vec3i proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec3i, cv::Vec3i> functor(proc); m->forEach<cv::Vec3i>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec3i>()														
core_Mat_forEach_Vec4i	CVAPI(ExceptionStatus) core_Mat_forEach_Vec4i(cv::Mat *m, Mat_foreach_Vec4i proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec4i, cv::Vec4i> functor(proc); m->forEach<cv::Vec4i>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec4i>()														
core_Mat_forEach_Vec6i	CVAPI(ExceptionStatus) core_Mat_forEach_Vec6i(cv::Mat *m, Mat_foreach_Vec6i proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec6i, cv::Vec6i> functor(proc); m->forEach<cv::Vec6i>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec6i>()														
core_Mat_forEach_float	CVAPI(ExceptionStatus) core_Mat_forEach_float(cv::Mat *m, Mat_foreach_float proc) { BEGIN_WRAP const Functor<Mat_foreach_float, float> functor(proc); m->forEach<float>(functor); END_WRAP }	cv::Mat::forEach<float>()														
core_Mat_forEach_Vec2f	CVAPI(ExceptionStatus) core_Mat_forEach_Vec2f(cv::Mat *m, Mat_foreach_Vec2f proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec2f, cv::Vec2f> functor(proc); m->forEach<cv::Vec2f>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec2f>()														
core_Mat_forEach_Vec3f	CVAPI(ExceptionStatus) core_Mat_forEach_Vec3f(cv::Mat *m, Mat_foreach_Vec3f proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec3f, cv::Vec3f> functor(proc); m->forEach<cv::Vec3f>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec3f>()														
core_Mat_forEach_Vec4f	CVAPI(ExceptionStatus) core_Mat_forEach_Vec4f(cv::Mat *m, Mat_foreach_Vec4f proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec4f, cv::Vec4f> functor(proc); m->forEach<cv::Vec4f>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec4f>()														

core_Mat_forEach_Vec6f	CVAPI(ExceptionStatus) core_Mat_forEach_Vec6f(cv::Mat *m, Mat_foreach_Vec6f proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec6f, cv::Vec6f> functor(proc); m->forEach<cv::Vec6f>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec6f>()														
core_Mat_forEach_double	CVAPI(ExceptionStatus) core_Mat_forEach_double(cv::Mat *m, Mat_foreach_double proc) { BEGIN_WRAP const Functor<Mat_foreach_double, double> functor(proc); m->forEach<double>(functor); END_WRAP }	cv::Mat::forEach<double>()														
core_Mat_forEach_Vec2d	CVAPI(ExceptionStatus) core_Mat_forEach_Vec2d(cv::Mat *m, Mat_foreach_Vec2d proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec2d, cv::Vec2d> functor(proc); m->forEach<cv::Vec2d>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec2d>()														
core_Mat_forEach_Vec3d	CVAPI(ExceptionStatus) core_Mat_forEach_Vec3d(cv::Mat *m, Mat_foreach_Vec3d proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec3d, cv::Vec3d> functor(proc); m->forEach<cv::Vec3d>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec3d>()														
core_Mat_forEach_Vec4d	CVAPI(ExceptionStatus) core_Mat_forEach_Vec4d(cv::Mat *m, Mat_foreach_Vec4d proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec4d, cv::Vec4d> functor(proc); m->forEach<cv::Vec4d>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec4d>()														
core_Mat_forEach_Vec6d	CVAPI(ExceptionStatus) core_Mat_forEach_Vec6d(cv::Mat *m, Mat_foreach_Vec6d proc) { BEGIN_WRAP const Functor<Mat_foreach_Vec6d, cv::Vec6d> functor(proc); m->forEach<cv::Vec6d>(functor); END_WRAP }	cv::Mat::forEach<cv::Vec6d>()														
core_Mat_operatorUnaryMinus	CVAPI(ExceptionStatus) core_Mat_operatorUnaryMinus(cv::Mat *mat, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = -(*mat); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i,j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size				•						
core_Mat_operatorAdd_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorAdd_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) + (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i,j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						
core_Mat_operatorAdd_MatScalar	CVAPI(ExceptionStatus) core_Mat_operatorAdd_MatScalar(cv::Mat *a, MyCvScalar s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) + cpp(s); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i,j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						
core_Mat_operatorAdd_ScalarMat	CVAPI(ExceptionStatus) core_Mat_operatorAdd_ScalarMat(MyCvScalar s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cpp(s) + (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i,j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is				•						

core_Mat_operatorMinus_Mat	CVAPI(ExceptionStatus) core_Mat_operatorMinus_Mat(cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = -(*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorSubtract_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorSubtract_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) - (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorSubtract_MatScalar	CVAPI(ExceptionStatus) core_Mat_operatorSubtract_MatScalar(cv::Mat *a, MyCvScalar s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) - cpp(s); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorSubtract_ScalarMat	CVAPI(ExceptionStatus) core_Mat_operatorSubtract_ScalarMat(MyCvScalar s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cpp(s) - (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorMultiply_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorMultiply_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) * (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorMultiply_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorMultiply_MatDouble(cv::Mat *a, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) * s; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorMultiply_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorMultiply_DoubleMat(double s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = s * (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							

core_Mat_operatorDivide_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorDivide_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) / (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorDivide_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorDivide_MatDouble(cv::Mat *a, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) / s; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorDivide_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorDivide_DoubleMat(double s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = s / (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorAnd_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorAnd_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) & (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorAnd_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorAnd_MatDouble(cv::Mat *a, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) & s; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorAnd_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorAnd_DoubleMat(double s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = s & (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorOr_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorOr_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a)	(*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							

core_Mat_operatorOr_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorOr_MatDouble(cv::Mat *a, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) | s; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorOr_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorOr_DoubleMat(double s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = s | (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorXor_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorXor_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) ^ (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorXor_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorXor_MatDouble(cv::Mat *a, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) ^ s; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorXor_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorXor_DoubleMat(double s, cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = s ^ (*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorNot	CVAPI(ExceptionStatus) core_Mat_operatorNot(cv::Mat *a, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = ~(*a); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorLT_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorLT_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) < (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							

core_Mat_operatorLT_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorLT_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a < (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorLT_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorLT_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) < b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorLE_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorLE_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) <= (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorLE_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorLE_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a <= (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorLE_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorLE_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) <= b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorGT_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorGT_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) > (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorGT_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorGT_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a > (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							

core_Mat_operatorGT_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorGT_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) > b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorGE_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorGE_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) >= (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorGE_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorGE_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a >= (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorGE_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorGE_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) >= b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorEQ_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorEQ_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) == (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorEQ_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorEQ_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a == (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							
core_Mat_operatorEQ_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorEQ_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) == b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i_0,...,i_{M.dims-1}))¥, where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_j}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is minimal and always equal to the element size					•							

core_Mat_operatorNE_MatMat	CVAPI(ExceptionStatus) core_Mat_operatorNE_MatMat(cv::Mat *a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) != (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i,0,...,j_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is					•							
core_Mat_operatorNE_DoubleMat	CVAPI(ExceptionStatus) core_Mat_operatorNE_DoubleMat(double a, cv::Mat *b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = a != (*b); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i,0,...,j_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is					•							
core_Mat_operatorNE_MatDouble	CVAPI(ExceptionStatus) core_Mat_operatorNE_MatDouble(cv::Mat *a, double b, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*a) != b; *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::Mat	classcv_1_1Mat.html	n-dimensional dense array class The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat). The data layout of the array M is defined by the array M.step[], so that the address of element ¥((i,0,...,j_{M.dims-1})¥), where ¥(0¥leq i_k<M.size[k]¥), is computed as: ¥[addr(M_{i_0,...,i_{M.dims-1}}) = M.data + M.step[0]*i_0 + M.step[1]*i_1 + ... + M.step[M.dims-1]*i_{M.dims-1}¥] In case of a 2-dimensional array, the above formula is reduced to: ¥[addr(M_{i_0}) = M.data + M.step[0]*i + M.step[1]*j¥] Note that M.step[i] >= M.step[i+1] (in fact, M.step[i] >= M.step[i+1]*M.size[i+1]). This means that 2-dimensional matrices are stored row-by-row, 3-dimensional matrices are stored plane-by-plane, and so on. M.step[M.dims-1] is					•							
core_MatExpr_new1	CVAPI(ExceptionStatus) core_MatExpr_new1(cv::MatExpr **returnValue) { BEGIN_WRAP *returnValue = new cv::MatExpr; END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),								•				
core_MatExpr_new2	CVAPI(ExceptionStatus) core_MatExpr_new2(cv::Mat *mat_,cv::MatExpr **returnValue) { BEGIN_WRAP *returnValue = new cv::MatExpr(*mat); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),								•				
core_MatExpr_delete	CVAPI(ExceptionStatus) core_MatExpr_delete(cv::MatExpr *self) { BEGIN_WRAP delete self; END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),									•			
core_MatExpr_toMat	CVAPI(ExceptionStatus) core_MatExpr_toMat(cv::MatExpr *self, cv::Mat *returnValue) { BEGIN_WRAP *returnValue = (*self); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),								•				
core_MatExpr_row	CVAPI(ExceptionStatus) core_MatExpr_row(cv::MatExpr *self, int y, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->row(y); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::row()	classcv_1_1MatExpr.html					1								

core_MatExpr_col	CVAPI(ExceptionStatus) core_MatExpr_col(cv::MatExpr *self, int x, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->col(x); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::col()	classcv_1_1MatExpr.html				1									
core_MatExpr_diag	CVAPI(ExceptionStatus) core_MatExpr_diag(cv::MatExpr *self, int d, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->diag(d); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::diag()	classcv_1_1MatExpr.html				1									
core_MatExpr_submat	CVAPI(ExceptionStatus) core_MatExpr_submat(cv::MatExpr *self, int rowStart, int rowEnd, int colStart, int colEnd, cv::MatExpr **returnValue) { BEGIN_WRAP const cv::Range rowRange(rowStart, rowEnd); const cv::Range colRange(colStart, colEnd); const auto ret = (*self)(rowRange, colRange); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				1		•						
core_MatExpr_t	CVAPI(ExceptionStatus) core_MatExpr_toMat(cv::MatExpr *self, cv::Mat *returnValue) { BEGIN_WRAP *returnValue = (*self); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),						•						
core_MatExpr_inv	CVAPI(ExceptionStatus) core_MatExpr_inv(cv::MatExpr *self, int method, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->inv(method); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::inv()	classcv_1_1MatExpr.html				1									
core_MatExpr_mul_toMatExpr	CVAPI(ExceptionStatus) core_MatExpr_mul_toMatExpr(cv::MatExpr *self, cv::MatExpr *e, double scale, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->mul(*e, scale); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::mul()	classcv_1_1MatExpr.html				2									
core_MatExpr_mul_toMat	CVAPI(ExceptionStatus) core_MatExpr_mul_toMatExpr(cv::MatExpr *self, cv::MatExpr *e, double scale, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = self->mul(*e, scale); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr::mul()	classcv_1_1MatExpr.html				2									
core_MatExpr_cross	CVAPI(ExceptionStatus) core_MatExpr_cross(cv::MatExpr *self, cv::Mat *m, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = self->cross(*m); *returnValue = new cv::Mat(ret); END_WRAP }	cv::MatExpr::cross()	classcv_1_1MatExpr.html				1									
core_MatExpr_dot	CVAPI(ExceptionStatus) core_MatExpr_dot(cv::MatExpr *self, cv::Mat *m, double *returnValue) { BEGIN_WRAP *returnValue = self->dot(*m); END_WRAP }	cv::MatExpr::dot()	classcv_1_1MatExpr.html				1									
core_MatExpr_size	CVAPI(ExceptionStatus) core_MatExpr_size(cv::MatExpr *self, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(self->size()); END_WRAP }	cv::MatExpr::size()	classcv_1_1MatExpr.html				1									
core_MatExpr_type	CVAPI(ExceptionStatus) core_MatExpr_type(cv::MatExpr *self, int *returnValue) { BEGIN_WRAP *returnValue = self->type(); END_WRAP }	cv::MatExpr::type()	classcv_1_1MatExpr.html				1									
core_operatorUnaryMinus_MatExpr	CVAPI(ExceptionStatus) core_operatorUnaryMinus_MatExpr(cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = -(*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),						•						

core_operatorUnaryNot_MatExpr	CVAPI(ExceptionStatus) core_operatorUnaryNot_MatExpr(cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = ~(*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorAdd_MatExprMat	CVAPI(ExceptionStatus) core_operatorAdd_MatExprMat(cv::MatExpr *e, cv:: Mat *m, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e) + (*m); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorAdd_MatMatExpr	CVAPI(ExceptionStatus) core_operatorAdd_MatMatExpr(cv::Mat *m, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*m) + (*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorAdd_MatExprScalar	CVAPI(ExceptionStatus) core_operatorAdd_MatExprScalar(cv::MatExpr *e, MyCvScalar s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e) + cpp(s); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorAdd_ScalarMatExpr	CVAPI(ExceptionStatus) core_operatorAdd_ScalarMatExpr(MyCvScalar s, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cpp(s) + (*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorAdd_MatExprMatExpr	CVAPI(ExceptionStatus) core_operatorAdd_MatExprMatExpr(cv::MatExpr *e1, cv::MatExpr *e2, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e1) + (*e2); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							
core_operatorSubtract_MatExprMat	CVAPI(ExceptionStatus) core_operatorSubtract_MatExprMat(cv::MatExpr *e, cv::Mat *m, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e) - (*m); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),					•							

core_operatorSubtract_MatMatExpr	CVAPI(ExceptionStatus) core_operatorSubtract_MatMatExpr(cv::Mat *m, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*m) - (*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorSubtract_MatExprScalar	CVAPI(ExceptionStatus) core_operatorSubtract_MatExprScalar(cv::MatExpr *e, MyCvScalar s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e) - cpp(s); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorSubtract_ScalarMatExpr	CVAPI(ExceptionStatus) core_operatorSubtract_ScalarMatExpr(MyCvScalar s, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = cpp(s) - (*e); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorSubtract_MatExprMatExpr	CVAPI(ExceptionStatus) core_operatorSubtract_MatExprMatExpr(cv::MatExpr *e1, cv::MatExpr *e2, cv::MatExpr **returnValue) { BEGIN_WRAP const auto expr = (*e1) - (*e2); *returnValue = new cv::MatExpr(expr); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorMultiply_MatExprMat	CVAPI(ExceptionStatus) core_operatorMultiply_MatExprMat(cv::MatExpr *e, cv::Mat *m, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e) * (*m); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorMultiply_MatMatExpr	CVAPI(ExceptionStatus) core_operatorMultiply_MatMatExpr(cv::Mat *m, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*m) * (*e); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						
core_operatorMultiply_MatExprDouble	CVAPI(ExceptionStatus) core_operatorMultiply_MatExprDouble(cv::MatExpr *e, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e) * s; *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),				•						

core_operatorMultiply_DoubleMatExpr	CVAPI(ExceptionStatus) core_operatorMultiply_DoubleMatExpr(double s, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = s * (*e); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorMultiply_MatExprMatExpr	CVAPI(ExceptionStatus) core_operatorMultiply_MatExprMatExpr(cv::MatExpr *e1, cv::MatExpr *e2, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e1) * (*e2); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorDivide_MatExprMat	CVAPI(ExceptionStatus) core_operatorDivide_MatExprMat(cv::MatExpr *e, cv::Mat *m, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e) / (*m); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorDivide_MatMatExpr	CVAPI(ExceptionStatus) core_operatorDivide_MatMatExpr(cv::Mat *m, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*m) / (*e); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorDivide_MatExprDouble	CVAPI(ExceptionStatus) core_operatorDivide_MatExprDouble(cv::MatExpr *e, double s, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e) / s; *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorDivide_DoubleMatExpr	CVAPI(ExceptionStatus) core_operatorDivide_DoubleMatExpr(double s, cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = s / (*e); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						
core_operatorDivide_MatExprMatExpr	CVAPI(ExceptionStatus) core_operatorDivide_MatExprMatExpr(cv::MatExpr *e1, cv::MatExpr *e2, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = (*e1) / (*e2); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &,	, ^. Element-wise minimum and maximum: min(A, B),					•						

core_abs_MatExpr	CVAPI(ExceptionStatus) core_abs_MatExpr(cv::MatExpr *e, cv::MatExpr **returnValue) { BEGIN_WRAP const auto ret = cv::abs(*e); *returnValue = new cv::MatExpr(ret); END_WRAP }	cv::MatExpr	classcv_1_1MatExpr.html	Matrix expression representation. This is a list of implemented matrix operations that can be combined in arbitrary complex expressions (here A, B stand for matrices (Mat), s for a scalar (Scalar), alpha for a real-valued scalar (double)): Addition, subtraction, negation: A+B, A-B, A+s, A-s, s+A, s-A, -A Scaling: A*alpha Per-element multiplication and division: A.mul(B), A/B, alpha/A Matrix multiplication: A*B Transposition: A.t() (means AT) Matrix inversion and pseudo-inversion, solving linear systems and least-squares problems: A.inv([method]) (~ A⁻¹*B), A.inv([method])*B (~ X: AX=B) Comparison: A cmpop B, A cmpop alpha, alpha cmpop A, where cmpop is one of >, >=, ==, !=, <=, <. The result of comparison is an 8-bit single channel mask whose elements are set to 255 (if the particular element or pair of elements satisfy the condition) or 0. Bitwise logical operations: A logicop B, A logicop s, s logicop A, ~A, where logicop is one of &, |, ^. Element-wise minimum and maximum: min(A, B),									•						
core_OutputArray_new_byMat	CVAPI(ExceptionStatus) core_OutputArray_new_byMat(cv::Mat *mat, cv::_OutputArray **returnValue) { BEGIN_WRAP const cv::_OutputArray ia(*mat); *returnValue = new cv::_OutputArray(ia); END_WRAP }	cv::_OutputArray	classcv_1_1__OutputArray.html	This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters), it is also not very difficult. Take the sample above as the reference. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not. There are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators: typedef OutputArray OutputArrayOfArrays;typedef OutputArray										•					
core_OutputArray_new_byUMat	CVAPI(ExceptionStatus) core_OutputArray_new_byUMat(cv::UMat* mat, cv::_OutputArray** returnValue) { BEGIN_WRAP const cv::_OutputArray ia(*mat); *returnValue = new cv::_OutputArray(ia); END_WRAP }	cv::_OutputArray	classcv_1_1__OutputArray.html	This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters), it is also not very difficult. Take the sample above as the reference. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not. There are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators: typedef OutputArray OutputArrayOfArrays;typedef OutputArray										•					
core_OutputArray_new_byScalar	CVAPI(ExceptionStatus) core_OutputArray_new_byScalar(MyCvScalar scalar, cv::_OutputArray **returnValue) { BEGIN_WRAP cv::Scalar scalarVal(cpp(scalar)); const cv::_OutputArray ia(scalarVal); *returnValue = new cv::_OutputArray(ia); END_WRAP }	cv::_OutputArray	classcv_1_1__OutputArray.html	This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters), it is also not very difficult. Take the sample above as the reference. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not. There are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators: typedef OutputArray OutputArrayOfArrays;typedef OutputArray										•					
core_OutputArray_new_byVectorOfMat	CVAPI(ExceptionStatus) core_OutputArray_new_byVectorOfMat(std::vector<cv::Mat> *vector, cv::_OutputArray **returnValue) { BEGIN_WRAP const cv::_OutputArray ia(*vector); *returnValue = new cv::_OutputArray(ia); END_WRAP }	cv::_OutputArray	classcv_1_1__OutputArray.html	This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters), it is also not very difficult. Take the sample above as the reference. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not. There are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators: typedef OutputArray OutputArrayOfArrays;typedef OutputArray										•					
core_OutputArray_delete	CVAPI(ExceptionStatus) core_OutputArray_delete(cv::_OutputArray *oa) { BEGIN_WRAP delete oa; END_WRAP }	cv::_OutputArray	classcv_1_1__OutputArray.html	This type is very similar to InputArray except that it is used for input/output and output function parameters. Just like with InputArray, OpenCV users should not care about OutputArray, they just pass Mat, vector<T> etc. to the functions. The same limitation as for InputArray: Do not explicitly create OutputArray instances applies here too. If you want to make your function polymorphic (i.e. accept different arrays as output parameters), it is also not very difficult. Take the sample above as the reference. Note that _OutputArray::create() needs to be called before _OutputArray::getMat(). This way you guarantee that the output array is properly allocated. Optional output parameters. If you do not need certain output array to be computed and returned to you, pass cv::noArray(), just like you would in the case of optional input array. At the implementation level, use _OutputArray::needed() to check if certain output array needs to be computed or not. There are several synonyms for OutputArray that are used to assist automatic Python/Java/... wrapper generators: typedef OutputArray OutputArrayOfArrays;typedef OutputArray														•	
core_OutputArray_getMat	CVAPI(ExceptionStatus) core_OutputArray_getMat(cv::_OutputArray *oa, cv::Mat **returnValue) { BEGIN_WRAP auto& mat = oa->getMatRef(); *returnValue = new cv::Mat(mat); END_WRAP }	cv::_OutputArray::getMatRef()	classcv_1_1__OutputArray.html				1									
core_OutputArray_getScalar	CVAPI(ExceptionStatus) core_OutputArray_getScalar(cv::_OutputArray *oa, MyCvScalar *returnValue) { BEGIN_WRAP cv::Mat &mat = oa->getMatRef(); const auto scalar = mat.at<cv::Scalar>(0); *returnValue = c(scalar); END_WRAP }	cv::_OutputArray::getMatRef()	classcv_1_1__OutputArray.html				1									
core_OutputArray_getVectorOfMat	CVAPI(ExceptionStatus) core_OutputArray_getVectorOfMat(cv::_OutputArray *oa, std::vector<cv::Mat*> *vector) { BEGIN_WRAP std::vector<cv::Mat> temp; oa->getMatVector(temp); vector->resize(temp.size()); for (size_t i = 0; i < temp.size(); i++) { (*vector)[i] = new cv::Mat(temp[i]); } END_WRAP }	cv::_OutputArray::getMatVector()															

core_SparseMat_new1	CVAPI(ExceptionStatus) core_SparseMat_new1(cv::SparseMat **returnValue) { BEGIN_WRAP *returnValue = new cv::SparseMat; END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:								•		
core_SparseMat_new2	CVAPI(ExceptionStatus) core_SparseMat_new2(int dims, const int *sizes, int type, cv::SparseMat **returnValue) { BEGIN_WRAP *returnValue = new cv::SparseMat(dims, sizes, type); END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:								•		
core_SparseMat_new3	CVAPI(ExceptionStatus) core_SparseMat_new3(cv::Mat *m, cv::SparseMat **returnValue) { BEGIN_WRAP *returnValue = new cv::SparseMat(*m); END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:								•		
core_SparseMat_delete	CVAPI(ExceptionStatus) core_SparseMat_delete(cv::SparseMat *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:								•		
core_SparseMat_operatorAssign_SparseMat	CVAPI(ExceptionStatus) core_SparseMat_operatorAssign_SparseMat(cv::SparseMat *obj, cv::SparseMat *m) { BEGIN_WRAP *obj = *m; END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:					•					
core_SparseMat_operatorAssign_Mat	CVAPI(ExceptionStatus) core_SparseMat_operatorAssign_Mat(cv::SparseMat *obj, cv::Mat *m) { BEGIN_WRAP *obj = *m; END_WRAP }	cv::SparseMat	classcv_1_1SparseMat.html	The class SparseMat represents multi-dimensional sparse numerical arrays. Such a sparse array can store elements of any type that Mat can store. Sparse means that only non-zero elements are stored (though, as a result of operations on a sparse matrix, some of its stored elements can actually become 0. It is up to you to detect such elements and delete them using SparseMat::erase). The non-zero elements are stored in a hash table that grows when it is filled so that the search time is O(1) in average (regardless of whether element is there or not). Elements can be accessed using the following methods: Query operations (SparseMat::ptr and the higher-level SparseMat::ref, SparseMat::value and SparseMat::find), for example: const int dims = 5;int size[5] = {10, 10, 10, 10, 10};SparseMat sparse_mat(dims, size, CV_32F);for(int i = 0; i < 1000; i++){ int idx[dims]; for(int k = 0; k < dims; k++) idx[k] = rand() % size[k]; sparse_mat.ref<float>(idx) += 1.f;}cout << "nnz = " << sparse_mat.nzcount() << endl; Sparse matrix iterators. They are similar to MatIterator but different from NAryMatIterator. That is, the iteration loop is familiar to STL users:					•					
core_SparseMat_clone	CVAPI(ExceptionStatus) core_SparseMat_clone(cv::SparseMat *obj, cv::SparseMat **returnValue) { BEGIN_WRAP const auto sm = obj->clone(); *returnValue = new cv::SparseMat(sm); END_WRAP }	cv::SparseMat::clone()	classcv_1_1SparseMat.html	creates full copy of the matrix			1								
core_SparseMat_copyTo_SparseMat	CVAPI(ExceptionStatus) core_SparseMat_copyTo_SparseMat(cv::SparseMat *obj, cv::SparseMat *m) { BEGIN_WRAP obj->copyTo(*m); END_WRAP }	cv::SparseMat::copyTo()	classcv_1_1SparseMat.html	copies all the data to the destination matrix. All the previous content of m is erased			1								
core_SparseMat_copyTo_Mat	CVAPI(ExceptionStatus) core_SparseMat_copyTo_Mat(cv::SparseMat *obj, cv::Mat *m) { BEGIN_WRAP obj->copyTo(*m); END_WRAP }	cv::SparseMat::copyTo()	classcv_1_1SparseMat.html	copies all the data to the destination matrix. All the previous content of m is erased			1								
core_SparseMat_convertTo_SparseMat	CVAPI(ExceptionStatus) core_SparseMat_convertTo_SparseMat(cv::SparseMat *obj, cv::SparseMat *m, int rtype, double alpha) { BEGIN_WRAP obj->convertTo(*m, rtype, alpha); END_WRAP }	cv::SparseMat::convertTo()	classcv_1_1SparseMat.html	converts sparse matrix to dense n-dim matrix with optional type conversion and scaling.	m : [out] rtype : [in] alpha : [in] beta : [in]		3								

core_SparseMat_convertTo_Mat	CVAPI(ExceptionStatus) core_SparseMat_convertTo_Mat(cv::SparseMat *obj, cv::Mat *m, int rtype, double alpha = 1, double beta = 0) { BEGIN_WRAP obj->convertTo(*m, rtype, alpha, beta); END_WRAP }	cv::SparseMat::convertTo()	classcv_1_1SparseMat.html	converts sparse matrix to dense n-dim matrix with optional type conversion and scaling.	m : [out] rtype : [in] alpha : [in] beta : [in]		4									
core_SparseMat_assignTo	CVAPI(ExceptionStatus) core_SparseMat_assignTo(cv::SparseMat *obj, cv::SparseMat *m, int type = -1) { BEGIN_WRAP obj->assignTo(*m, type); END_WRAP }	cv::SparseMat::assignTo()	classcv_1_1SparseMat.html				2									
core_SparseMat_create	CVAPI(ExceptionStatus) core_SparseMat_create(cv::SparseMat *obj, int dims, const int* sizes, int type) { BEGIN_WRAP obj->create(dims, sizes, type); END_WRAP }	cv::SparseMat::create()	classcv_1_1SparseMat.html	reallocates sparse matrix.		If the matrix already had the proper size and type, it is simply cleared with clear(), otherwise, the old matrix is released (using release()) and the new one is allocated.	3									
core_SparseMat_clear	CVAPI(ExceptionStatus) core_SparseMat_clear(cv::SparseMat *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::SparseMat::clear()	classcv_1_1SparseMat.html	sets all the sparse matrix elements to 0, which means clearing the hash table.			1									
core_SparseMat_addref	CVAPI(ExceptionStatus) core_SparseMat_addref(cv::SparseMat *obj) { BEGIN_WRAP obj->addref(); END_WRAP }	cv::SparseMat::addref()	classcv_1_1SparseMat.html	manually increments the reference counter to the header.			1									
core_SparseMat_release	CVAPI(ExceptionStatus) core_SparseMat_release(cv::SparseMat *obj) { BEGIN_WRAP obj->release(); END_WRAP }	cv::SparseMat::release()	classcv_1_1SparseMat.html				1									
core_SparseMat_elemSize	CVAPI(ExceptionStatus) core_SparseMat_elemSize(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->elemSize()); END_WRAP }	cv::SparseMat::elemSize()	classcv_1_1SparseMat.html	converts sparse matrix to the old-style representation; all the elements are copied.		returns the size of each element in bytes (not including the overhead - the space occupied by SparseMat::Node elements)	1									
core_SparseMat_elemSize1	CVAPI(ExceptionStatus) core_SparseMat_elemSize1(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->elemSize1()); END_WRAP }	cv::SparseMat::elemSize1()	classcv_1_1SparseMat.html	returns elemSize()/channels()			1									
core_SparseMat_type	CVAPI(ExceptionStatus) core_SparseMat_type(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->type(); END_WRAP }	cv::SparseMat::type()	classcv_1_1SparseMat.html	returns type of sparse matrix elements			1									
core_SparseMat_depth	CVAPI(ExceptionStatus) core_SparseMat_depth(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->depth(); END_WRAP }	cv::SparseMat::depth()	classcv_1_1SparseMat.html	returns the depth of sparse matrix elements			1									
core_SparseMat_channels	CVAPI(ExceptionStatus) core_SparseMat_channels(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->channels(); END_WRAP }	cv::SparseMat::channels()	classcv_1_1SparseMat.html	returns the number of channels			1									
core_SparseMat_size1	CVAPI(ExceptionStatus) core_SparseMat_size1(cv::SparseMat *obj, const int **returnValue) { BEGIN_WRAP *returnValue = obj->size(); END_WRAP }	cv::SparseMat::size()	classcv_1_1SparseMat.html	returns the array of sizes, or NULL if the matrix is not allocated			1									
core_SparseMat_size2	CVAPI(ExceptionStatus) core_SparseMat_size2(cv::SparseMat *obj, int i, int *returnValue) { BEGIN_WRAP *returnValue = obj->size(i); END_WRAP }	cv::SparseMat::size()	classcv_1_1SparseMat.html	returns the array of sizes, or NULL if the matrix is not allocated			1									
core_SparseMat_dims	CVAPI(ExceptionStatus) core_SparseMat_dims(cv::SparseMat *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->dims(); END_WRAP }	cv::SparseMat::dims()	classcv_1_1SparseMat.html	returns the matrix dimensionality			1									
core_SparseMat_nzcount	CVAPI(ExceptionStatus) core_SparseMat_nzcount(cv::SparseMat *obj, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->nzcount(); END_WRAP }	cv::SparseMat::nzcount()	classcv_1_1SparseMat.html	returns the number of non-zero elements (=the number of hash table nodes)			1									
core_SparseMat_hash_1d	CVAPI(ExceptionStatus) core_SparseMat_hash_1d(cv::SparseMat *obj, int i0, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->hash(i0); END_WRAP }	cv::SparseMat::hash()	classcv_1_1SparseMat.html	computes the element hash value (1D case)			1									
core_SparseMat_hash_2d	CVAPI(ExceptionStatus) core_SparseMat_hash_2d(cv::SparseMat *obj, int i0, int i1, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->hash(i0, i1); END_WRAP }	cv::SparseMat::hash()	classcv_1_1SparseMat.html	computes the element hash value (1D case)			2									
core_SparseMat_hash_3d	CVAPI(ExceptionStatus) core_SparseMat_hash_3d(cv::SparseMat *obj, int i0, int i1, int i2, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->hash(i0, i1, i2); END_WRAP }	cv::SparseMat::hash()	classcv_1_1SparseMat.html	computes the element hash value (1D case)			3									
core_SparseMat_hash_nd	CVAPI(ExceptionStatus) core_SparseMat_hash_nd(cv::SparseMat *obj, const int* idx, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->hash(idx); END_WRAP }	cv::SparseMat::hash()	classcv_1_1SparseMat.html	computes the element hash value (1D case)			1									

core_SparseMat_ptr_1d	CVAPI(ExceptionStatus) core_SparseMat_ptr_1d(cv::SparseMat *obj, int i0, int createMissing, uint64* hashVal, uchar **returnValue) { BEGIN_WRAP if (hashVal == nullptr) { *returnValue = obj->ptr(i0, createMissing != 0); }else { auto hashVal0 = static_cast<size_t>(*hashVal); *returnValue = obj->ptr(i0, createMissing != 0, &hashVal0); } END_WRAP }	cv::SparseMat::ptr()	classcv_1_1SparseMat.html	returns pointer to the specified element (1D case)		specialized variants for 1D, 2D, 3D cases and the generic_type one for n-D case. return pointer to the matrix element.if the element is there (it's non-zero), the pointer to it is returned if it's not there and createMissing=false, NULL pointer is returned if it's not there and createMissing=true, then the new element is created and initialized with 0. Pointer to it is returned if the optional hashval pointer is not NULL, the element hash value is not computed, but *hashval is taken instead.	1								
core_SparseMat_ptr_2d	CVAPI(ExceptionStatus) core_SparseMat_ptr_2d(cv::SparseMat *obj, int i0, int i1, int createMissing, uint64* hashVal, uchar **returnValue) { BEGIN_WRAP if (hashVal == nullptr) { *returnValue = obj->ptr(i0, i1, createMissing != 0); } else { auto hashVal0 = static_cast<size_t>(*hashVal); *returnValue = obj->ptr(i0, i1, createMissing != 0, &hashVal0); } END_WRAP }	cv::SparseMat::ptr()	classcv_1_1SparseMat.html	returns pointer to the specified element (1D case)		specialized variants for 1D, 2D, 3D cases and the generic_type one for n-D case. return pointer to the matrix element.if the element is there (it's non-zero), the pointer to it is returned if it's not there and createMissing=false, NULL pointer is returned if it's not there and createMissing=true, then the new element is created and initialized with 0. Pointer to it is returned if the optional hashval pointer is not NULL, the element hash value is not computed, but *hashval is taken instead.	1								
core_SparseMat_ptr_3d	CVAPI(ExceptionStatus) core_SparseMat_ptr_3d(cv::SparseMat *obj, int i0, int i1, int i2, int createMissing, uint64* hashVal, uchar **returnValue) { BEGIN_WRAP if (hashVal == nullptr) { *returnValue = obj->ptr(i0, i1, i2, createMissing != 0); } else { auto hashVal0 = static_cast<size_t>(*hashVal); *returnValue = obj->ptr(i0, i1, i2, createMissing != 0, &hashVal0); } END_WRAP }	cv::SparseMat::ptr()	classcv_1_1SparseMat.html	returns pointer to the specified element (1D case)		specialized variants for 1D, 2D, 3D cases and the generic_type one for n-D case. return pointer to the matrix element.if the element is there (it's non-zero), the pointer to it is returned if it's not there and createMissing=false, NULL pointer is returned if it's not there and createMissing=true, then the new element is created and initialized with 0. Pointer to it is returned if the optional hashval pointer is not NULL, the element hash value is not computed, but *hashval is taken instead.	1								
core_SparseMat_ptr_nd	CVAPI(ExceptionStatus) core_SparseMat_ptr_nd(cv::SparseMat *obj, const int* idx, int createMissing, uint64* hashVal, uchar **returnValue) { BEGIN_WRAP if (hashVal == nullptr) { *returnValue = obj->ptr(idx, createMissing != 0); } else { auto hashVal0 = static_cast<size_t>(*hashVal); *returnValue = obj->ptr(idx, createMissing != 0, &hashVal0); } END_WRAP }	cv::SparseMat::ptr()	classcv_1_1SparseMat.html	returns pointer to the specified element (1D case)		specialized variants for 1D, 2D, 3D cases and the generic_type one for n-D case. return pointer to the matrix element.if the element is there (it's non-zero), the pointer to it is returned if it's not there and createMissing=false, NULL pointer is returned if it's not there and createMissing=true, then the new element is created and initialized with 0. Pointer to it is returned if the optional hashval pointer is not NULL, the element hash value is not computed, but *hashval is taken instead.	1								
core_UMat_new1	CVAPI(ExceptionStatus) core_UMat_new1(const cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(usageFlags); END_WRAP }	cv::Umat										•				
core_UMat_new2	CVAPI(ExceptionStatus) core_UMat_new2(const int rows, const int cols, const int type, const cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(rows, cols, type, usageFlags); END_WRAP }	cv::Umat										•				
core_UMat_new3	CVAPI(ExceptionStatus) core_UMat_new3(cv::Size size, int type, cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(size, type, usageFlags); END_WRAP }	cv::Umat										•				
core_UMat_new4	CVAPI(ExceptionStatus) core_UMat_new4(const int ndims, const int* sizes, const int type, const cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(ndims, sizes, type, usageFlags); END_WRAP }	cv::Umat										•				
core_UMat_new5	CVAPI(ExceptionStatus) core_UMat_new5(cv::Size size, int type, MyCvScalar s, cv::UMatUsageFlags usageFlags, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(size, type, cpp(s), usageFlags); END_WRAP }	cv::Umat										•				
core_UMat_new6	CVAPI(ExceptionStatus) core_UMat_new6(cv::UMat* umat, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(*umat); END_WRAP }	cv::Umat										•				
core_UMat_new7	CVAPI(ExceptionStatus) core_UMat_new7(cv::UMat* umat, const MyCvSlice rowRange, const MyCvSlice colRange, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(*umat, cpp(rowRange), cpp(colRange)); END_WRAP }	cv::Umat										•				
core_UMat_new8	CVAPI(ExceptionStatus) core_UMat_new8(cv::UMat* umat, const MyCvRect roi, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(*umat, cpp(roi)); END_WRAP }	cv::Umat										•				

core_UMat_new9	CVAPI(ExceptionStatus) core_UMat_new9(cv::UMat* umat, cv::Range* ranges, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(*umat, ranges); END_WRAP }	cv::Umat											•			
core_UMat_delete	CVAPI(ExceptionStatus) core_UMat_delete(cv::UMat* self) { BEGIN_WRAP delete self; END_WRAP }	cv::Umat												•		
core_UMat_getMat	CVAPI(ExceptionStatus) core_UMat_getMat(cv::UMat* self, cv::AccessFlag accessFlag, cv::Mat** returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(self- >getMat(accessFlag)); END_WRAP }	cv::Umat::getMat()														
core_UMat_row	CVAPI(ExceptionStatus) core_UMat_row(cv::UMat* self, int y, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(self->row(y)); END_WRAP }	cv::Umat::row()														
core_UMat_col	CVAPI(ExceptionStatus) core_UMat_col(cv::UMat* self, int x, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(self->col(x)); END_WRAP }	cv::Umat::col()														
core_UMat_rowRange	CVAPI(ExceptionStatus) core_UMat_rowRange(cv::UM at* self, int startRow, int endRow, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(self- >rowRange(startRow, endRow)); END_WRAP }	cv::Umat::rowRange()														
core_UMat_colRange	CVAPI(ExceptionStatus) core_UMat_colRange(cv::UMa t* self, int startCol, int endCol, cv::UMat** returnValue) { BEGIN_WRAP *returnValue = new cv::UMat(self- >colRange(startCol, endCol)); END_WRAP }	cv::Umat::colRange()														
core_UMat_diag	CVAPI(ExceptionStatus) core_UMat_diag(cv::UMat* self, int d, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self- >diag(d); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::diag()														
core_UMat_diag_static	CVAPI(ExceptionStatus) core_UMat_diag_static(cv::U Mat* self, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = cv::UMat::diag(*self); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::diag()														
core_UMat_clone	CVAPI(ExceptionStatus) core_UMat_clone(cv::UMat* self, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self- >clone(); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::clone()														
core_UMat_copyTo1	CVAPI(ExceptionStatus) core_UMat_copyTo1(cv::UMat * self, cv::_OutputArray* m) { BEGIN_WRAP self->copyTo(*m); END_WRAP }	cv::Umat::copyTo()														
core_UMat_copyTo2	CVAPI(ExceptionStatus) core_UMat_copyTo2(cv::UMat * self, cv::_OutputArray* m, cv::_InputArray* mask) { BEGIN_WRAP self->copyTo(*m, entity(mask)); END_WRAP }	cv::Umat::copyTo()														
core_UMat_copyTo_toUMat1	CVAPI(ExceptionStatus) core_UMat_copyTo_toUMat1(cv::UMat* self, cv::UMat* m) { BEGIN_WRAP self->copyTo(*m); END_WRAP }	cv::Umat::copyTo()														
core_UMat_copyTo_toUMat2	CVAPI(ExceptionStatus) core_UMat_copyTo_toUMat2(cv::UMat* self, cv::UMat* m, cv::_InputArray* mask) { BEGIN_WRAP self->copyTo(*m, entity(mask)); END_WRAP }	cv::Umat::copyTo()														
core_UMat_convertTo	CVAPI(ExceptionStatus) core_UMat_convertTo(cv::UM at* self, cv::_OutputArray* m, int rtype, double alpha, double beta) { BEGIN_WRAP self->convertTo(*m, rtype, alpha, beta); END_WRAP }	cv::Umat::convertTo()														
core_UMat_assignTo	CVAPI(ExceptionStatus) core_UMat_assignTo(cv::UMa t* self, cv::UMat* m, int type) { BEGIN_WRAP self->assignTo(*m, type); END_WRAP }	cv::Umat::assignTo()														
core_UMat_setTo_Scalar	CVAPI(ExceptionStatus) core_UMat_setTo_Scalar(cv:: UMat* self, MyCvScalar value, cv::UMat* mask) { BEGIN_WRAP if (mask == nullptr) self- >setTo(cpp(value)); else self->setTo(cpp(value), entity(mask)); END_WRAP }	cv::Umat::setTo()														
core_UMat_setTo_InputArray	CVAPI(ExceptionStatus) core_UMat_setTo_InputArray(cv::UMat* self, cv::_InputArray* value, cv::UMat* mask) { BEGIN_WRAP self->setTo(*value, entity(mask)); END_WRAP }	cv::Umat::setTo()														

core_UMat_reshape1	CVAPI(ExceptionStatus) core_UMat_reshape1(cv::UMat* self, int cn, int rows, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self->reshape(cn, rows); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::reshape()																
core_UMat_reshape2	CVAPI(ExceptionStatus) core_UMat_reshape2(cv::UMat* self, int cn, int newndims, const int* newsz, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self->reshape(cn, newndims, newsz); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::reshape()																
core_UMat_t	CVAPI(ExceptionStatus) core_UMat_t(cv::UMat* self, cv::UMat** returnValue) { BEGIN_WRAP const auto expr = self->t(); *returnValue = new cv::UMat(expr); END_WRAP }	cv::Umat::t()																
core_UMat_inv	CVAPI(ExceptionStatus) core_UMat_inv(cv::UMat* self, int method, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self->inv(method); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::inv()																
core_UMat_mul	CVAPI(ExceptionStatus) core_UMat_mul(cv::UMat* self, cv::_InputArray* m, double scale, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self->mul(*m, scale); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::mul()																
core_UMat_dot	CVAPI(ExceptionStatus) core_UMat_dot(cv::UMat* self, cv::_InputArray* m, double* returnValue) { BEGIN_WRAP *returnValue = self->dot(*m); END_WRAP }	cv::Umat::dot()																
core_UMat_zeros1	CVAPI(ExceptionStatus) core_UMat_zeros1(int rows, int cols, int type, cv::UMat** returnValue) { BEGIN_WRAP const auto expr = cv::UMat::zeros(rows, cols, type); *returnValue = new cv::UMat(expr); END_WRAP }	cv::Umat::zeros()																
core_UMat_zeros2	CVAPI(ExceptionStatus) core_UMat_zeros2(int ndims, const int* sz, int type, cv::UMat** returnValue) { BEGIN_WRAP const auto expr = cv::UMat::zeros(ndims, sz, type); *returnValue = new cv::UMat(expr); END_WRAP }	cv::Umat::zeros()																
core_UMat_ones1	CVAPI(ExceptionStatus) core_UMat_ones1(int rows, int cols, int type, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = cv::UMat::ones(rows, cols, type); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::ones()																
core_UMat_ones2	CVAPI(ExceptionStatus) core_UMat_ones2(int ndims, const int* sz, int type, cv::UMat** returnValue) { BEGIN_WRAP cv::UMat ret = cv::UMat::ones(ndims, sz, type); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::ones()																
core_UMat_eye	CVAPI(ExceptionStatus) core_UMat_eye(int rows, int cols, int type, cv::UMat** returnValue) { BEGIN_WRAP const auto eye = cv::UMat::eye(rows, cols, type); *returnValue = new cv::UMat(eye); END_WRAP }	cv::Umat::eye()																
core_UMat_create1	CVAPI(ExceptionStatus) core_UMat_create1(cv::UMat* self, int rows, int cols, int type, cv::UMatUsageFlags usageFlags) { BEGIN_WRAP self->create(rows, cols, type, usageFlags); END_WRAP }	cv::Umat::create()																
core_UMat_create2	CVAPI(ExceptionStatus) core_UMat_create2(cv::UMat* self, int ndims, const int* sizes, int type, cv::UMatUsageFlags usageFlags) { BEGIN_WRAP self->create(ndims, sizes, type, usageFlags); END_WRAP }	cv::Umat::create()																
core_UMat_locateROI	CVAPI(ExceptionStatus) core_UMat_locateROI(cv::UMat* self, MyCvSize* wholeSize, MyCvPoint* ofs) { BEGIN_WRAP cv::Size wholeSize2; cv::Point ofs2; self->locateROI(wholeSize2, ofs2); *wholeSize = c(cv::Size(wholeSize2.width, wholeSize2.height)); *ofs = c(cv::Point(ofs2.x, ofs2.y)); END_WRAP }	cv::Umat::locateROI()																
core_UMat_adjustROI	CVAPI(ExceptionStatus) core_UMat_adjustROI(cv::UMat* self, int dtop, int dbottom, int dleft, int dright, cv::UMat** returnValue) { BEGIN_WRAP const auto ret = self->adjustROI(dtop, dbottom, dleft, dright); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat::adjustROI()																

core_UMat_subMat1	CVAPI(ExceptionStatus) core_UMat_subMat1(cv::UMat* self, int rowStart, int rowEnd, int colStart, int colEnd, cv::UMat** returnValue) { BEGIN_WRAP const cv::Range rowRange(rowStart, rowEnd); const cv::Range colRange(colStart, colEnd); const auto ret = (*self)(rowRange, colRange); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat								•							
core_UMat_subMat2	CVAPI(ExceptionStatus) core_UMat_subMat2(cv::UMat* self, int nRanges, MyCvSlice* ranges, cv::UMat** returnValue) { BEGIN_WRAP std::vector<cv::Range> rangesVec(nRanges); for (auto i = 0; i < nRanges; i++) { rangesVec[i] = (cpp(ranges[i])); } const auto ret = (*self)(&rangesVec[0]); *returnValue = new cv::UMat(ret); END_WRAP }	cv::Umat								•							
core_UMat_isContinuous	CVAPI(ExceptionStatus) core_UMat_isContinuous(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->isContinuous() ? 1 : 0; END_WRAP }	cv::Umat::isContinuous()															
core_UMat_isSubmatrix	CVAPI(ExceptionStatus) core_UMat_isSubmatrix(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->isSubmatrix() ? 1 : 0; END_WRAP }	cv::Umat::isSubmatrix()															
core_UMat_elemSize	CVAPI(ExceptionStatus) core_UMat_elemSize(cv::UMat* self, size_t* returnValue) { BEGIN_WRAP *returnValue = self->elemSize(); END_WRAP }	cv::Umat::elemSize()															
core_UMat_elemSize1	CVAPI(ExceptionStatus) core_UMat_elemSize1(cv::UMat* self, size_t* returnValue) { BEGIN_WRAP *returnValue = self->elemSize1(); END_WRAP }	cv::Umat::elemSize1()															
core_UMat_type	CVAPI(ExceptionStatus) core_UMat_type(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->type(); END_WRAP }	cv::Umat::type()															
core_UMat_depth	CVAPI(ExceptionStatus) core_UMat_depth(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->depth(); END_WRAP }	cv::Umat::depth()															
core_UMat_channels	CVAPI(ExceptionStatus) core_UMat_channels(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->channels(); END_WRAP }	cv::Umat::channels()															
core_UMat_step1	CVAPI(ExceptionStatus) core_UMat_step1(cv::UMat* self, int i, size_t* returnValue) { BEGIN_WRAP *returnValue = self->step1(); END_WRAP }	cv::Umat::step1()															
core_UMat_empty	CVAPI(ExceptionStatus) core_UMat_empty(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->empty() ? 1 : 0; END_WRAP }	cv::Umat::empty()															
core_UMat_total	CVAPI(ExceptionStatus) core_UMat_total(cv::UMat* self, size_t* returnValue) { BEGIN_WRAP *returnValue = self->total(); END_WRAP }	cv::Umat::total()															
core_UMat_checkVector	CVAPI(ExceptionStatus) core_UMat_checkVector(cv::UMat* self, int elemChannels, int depth, int requireContinuous, int* returnValue) { BEGIN_WRAP *returnValue = self->checkVector(elemChannels, depth, requireContinuous != 0); END_WRAP }	cv::Umat::checkVector()															
core_UMat_flags	CVAPI(ExceptionStatus) core_UMat_flags(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->flags; END_WRAP }	cv::Umat::flags									•						
core_UMat_dims	CVAPI(ExceptionStatus) core_UMat_dims(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->dims; END_WRAP }	cv::Umat::dims									•						
core_UMat_rows	CVAPI(ExceptionStatus) core_UMat_rows(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->rows; END_WRAP }	cv::Umat::rows									•						
core_UMat_cols	CVAPI(ExceptionStatus) core_UMat_cols(cv::UMat* self, int* returnValue) { BEGIN_WRAP *returnValue = self->cols; END_WRAP }	cv::Umat::cols									•						
core_UMat_size	CVAPI(ExceptionStatus) core_UMat_size(cv::UMat* self, MyCvSize* returnValue) { BEGIN_WRAP *returnValue = c(self->size()); END_WRAP }	cv::Umat::size()															

core_UMat_sizeAt	CVAPI(ExceptionStatus) core_UMat_sizeAt(cv::UMat* self, int i, int* returnValue) { BEGIN_WRAP *returnValue = self->size[i]; END_WRAP }	cv::Umat::size								•						
core_UMat_step	CVAPI(ExceptionStatus) core_UMat_step1(cv::UMat* self, int i, size_t* returnValue) { BEGIN_WRAP *returnValue = self->step1(i); END_WRAP }	cv::Umat::step1()														
core_UMat_stepAt	CVAPI(ExceptionStatus) core_UMat_stepAt(cv::UMat* self, int i, size_t* returnValue) { BEGIN_WRAP *returnValue = self->step[i]; END_WRAP }	cv::Umat::step								•						
dnn_readNetFromDarknet_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromDarknet(const char *cfgFile, const char *darknetModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto darknetModelStr = (darknetModel == nullptr) ? cv::String() : cv::String(darknetModel); const auto net = cv::dnn::readNetFromDarknet(cfgFile, darknetModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromDarknet()	group__dnn.html	Reads a network model stored in Darknet model files.	cfgFile : path to the .cfg file with text description of the network architecture. darknetModel : path to the .weights file with learned network.		2									
dnn_readNetFromDarknet_Windows	CVAPI(ExceptionStatus) dnn_readNetFromDarknet(const char *cfgFile, const char *darknetModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto darknetModelStr = (darknetModel == nullptr) ? cv::String() : cv::String(darknetModel); const auto net = cv::dnn::readNetFromDarknet(cfgFile, darknetModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromDarknet()	group__dnn.html	Reads a network model stored in Darknet model files.	cfgFile : path to the .cfg file with text description of the network architecture. darknetModel : path to the .weights file with learned network.		2									
dnn_readNetFromDarknet	CVAPI(ExceptionStatus) dnn_readNetFromDarknet_InputArray(const char* bufferCfg, size_t lenCfg, const char* bufferModel, size_t lenModel, cv::dnn::Net** returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromDarknet(bufferCfg, lenCfg, bufferModel, lenModel); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromDarknet()	group__dnn.html	Reads a network model stored in Darknet model files.	bufferCfg : A buffer contains a content of .cfg file with text description of the network architecture. lenCfg : Number of bytes to read from bufferCfg bufferModel : A buffer contains a content of .weights file with learned network. lenModel : Number of bytes to read from bufferModel		4									
dnn_readNetFromCaffe_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromCaffe(const char *prototxt, const char *caffeModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto caffeModelStr = (caffeModel == nullptr) ? cv::String() : cv::String(caffeModel); const auto net = cv::dnn::readNetFromCaffe(prototxt, caffeModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromCaffe()	group__dnn.html	Reads a network model stored in Caffe framework's format.	prototxt : path to the .prototxt file with text description of the network architecture. caffeModel : path to the .caffemodel file with learned network.	Examples: samples/dnn/colorization.cpp.	2									
dnn_readNetFromCaffe_Windows	CVAPI(ExceptionStatus) dnn_readNetFromCaffe(const char *prototxt, const char *caffeModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto caffeModelStr = (caffeModel == nullptr) ? cv::String() : cv::String(caffeModel); const auto net = cv::dnn::readNetFromCaffe(prototxt, caffeModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromCaffe()	group__dnn.html	Reads a network model stored in Caffe framework's format.	prototxt : path to the .prototxt file with text description of the network architecture. caffeModel : path to the .caffemodel file with learned network.	Examples: samples/dnn/colorization.cpp.	2									
dnn_readNetFromCaffe	CVAPI(ExceptionStatus) dnn_readNetFromCaffe_InputArray(const char* bufferProto, size_t lenProto, const char* bufferModel, size_t lenModel, cv::dnn::Net** returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromCaffe(bufferProto, lenProto, bufferModel, lenModel); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromCaffe()	group__dnn.html	Reads a network model stored in Caffe model in memory.	bufferProto : buffer containing the content of the .prototxt file lenProto : length of bufferProto bufferModel : buffer containing the content of the .caffemodel file lenModel : length of bufferModel	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	4									
dnn_readNetFromTensorflow_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromTensorflow(const char *model, const char *config, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto configStr = (config == nullptr) ? cv::String() : cv::String(config); const auto net = cv::dnn::readNetFromTensorflow(model, configStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromTensorflow()	group__dnn.html	Reads a network model stored in TensorFlow framework's format.	model : path to the .pb file with binary protobuf description of the network architecture config : path to the .pbtxt file that contains text graph definition in protobuf format. Resulting Net object is built by text graph using weights from a binary one that let us make it more flexible.		2									
dnn_readNetFromTensorflow_Windows	CVAPI(ExceptionStatus) dnn_readNetFromTensorflow(const char *model, const char *config, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto configStr = (config == nullptr) ? cv::String() : cv::String(config); const auto net = cv::dnn::readNetFromTensorflow(model, configStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromTensorflow()	group__dnn.html	Reads a network model stored in TensorFlow framework's format.	model : path to the .pb file with binary protobuf description of the network architecture config : path to the .pbtxt file that contains text graph definition in protobuf format. Resulting Net object is built by text graph using weights from a binary one that let us make it more flexible.		2									
dnn_readNetFromTensorflow	CVAPI(ExceptionStatus) dnn_readNetFromTensorflow_InputArray(const char *model,size_t lenModel, const char *config, size_t lenConfig, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromTensorflow(model, lenModel, config, lenConfig); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromTensorflow()	group__dnn.html	Reads a network model stored in TensorFlow framework's format.	bufferModel : buffer containing the content of the pb file lenModel : length of bufferModel bufferConfig : buffer containing the content of the pbtxt file lenConfig : length of bufferConfig	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	4									

dnn_readNetFromTorch_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromTorch(const char *model, const int isBinary, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromTorch(model, isBinary != 0); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromTorch()	group__dnn.html	Reads a network model stored in Torch7 framework's format.	model : path to the file, dumped from Torch by using torch.save() function. isBinary : specifies whether the network was serialized in ascii mode or binary. evaluate : specifies testing phase of network. If true, it's similar to evaluate() method in Torch.	NoteAScii mode of Torch serializer is more preferable, because binary mode extensively use long type of C language, which has various bit-length on different systems.The loading file must contain serialized nn.Module object with importing network. Try to eliminate a custom objects from serialazing data to avoid importing errors.List of supported layers (i.e. object instances derived from Torch nn.Module class):nn.Sequential nn.Parallel nn.Concat nn.Linear nn.SpatialConvolution nn.SpatialMaxPooling, nn.SpatialAveragePooling nn.ReLU, nn.TanH, nn.Sigmoid nn.Reshape nn.SoftMax, nn.LogSoftMaxAlso some equivalents of these classes from cunn, cudnn, and fbcunn may be successfully	2								
dnn_readNetFromTorch_Windows	CVAPI(ExceptionStatus) dnn_readNetFromTorch(const char *model, const int isBinary, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromTorch(model, isBinary != 0); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromTorch()	group__dnn.html	Reads a network model stored in Torch7 framework's format.	model : path to the file, dumped from Torch by using torch.save() function. isBinary : specifies whether the network was serialized in ascii mode or binary. evaluate : specifies testing phase of network. If true, it's similar to evaluate() method in Torch.	NoteAScii mode of Torch serializer is more preferable, because binary mode extensively use long type of C language, which has various bit-length on different systems.The loading file must contain serialized nn.Module object with importing network. Try to eliminate a custom objects from serialazing data to avoid importing errors.List of supported layers (i.e. object instances derived from Torch nn.Module class):nn.Sequential nn.Parallel nn.Concat nn.Linear nn.SpatialConvolution nn.SpatialMaxPooling, nn.SpatialAveragePooling nn.ReLU, nn.TanH, nn.Sigmoid nn.Reshape nn.SoftMax, nn.LogSoftMaxAlso some equivalents of these classes from cunn, cudnn, and fbcunn may be successfully	2								
dnn_readNet_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromDarknet(const char *cfgFile, const char *darknetModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto darknetModelStr = (darknetModel == nullptr) ? cv::String() : cv::String(darknetModel); const auto net = cv::dnn::readNetFromDarknet (cfgFile, darknetModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromDarknet()	group__dnn.html	Reads a network model stored in Darknet model files.	cfgFile : path to the .cfg file with text description of the network architecture. darknetModel : path to the .weights file with learned network.		2								
dnn_readNet_Windows	CVAPI(ExceptionStatus) dnn_readNetFromDarknet(const char *cfgFile, const char *darknetModel, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto darknetModelStr = (darknetModel == nullptr) ? cv::String() : cv::String(darknetModel); const auto net = cv::dnn::readNetFromDarknet (cfgFile, darknetModelStr); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromDarknet()	group__dnn.html	Reads a network model stored in Darknet model files.	cfgFile : path to the .cfg file with text description of the network architecture. darknetModel : path to the .weights file with learned network.		2								
dnn_readTorchBlob_NotWindows	CVAPI(ExceptionStatus) dnn_readTorchBlob(const char *filename, const int isBinary, cv::Mat **returnValue) { BEGIN_WRAP const auto blob = cv::dnn::readTorchBlob(filena me, isBinary != 0); *returnValue = new cv::Mat(blob); END_WRAP }	cv::dnn::readTorchBlob()	group__dnn.html	Loads blob which was serialized as torch.Tensor object of Torch7 framework.		WarningThis function has the same limitations as readNetFromTorch().	2								
dnn_readTorchBlob_Windows	CVAPI(ExceptionStatus) dnn_readTorchBlob(const char *filename, const int isBinary, cv::Mat **returnValue) { BEGIN_WRAP const auto blob = cv::dnn::readTorchBlob(filena me, isBinary != 0); *returnValue = new cv::Mat(blob); END_WRAP }	cv::dnn::readTorchBlob()	group__dnn.html	Loads blob which was serialized as torch.Tensor object of Torch7 framework.		WarningThis function has the same limitations as readNetFromTorch().	2								
dnn_readNetFromModelOptimizer_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromModelOptimizer(const char *xml, const char *bin, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromModelO ptimizer(xml, bin); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromModelOptimizer()	group__dnn.html	Load a network from Intel's Model Optimizer intermediate representation.	xml : [in] bin : [in]		2								
dnn_readNetFromModelOptimizer_Windows	CVAPI(ExceptionStatus) dnn_readNetFromModelOptimizer(const char *xml, const char *bin, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromModelO ptimizer(xml, bin); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromModelOptimizer()	group__dnn.html	Load a network from Intel's Model Optimizer intermediate representation.	xml : [in] bin : [in]		2								
dnn_readNetFromONNX_NotWindows	CVAPI(ExceptionStatus) dnn_readNetFromONNX(const char *onnxFile, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromONNX(onnxFile); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromONNX()	group__dnn.html	Reads a network model ONNX.	onnxFile : path to the .onnx file with text description of the network architecture.		1								
dnn_readNetFromONNX_Windows	CVAPI(ExceptionStatus) dnn_readNetFromONNX(const char *onnxFile, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromONNX(onnxFile); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromONNX()	group__dnn.html	Reads a network model ONNX.	onnxFile : path to the .onnx file with text description of the network architecture.		1								
dnn_readNetFromONNX	CVAPI(ExceptionStatus) dnn_readNetFromONNX_InputArray(const char* buffer, size_t sizeBuffer, cv::dnn::Net** returnValue) { BEGIN_WRAP const auto net = cv::dnn::readNetFromONNX(buffer, sizeBuffer); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::readNetFromONNX()	group__dnn.html	Reads a network model from ONNX in-memory buffer.	buffer : memory address of the first byte of the buffer. sizeBuffer : size of the buffer.		2								
dnn_readTensorFromONNX_NotWindows	CVAPI(ExceptionStatus) dnn_readTensorFromONNX(const char *path, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::dnn::readTensorFromONN X(path); *returnValue = new cv::Mat(mat); END_WRAP }	cv::dnn::readTensorFromONNX()	group__dnn.html	Creates blob from .pb file.	path : to the .pb file with input tensor.		1								

dnn_readTensorFromONNX_Windows	CVAPI(ExceptionStatus) dnn_readTensorFromONNX(co nst char *path, cv::Mat **returnValue) { BEGIN_WRAP const auto mat = cv::dnn::readTensorFromONN X(path); *returnValue = new cv::Mat(mat); END_WRAP }	cv::dnn::readTensorFromONNX()	group__dnn.html	Creates blob from .pb file.	path : to the .pb file with input tensor.		1									
dnn_blobFromImage	CVAPI(ExceptionStatus) dnn_blobFromImage(cv::Mat *image, const double scalefactor, const MyCvSize size, const MyCvScalar mean, const int swapRB, const int crop, cv::Mat **returnValue) { BEGIN_WRAP const auto blob = cv::dnn::blobFromImage(*im age, scalefactor, cpp(size), cpp(mean), swapRB != 0, crop != 0); *returnValue = new cv::Mat(blob); END_WRAP }	cv::dnn::blobFromImage()	group__dnn.html	Creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtract mean values, scales values by scalefactor, swap Blue and Red channels.	image : input image (with 1-, 3- or 4-channels). size : spatial size for output image mean : scalar with mean values which are subtracted from channels. Values are intended to be in (mean-R, mean-G, mean-B) order if image has BGR ordering and swapRB is true. scalefactor : multiplier for image values. swapRB : flag which indicates that swap first and last channels in 3-channel image is necessary. crop : flag which indicates whether image will be cropped after resize or not ddepth : Depth of output blob. Choose CV_32F or CV_8U.	if crop is true, input image is resized so one side after resize is equal to corresponding dimension in size and another one is equal or larger. Then, crop from the center is performed. If crop is false, direct resize without cropping and preserving aspect ratio is performed.Examples: samples/dnn/classification.cpp, samples/dnn/colorization.cpp, samples/dnn/object_detection.cpp, samples/dnn/openpose.cpp, and samples/dnn/segmentation.cpp.	6								
dnn_blobFromImages	CVAPI(ExceptionStatus) dnn_blobFromImages(const cv::Mat **images, const int imagesLength, const double scalefactor, const MyCvSize size, const MyCvScalar mean, const int swapRB, const int crop, cv::Mat **returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesLength, imagesVec); const auto blob = cv::dnn::blobFromImages(im agesVec, scalefactor, cpp(size), cpp(mean), swapRB != 0, crop != 0); *returnValue = new cv::Mat(blob); END_WRAP }	cv::dnn::blobFromImages()	group__dnn.html	Creates 4-dimensional blob from series of images. Optionally resizes and crops images from center, subtract mean values, scales values by scalefactor, swap Blue and Red channels.	images : input images (all with 1-, 3- or 4-channels). size : spatial size for output image mean : scalar with mean values which are subtracted from channels. Values are intended to be in (mean-R, mean-G, mean-B) order if image has BGR ordering and swapRB is true. scalefactor : multiplier for images values. swapRB : flag which indicates that swap first and last channels in 3-channel image is necessary. crop : flag which indicates whether image will be cropped after resize or not ddepth : Depth of output blob. Choose CV_32F or CV_8U.	if crop is true, input image is resized so one side after resize is equal to corresponding dimension in size and another one is equal or larger. Then, crop from the center is performed. If crop is false, direct resize without cropping and preserving aspect ratio is performed.	6								
dnn_shrinkCaffeModel_NotWindows	CVAPI(ExceptionStatus) dnn_shrinkCaffeModel(const char *src, const char *dst, const char **layersTypes, int layersTypesSize) { BEGIN_WRAP std::vector<cv::String> layersTypesVec(layersTypesSi ze); for (int i = 0; i < layersTypesSize; i++) { layersTypesVec[i].assign(layer sTypes[i]); } cv::dnn::shrinkCaffeModel(src , dst, layersTypesVec); END_WRAP }	cv::dnn::shrinkCaffeModel()	group__dnn.html	Convert all weights of Caffe network to half precision floating point.	src : Path to origin model from Caffe framework contains single precision floating point weights (usually has .caffemodel extension). dst : Path to destination model with updated weights. layersTypes : Set of layers types which parameters will be converted. By default, converts only Convolutional and Fully-Connected layers' weights.	NoteShrinked model has no origin float32 weights so it can't be used in origin Caffe framework anymore. However the structure of data is taken from NVidia's Caffe fork: https://github.com/NVIDIA/caffe. So the resulting model may be used there.	3								
dnn_shrinkCaffeModel_Windows	CVAPI(ExceptionStatus) dnn_shrinkCaffeModel(const char *src, const char *dst, const char **layersTypes, int layersTypesSize) { BEGIN_WRAP std::vector<cv::String> layersTypesVec(layersTypesSi ze); for (int i = 0; i < layersTypesSize; i++) { layersTypesVec[i].assign(layer sTypes[i]); } cv::dnn::shrinkCaffeModel(src , dst, layersTypesVec); END_WRAP }	cv::dnn::shrinkCaffeModel()	group__dnn.html	Convert all weights of Caffe network to half precision floating point.	src : Path to origin model from Caffe framework contains single precision floating point weights (usually has .caffemodel extension). dst : Path to destination model with updated weights. layersTypes : Set of layers types which parameters will be converted. By default, converts only Convolutional and Fully-Connected layers' weights.	NoteShrinked model has no origin float32 weights so it can't be used in origin Caffe framework anymore. However the structure of data is taken from NVidia's Caffe fork: https://github.com/NVIDIA/caffe. So the resulting model may be used there.	3								
dnn_writeTextGraph_NotWindows	CVAPI(ExceptionStatus) dnn_writeTextGraph(const char *model, const char *output) { BEGIN_WRAP cv::dnn::writeTextGraph(mod el, output); END_WRAP }	cv::dnn::writeTextGraph()	group__dnn.html	Create a text representation for a binary network stored in protocol buffer format.	model : [in] output : [in]	NoteTo reduce output file size, trained weights are not included.	2									
dnn_writeTextGraph_Windows	CVAPI(ExceptionStatus) dnn_writeTextGraph(const char *model, const char *output) { BEGIN_WRAP cv::dnn::writeTextGraph(mod el, output); END_WRAP }	cv::dnn::writeTextGraph()	group__dnn.html	Create a text representation for a binary network stored in protocol buffer format.	model : [in] output : [in]	NoteTo reduce output file size, trained weights are not included.	2									
dnn_NMSBoxes_Rect	CVAPI(ExceptionStatus) dnn_NMSBoxes_Rect(std::vec tor<cv::Rect> *bboxes, std::vector<float> *scores, const float score_threshold, const float nms_threshold, std::vector<int> *indices, const float eta, const int top_k) { BEGIN_WRAP cv::dnn::NMSBoxes(*bboxes, *scores, score_threshold, nms_threshold, *indices, eta, top_k); END_WRAP }	cv::dnn::NMSBoxes()	group__dnn.html	Performs non maximum suppression given boxes and corresponding scores.	bboxes : a set of bounding boxes to apply NMS. scores : a set of corresponding confidences. score_threshold : a threshold used to filter boxes by score. nms_threshold : a threshold used in non maximum suppression. indices : the kept indices of bboxes after NMS. eta : a coefficient in adaptive threshold formula: $$nms_threshold_{i+1}=eta\cdot nms_threshold_i$$. top_k : if >0, keep at most top_k picked indices.	Examples: samples/dnn/object_detection.cpp.	7								
dnn_NMSBoxes_Rect2d	CVAPI(ExceptionStatus) dnn_NMSBoxes_Rect2d(std::v ector<cv::Rect2d> *bboxes, std::vector<float> *scores, const float score_threshold, const float nms_threshold, std::vector<int> *indices, const float eta, const int top_k) { BEGIN_WRAP cv::dnn::NMSBoxes(*bboxes, *scores, score_threshold, nms_threshold, *indices, eta, top_k); END_WRAP }	cv::dnn::NMSBoxes()	group__dnn.html	Performs non maximum suppression given boxes and corresponding scores.	bboxes : a set of bounding boxes to apply NMS. scores : a set of corresponding confidences. score_threshold : a threshold used to filter boxes by score. nms_threshold : a threshold used in non maximum suppression. indices : the kept indices of bboxes after NMS. eta : a coefficient in adaptive threshold formula: $$nms_threshold_{i+1}=eta\cdot nms_threshold_i$$. top_k : if >0, keep at most top_k picked indices.	Examples: samples/dnn/object_detection.cpp.	7								
dnn_NMSBoxes_RotatedRect	CVAPI(ExceptionStatus) dnn_NMSBoxes_RotatedRect(std::vector<cv::RotatedRect > *bboxes, std::vector<float> *scores, const float score_threshold, const float nms_threshold, std::vector<int> *indices, const float eta, const int top_k) { BEGIN_WRAP cv::dnn::NMSBoxes(*bboxes, *scores, score_threshold, nms_threshold, *indices, eta, top_k); END_WRAP }	cv::dnn::NMSBoxes()	group__dnn.html	Performs non maximum suppression given boxes and corresponding scores.	bboxes : a set of bounding boxes to apply NMS. scores : a set of corresponding confidences. score_threshold : a threshold used to filter boxes by score. nms_threshold : a threshold used in non maximum suppression. indices : the kept indices of bboxes after NMS. eta : a coefficient in adaptive threshold formula: $$nms_threshold_{i+1}=eta\cdot nms_threshold_i$$. top_k : if >0, keep at most top_k picked indices.	Examples: samples/dnn/object_detection.cpp.	7								
dnn_resetMyriadDevice	CVAPI(ExceptionStatus) dnn_resetMyriadDevice() { BEGIN_WRAP cv::dnn::resetMyriadDevice(); END_WRAP }	cv::dnn::resetMyriadDevice()	namespacecv_1_1dnn.html	Release a Myriad device (binded by OpenCV).		Single Myriad device cannot be shared across multiple processes which uses Inference Engine's Myriad plugin.	1									
dnn_Net_new	CVAPI(ExceptionStatus) dnn_Net_new(cv::dnn::Net **returnValue) { BEGIN_WRAP *returnValue = new cv::dnn::Net; END_WRAP }	cv::dnn::Net	classcv_1_1dnn_1_1Net.html	This class allows to create and manipulate comprehensive artificial neural networks. Neural network is presented as directed acyclic graph (DAG), where vertices are Layer instances, and edges specify relationships between layers inputs and outputs. Each network layer has unique integer id and unique string name inside its network. LayerId can store either layer name or layer id. This class supports reference counting of its instances, i. e. copies point to the same instance. Examples: samples/dnn/colorization.cpp, and samples/dnn/openpose.cpp.			1						•			

dnn_Net_delete	CVAPI(ExceptionStatus) dnn_Net_delete(cv::dnn::Net* net) { BEGIN_WRAP delete net; END_WRAP }	cv::dnn::Net	classcv_1_1dnn_1_1Net.html	This class allows to create and manipulate comprehensive artificial neural networks. Neural network is presented as directed acyclic graph (DAG), where vertices are Layer instances, and edges specify relationships between layers inputs and outputs. Each network layer has unique integer id and unique string name inside its network. LayerId can store either layer name or layer id. This class supports reference counting of its instances, i. e. copies point to the same instance. Examples: samples/dnn/colorization.cpp, and samples/dnn/openpose.cpp.									•		
dnn_Net_readFromModelOptimizer	CVAPI(ExceptionStatus) dnn_Net_readFromModelOptimizer(const char *xml, const char *bin, cv::dnn::Net **returnValue) { BEGIN_WRAP const auto net = cv::dnn::Net::readFromModelOptimizer(xml, bin); *returnValue = new cv::dnn::Net(net); END_WRAP }	cv::dnn::Net::readFromModelOptimizer()	classcv_1_1dnn_1_1Net.html	Create a network from Intel's Model Optimizer intermediate representation (IR).	xml : [in] bin : [in]		2									
dnn_Net_empty	CVAPI(ExceptionStatus) dnn_Net_empty(cv::dnn::Net* net, int *returnValue) { BEGIN_WRAP *returnValue = net->empty() ? 1 : 0; END_WRAP }	cv::dnn::Net::empty()	classcv_1_1dnn_1_1Net.html	Returns true if there are no layers in the network.			1									
dnn_Net_dump	CVAPI(ExceptionStatus) dnn_Net_dump(cv::dnn::Net* net, std::string *outString) { BEGIN_WRAP outString->assign(net->dump()); END_WRAP }	cv::dnn::Net::dump()	classcv_1_1dnn_1_1Net.html	Dump net to String.			1									
dnn_Net_dumpToFile	CVAPI(ExceptionStatus) dnn_Net_dumpToFile(cv::dnn::Net* net, const char *path) { BEGIN_WRAP net->dumpToFile(path); END_WRAP }	cv::dnn::Net::dumpToFile()	classcv_1_1dnn_1_1Net.html	Dump net structure, hyperparameters, backend, target and fusion to dot file.	path : path to output file with .dot extension	See alsodump()	1									
dnn_Net_getLayerId	CVAPI(ExceptionStatus) dnn_Net_getLayerId(cv::dnn::Net* net, const char *layer, int *returnValue) { BEGIN_WRAP *returnValue = net->getLayerId(layer); END_WRAP }	cv::dnn::Net::getLayerId()	classcv_1_1dnn_1_1Net.html	Converts string name of the layer to the integer identifier.			1									
dnn_Net_getLayerNames	CVAPI(ExceptionStatus) dnn_Net_getLayerNames(cv::dnn::Net* net, std::vector<cv::String> *outVec) { BEGIN_WRAP const auto result = net->getLayerNames(); outVec->assign(result.begin(), result.end()); END_WRAP }	cv::dnn::Net::getLayerNames()	classcv_1_1dnn_1_1Net.html				2									
dnn_Net_connect1	CVAPI(ExceptionStatus) dnn_Net_connect1(cv::dnn::Net* net, const char *outPin, const char *inpPin) { BEGIN_WRAP net->connect(outPin, inpPin); END_WRAP }	cv::dnn::Net::connect()	classcv_1_1dnn_1_1Net.html	Connects output of the first layer to input of the second layer.	outPin : descriptor of the first layer output. inpPin : descriptor of the second layer input.	Descriptors have the following template <layer_name>[.input_number]:the first part of the template layer_name is string name of the added layer. If this part is empty then the network input pseudo layer will be used; the second optional part of the template input_number is either number of the layer input, either label one. If this part is omitted then the first layer input will be used. See alssetNetInputs(), Layer::inputNameToIndex(), Layer::outputNameToIndex()	2									
dnn_Net_connect2	CVAPI(ExceptionStatus) dnn_Net_connect2(cv::dnn::Net* net, int outLayerId, int outNum, int inpLayerId, int inpNum) { BEGIN_WRAP net->connect(outLayerId, outNum, inpLayerId, inpNum); END_WRAP }	cv::dnn::Net::connect()	classcv_1_1dnn_1_1Net.html	Connects #outNum output of the first layer to #inNum input of the second layer.	outLayerId : identifier of the first layer outNum : number of the first layer output inpLayerId : identifier of the second layer inpNum : number of the second layer input		4									
dnn_Net_setInputsNames	CVAPI(ExceptionStatus) dnn_Net_setInputsNames(cv::dnn::Net* net, const char **inputBlobNames, int inputBlobNamesLength) { BEGIN_WRAP std::vector<cv::String> inputBlobNamesVec(inputBlobNamesLength); for (auto i = 0; i < inputBlobNamesLength; i++) { inputBlobNamesVec[i] = inputBlobNames[i]; } net->setInputsNames(inputBlobNamesVec); END_WRAP }	cv::dnn::Net::setInputsNames()	classcv_1_1dnn_1_1Net.html	Sets outputs names of the network input pseudo layer.		Each net always has special own the network input pseudo layer with id=0. This layer stores the user blobs only and don't make any computations. In fact, this layer provides the only way to pass user data into the network. As any other layer, this layer can label its outputs and this function provides an easy way to do this.	1									
dnn_Net_forward1	CVAPI(ExceptionStatus) dnn_Net_forward1(cv::dnn::Net* net, const char *outputName, cv::Mat **returnValue) { BEGIN_WRAP const auto outputNameStr = (outputName == nullptr) ? cv::String() : cv::String(outputName); const auto net = net->forward(outputNameStr); *returnValue = new cv::Mat(net); END_WRAP }	cv::dnn::Net::forward()	classcv_1_1dnn_1_1Net.html	Runs forward pass to compute output of layer with name outputName.	outputName : name for layer which output is needed to get	By default runs forward pass for the whole network.Examples: samples/dnn/colorization.cpp, and samples/dnn/openpose.cpp.	1									
dnn_Net_forward2	CVAPI(ExceptionStatus) dnn_Net_forward2(cv::dnn::Net* net, cv::Mat **outputBlobs, int outputBlobsLength, const char *outputName) { BEGIN_WRAP const auto outputNameStr = (outputName == nullptr) ? cv::String() : cv::String(outputName); std::vector<cv::Mat> outputBlobsVec; toVec(outputBlobs, outputBlobsLength, outputBlobsVec); net->forward(outputBlobsVec, outputNameStr); for (auto i = 0; i < outputBlobsLength; i++) { *outputBlobs[i] =	cv::dnn::Net::forward()	classcv_1_1dnn_1_1Net.html	Runs forward pass to compute output of layer with name outputName.	outputBlobs : contains all output blobs for specified layer. outputName : name for layer which output is needed to get	If outputName is empty, runs forward pass for the whole network.	2									
dnn_Net_forward3	CVAPI(ExceptionStatus) dnn_Net_forward3(cv::dnn::Net* net, cv::Mat **outputBlobs, int outputBlobsLength, const char **outBlobNames, int outBlobNamesLength) { BEGIN_WRAP std::vector<cv::Mat> outputBlobsVec; toVec(outputBlobs, outputBlobsLength, outputBlobsVec); std::vector<cv::String> outBlobNamesVec(outBlobNamesLength); for (auto i = 0; i < outBlobNamesLength; i++) { outBlobNamesVec[i] = outBlobNames[i]; } net->forward(outputBlobsVec,	cv::dnn::Net::forward()	classcv_1_1dnn_1_1Net.html	Runs forward pass to compute output of layer with name outputName.	outputBlobs : contains all output blobs for specified layer. outputName : name for layer which output is needed to get	If outputName is empty, runs forward pass for the whole network.	2									

dnn_Net_setHalideScheduler	CVAPI(ExceptionStatus) dnn_Net_setHalideScheduler(cv::dnn::Net* net, const char *scheduler) { BEGIN_WRAP net->setHalideScheduler(scheduler); END_WRAP }	cv::dnn::Net::setHalideScheduler()	classcv_1_1dnn_1_1Net.html	Compile Halide layers.	scheduler : [in]	See alsosetPreferableBackendSchedule layers that support Halide backend. Then compile them for specific target. For layers that not represented in scheduling file or if no manual scheduling used at all, automatic scheduling will be applied.	1									
dnn_Net_setPreferableBackend	CVAPI(ExceptionStatus) dnn_Net_setPreferableBackend(cv::dnn::Net* net, int backendId) { BEGIN_WRAP net->setPreferableBackend(backendId); END_WRAP }	cv::dnn::Net::setPreferableBackend()	classcv_1_1dnn_1_1Net.html	Ask network to use specific computation backend where it supported.	backendId : [in]	See alsoBackendId OpenCV is compiled with Intal's Inference Engine library, DNN_BACKEND_DEFAULT means DNN_BACKEND_INFERENCE_ENGINE. Otherwise it equals to DNN_BACKEND_OPENCV.	1									
dnn_Net_setPreferableTarget	CVAPI(ExceptionStatus) dnn_Net_setPreferableTarget(cv::dnn::Net* net, int targetId) { BEGIN_WRAP net->setPreferableTarget(targetId); END_WRAP }	cv::dnn::Net::setPreferableTarget()	classcv_1_1dnn_1_1Net.html	Ask network to make computations on specific target device.	targetId : [in]	See alsoTargetList of supported combinations backend / target:DNN_BACKEND_OPENCV DNN_BACKEND_INFERENCE_ENGINE DNN_BACKEND_HALIDE DNN_BACKEND_CUDA DNN_TARGET_CPU + + + DNN_TARGET_OPENCL + + + DNN_TARGET_OPENCL_FP16 + + DNN_TARGET_MYRIAD + DNN_TARGET_FPGA + DNN_TARGET_CUDA + DNN_TARGET_CUDA_FP16 + DNN_TARGET_HDDL +Examples: samples/dnn/colorization.cpp	1									
dnn_Net_setInput	CVAPI(ExceptionStatus) dnn_Net_setInputsNames(cv::dnn::Net* net, const char **inputBlobNames, int inputBlobNamesLength) { BEGIN_WRAP std::vector<cv::String> inputBlobNamesVec(inputBlobNamesLength); for (auto i = 0; i < inputBlobNamesLength; i++) { inputBlobNamesVec[i] = inputBlobNames[i]; } net->setInputsNames(inputBlobNamesVec); END_WRAP }	cv::dnn::Net::setInput()	classcv_1_1dnn_1_1Net.html	Sets the new input value for the network.	blob : A new blob. Should have CV_32F or CV_8U depth. name : A name of input layer. scalefactor : An optional normalization scale. mean : An optional mean subtraction values.	See alsoconnect(String, String) to know format of the descriptor If scale or mean values are specified, a final input blob is computed as:$$input(n,c,h,w) = scalefactor * (blob(n,c,h,w) - mean_c)$$Examples: samples/dnn/colorization.cpp, and samples/dnn/openpose.cpp.	1									
dnn_Net_getUnconnectedOutLayers	CVAPI(ExceptionStatus) dnn_Net_getUnconnectedOutLayers(cv::dnn::Net* net, std::vector<int>* result) { BEGIN_WRAP const auto v = net->getUnconnectedOutLayers(); result->clear(); result->resize(v.size()); for (size_t i = 0; i < v.size(); i++) { result->at(i) = v[i]; } END_WRAP }	cv::dnn::Net::getUnconnectedOutLayers()	classcv_1_1dnn_1_1Net.html	Returns indexes of layers with unconnected outputs.			1									
dnn_Net_getUnconnectedOutLayersNames	CVAPI(ExceptionStatus) dnn_Net_getUnconnectedOutLayersNames(cv::dnn::Net* net, std::vector<std::string>* result) { BEGIN_WRAP const auto v = net->getUnconnectedOutLayersNames(); result->clear(); result->resize(v.size()); for (size_t i = 0; i < v.size(); i++) { result->at(i) = v[i]; } END_WRAP }	cv::dnn::Net::getUnconnectedOutLayersNames()	classcv_1_1dnn_1_1Net.html	Returns names of layers with unconnected outputs.			1									
dnn_Net_enableFusion	CVAPI(ExceptionStatus) dnn_Net_enableFusion(cv::dnn::Net* net, int fusion) { BEGIN_WRAP net->enableFusion(fusion != 0); END_WRAP }	cv::dnn::Net::enableFusion()	classcv_1_1dnn_1_1Net.html	Enables or disables layer fusion in the network.	fusion : true to enable the fusion, false to disable. The fusion is enabled by default.		1									
dnn_Net_getPerfProfile	CVAPI(ExceptionStatus) dnn_Net_getPerfProfile(cv::dnn::Net* net, std::vector<double> *timings, int64 *returnValue) { BEGIN_WRAP *returnValue = net->getPerfProfile(*timings); END_WRAP }	cv::dnn::Net::getPerfProfile()	classcv_1_1dnn_1_1Net.html	Returns overall time for inference and timings (in ticks) for layers.	timings : [out]	Indexes in returned vector correspond to layers ids. Some layers can be fused with others, in this case zero ticks count will be return for that skipped layers. Supported by DNN_BACKEND_OPENCV on DNN_TARGET_CPU only.	1									
face_Facemark_loadModel	CVAPI(ExceptionStatus) face_Facemark_loadModel(cv::face::Facemark *obj, const char *model) { BEGIN_WRAP obj->loadModel(model); END_WRAP }	cv::face::Facemark::loadModel()	classcv_1_1face_1_1Facemark.html	A function to load the trained model before the fitting process.	model : A string represent the filename of a trained model.	Example of usagefacemark->loadModel("../data/lbf.model");fragment	1									
face_Facemark_fit		cv::face::Facemark::fit()	classcv_1_1face_1_1Facemark.html	Detect facial landmarks from an image.	Image : Input image. faces : Output of the function which represent region of interest of the detected faces. Each face is stored in cv::Rect container. landmarks : The detected landmark points for each faces.	Example of usageMat image = imread("image.jpg");std::vector<Rect> faces;std::vector<std::vector<Point2f> > landmarks;facemark->fit(image, faces, landmarks);fragment	-1									
face_FacemarkLBF_create	CVAPI(ExceptionStatus) face_FacemarkLBF_create(cv::face::FacemarkLBF::Params *params, cv::Ptr<cv::face::FacemarkLBF> **returnValue) { BEGIN_WRAP const auto obj = (params == nullptr) ? cv::face::FacemarkLBF::create() : cv::face::FacemarkLBF::create(*params); *returnValue = clone(obj); END_WRAP }	cv::face::FacemarkLBF::create()	classcv_1_1face_1_1FacemarkLBF.html				1									
face_Ptr_FacemarkLBF_get	CVAPI(ExceptionStatus) face_Ptr_FacemarkLBF_get(cv::Ptr<cv::face::FacemarkLBF> *obj, cv::face::FacemarkLBF **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::face::FacemarkLBF											•		
face_Ptr_FacemarkLBF_delete	CVAPI(ExceptionStatus) face_Ptr_FacemarkLBF_delete(cv::Ptr<cv::face::FacemarkLBF> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::FacemarkLBF												•		
face_FacemarkLBF_Params_new	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_new(cv::face::FacemarkLBF::Params **returnValue) { BEGIN_WRAP *returnValue = new cv::face::FacemarkLBF::Params; END_WRAP }	cv::face::FacemarkLBF::Params												•		
face_FacemarkLBF_Params_delete	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_delete(cv::face::FacemarkLBF::Params *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::FacemarkLBF::Params												•		

face_FacemarkLBF_Params_shape_offset_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_shape_offset_get(cv::face::FacemarkLBF::Params *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->shape_offset; END_WRAP }	cv::face::FacemarkLBF::Params::shape_offset	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	offset for the loaded face landmark points						•					
face_FacemarkLBF_Params_shape_offset_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_shape_offset_set(cv::face::FacemarkLBF::Params *obj, double val) { BEGIN_WRAP obj->shape_offset = val; END_WRAP }	cv::face::FacemarkLBF::Params::shape_offset	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	offset for the loaded face landmark points							•				
face_FacemarkLBF_Params_cascade_face_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_cascade_face_get(cv::face::FacemarkLBF::Params *obj, std::string *s) { BEGIN_WRAP s->assign(obj->cascade_face); END_WRAP }	cv::face::FacemarkLBF::Params::cascade_face	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	filename of the face detector model						•					
face_FacemarkLBF_Params_cascade_face_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_cascade_face_set(cv::face::FacemarkLBF::Params *obj, const char *s) { BEGIN_WRAP obj->cascade_face = s; END_WRAP }	cv::face::FacemarkLBF::Params::cascade_face	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	filename of the face detector model							•				
face_FacemarkLBF_Params_verbose_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_verbose_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->verbose ? 1 : 0; END_WRAP }	cv::face::FacemarkLBF::Params::verbose	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	show the training print-out						•					
face_FacemarkLBF_Params_verbose_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_verbose_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->verbose = (val != 0); END_WRAP }	cv::face::FacemarkLBF::Params::verbose	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	show the training print-out							•				
face_FacemarkLBF_Params_n_landmarks_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_n_landmarks_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->n_landmarks; END_WRAP }	cv::face::FacemarkLBF::Params::n_landmarks	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of landmark points						•					
face_FacemarkLBF_Params_n_landmarks_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_n_landmarks_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->n_landmarks = val; END_WRAP }	cv::face::FacemarkLBF::Params::n_landmarks	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of landmark points							•				
face_FacemarkLBF_Params_initShape_n_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_initShape_n_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->initShape_n; END_WRAP }	cv::face::FacemarkLBF::Params::initShape_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	multiplier for augment the training data						•					
face_FacemarkLBF_Params_initShape_n_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_initShape_n_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->initShape_n = val; END_WRAP }	cv::face::FacemarkLBF::Params::initShape_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	multiplier for augment the training data							•				
face_FacemarkLBF_Params_stages_n_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_stages_n_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->stages_n; END_WRAP }	cv::face::FacemarkLBF::Params::stages_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of refinement stages						•					
face_FacemarkLBF_Params_stages_n_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_stages_n_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->stages_n = val; END_WRAP }	cv::face::FacemarkLBF::Params::stages_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of refinement stages							•				
face_FacemarkLBF_Params_tree_n_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_tree_n_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->tree_n; END_WRAP }	cv::face::FacemarkLBF::Params::tree_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of tree in the model for each landmark point refinement						•					
face_FacemarkLBF_Params_tree_n_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_tree_n_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->tree_n = val; END_WRAP }	cv::face::FacemarkLBF::Params::tree_n	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	number of tree in the model for each landmark point refinement							•				
face_FacemarkLBF_Params_tree_depth_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_tree_depth_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->tree_depth; END_WRAP }	cv::face::FacemarkLBF::Params::tree_depth	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	the depth of decision tree, defines the size of feature						•					
face_FacemarkLBF_Params_tree_depth_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_tree_depth_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->tree_depth = val; END_WRAP }	cv::face::FacemarkLBF::Params::tree_depth	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	the depth of decision tree, defines the size of feature							•				
face_FacemarkLBF_Params_bagging_overlap_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_bagging_overlap_get(cv::face::FacemarkLBF::Params *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->bagging_overlap; END_WRAP }	cv::face::FacemarkLBF::Params::bagging_overlap	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	overlap ratio for training the LBF feature						•					
face_FacemarkLBF_Params_bagging_overlap_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_bagging_overlap_set(cv::face::FacemarkLBF::Params *obj, double val) { BEGIN_WRAP obj->bagging_overlap = val; END_WRAP }	cv::face::FacemarkLBF::Params::bagging_overlap	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	overlap ratio for training the LBF feature							•				

face_FacemarkLBF_Params_model_filename_g	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_model_filename_get(cv::face::FacemarkLBF::Params *obj, std::string *s) { BEGIN_WRAP s->assign(obj->model_filename); END_WRAP }	cv::face::FacemarkLBF::Params::model_filename	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	filename where the trained model will be saved							•					
face_FacemarkLBF_Params_model_filename_s	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_model_filename_set(cv::face::FacemarkLBF::Params *obj, const char *s) { BEGIN_WRAP obj->model_filename = s; END_WRAP }	cv::face::FacemarkLBF::Params::model_filename	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	filename where the trained model will be saved								•				
face_FacemarkLBF_Params_save_model_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_save_model_get(cv::face::FacemarkLBF::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->save_model ? 1 : 0; END_WRAP }	cv::face::FacemarkLBF::Params::save_model	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	flag to save the trained model or not							•					
face_FacemarkLBF_Params_save_model_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_save_model_set(cv::face::FacemarkLBF::Params *obj, int val) { BEGIN_WRAP obj->save_model = (val != 0); END_WRAP }	cv::face::FacemarkLBF::Params::save_model	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	flag to save the trained model or not								•				
face_FacemarkLBF_Params_seed_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_seed_get(cv::face::FacemarkLBF::Params *obj, unsigned int *returnValue) { BEGIN_WRAP *returnValue = obj->seed; END_WRAP }	cv::face::FacemarkLBF::Params::seed	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	seed for shuffling the training data							•					
face_FacemarkLBF_Params_seed_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_seed_set(cv::face::FacemarkLBF::Params *obj, unsigned int val) { BEGIN_WRAP obj->seed = val; END_WRAP }	cv::face::FacemarkLBF::Params::seed	structcv_1_1face_1_1FacemarkLBF_1_1Params.html	seed for shuffling the training data								•				
face_FacemarkLBF_Params_feats_m_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_feats_m_get(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP std::copy(obj->feats_m.begin(), obj->feats_m.end(), std::back_inserter(*v)); END_WRAP }	cv::face::FacemarkLBF::Params::feats_m	structcv_1_1face_1_1FacemarkLBF_1_1Params.html								•					
face_FacemarkLBF_Params_feats_m_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_feats_m_set(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP obj->feats_m.clear(); std::copy(v->begin(), v->end(), std::back_inserter(obj->feats_m)); END_WRAP }	cv::face::FacemarkLBF::Params::feats_m	structcv_1_1face_1_1FacemarkLBF_1_1Params.html									•				
face_FacemarkLBF_Params_radius_m_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_radius_m_get(cv::face::FacemarkLBF::Params *obj, std::vector<double> *v) { BEGIN_WRAP std::copy(obj->radius_m.begin(), obj->radius_m.end(), std::back_inserter(*v)); END_WRAP }	cv::face::FacemarkLBF::Params::radius_m	structcv_1_1face_1_1FacemarkLBF_1_1Params.html								•					
face_FacemarkLBF_Params_radius_m_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_radius_m_set(cv::face::FacemarkLBF::Params *obj, std::vector<double> *v) { BEGIN_WRAP obj->radius_m.clear(); std::copy(v->begin(), v->end(), std::back_inserter(obj->radius_m)); END_WRAP }	cv::face::FacemarkLBF::Params::radius_m	structcv_1_1face_1_1FacemarkLBF_1_1Params.html									•				
face_FacemarkLBF_Params_pupils0_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_pupils0_get(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP std::copy(obj->pupils[0].begin(), obj->pupils[0].end(), std::back_inserter(*v)); END_WRAP }	cv::face::FacemarkLBF::Params::pupils									•					
face_FacemarkLBF_Params_pupils0_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_pupils0_set(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP obj->pupils[0].clear(); std::copy(v->begin(), v->end(), std::back_inserter(obj->pupils[0])); END_WRAP }	cv::face::FacemarkLBF::Params::pupils										•				
face_FacemarkLBF_Params_pupils1_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_pupils1_get(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP std::copy(obj->pupils[1].begin(), obj->pupils[1].end(), std::back_inserter(*v)); END_WRAP }	cv::face::FacemarkLBF::Params::pupils									•					
face_FacemarkLBF_Params_pupils1_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_pupils1_set(cv::face::FacemarkLBF::Params *obj, std::vector<int> *v) { BEGIN_WRAP obj->pupils[1].clear(); std::copy(v->begin(), v->end(), std::back_inserter(obj->pupils[1])); END_WRAP }	cv::face::FacemarkLBF::Params::pupils										•				
face_FacemarkLBF_Params_detectROI_get	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_detectROI_get(cv::face::FacemarkLBF::Params *obj, MyCvRect *returnValue) { BEGIN_WRAP *returnValue = c(obj->detectROI); END_WRAP }	cv::face::FacemarkLBF::Params::detectROI	structcv_1_1face_1_1FacemarkLBF_1_1Params.html								•					
face_FacemarkLBF_Params_detectROI_set	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_detectROI_set(cv::face::FacemarkLBF::Params *obj, MyCvRect val) { BEGIN_WRAP obj->detectROI = cpp(val); END_WRAP }	cv::face::FacemarkLBF::Params::detectROI	structcv_1_1face_1_1FacemarkLBF_1_1Params.html									•				

face_FacemarkLBF_Params_read	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_read(cv::face::FacemarkLBF::Params *obj, cv::FileNode *fn) { BEGIN_WRAP obj->read(*fn); END_WRAP }	cv::face::FacemarkLBF::Params::read()	structcv_1_1face_1_1FacemarkLBF_1_1Params.html				1									
face_FacemarkLBF_Params_write	CVAPI(ExceptionStatus) face_FacemarkLBF_Params_write(cv::face::FacemarkLBF::Params *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->write(*fs); END_WRAP }	cv::face::FacemarkLBF::Params::write()	structcv_1_1face_1_1FacemarkLBF_1_1Params.html				1									
face_FacemarkAAM_create	CVAPI(ExceptionStatus) face_FacemarkAAM_create(cv::face::FacemarkAAM::Params *params, cv::Ptr<cv::face::FacemarkAAM> **returnValue) { BEGIN_WRAP const auto obj = (params == nullptr) ? cv::face::FacemarkAAM::create() : cv::face::FacemarkAAM::create(*params); *returnValue = clone(obj); END_WRAP }	cv::face::FacemarkAAM::create()	classcv_1_1face_1_1FacemarkAAM.html	initializer			1									
face_Ptr_FacemarkAAM_get	CVAPI(ExceptionStatus) face_Ptr_FacemarkAAM_get(cv::Ptr<cv::face::FacemarkAAM> *obj, cv::face::FacemarkAAM **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::face::FacemarkAAM												•		
face_Ptr_FacemarkAAM_delete	CVAPI(ExceptionStatus) face_Ptr_FacemarkAAM_delete(cv::Ptr<cv::face::FacemarkAAM> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::FacemarkAAM										•				
face_FacemarkAAM_Params_new	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_new(cv::face::FacemarkAAM::Params **returnValue) { BEGIN_WRAP *returnValue = new cv::face::FacemarkAAM::Params; END_WRAP }	cv::face::FacemarkAAM::Params									•					
face_FacemarkAAM_Params_delete	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_delete(cv::face::FacemarkAAM::Params *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::FacemarkAAM::Params										•				
face_FacemarkAAM_Params_model_filename_	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_model_filename_get(cv::face::FacemarkAAM::Params *obj, std::string *s) { BEGIN_WRAP s->assign(obj->model_filename); END_WRAP }	cv::face::FacemarkAAM::Params::model_filename	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						
face_FacemarkAAM_Params_model_filename_	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_model_filename_set(cv::face::FacemarkAAM::Params *obj, const char *s) { BEGIN_WRAP obj->model_filename = s; END_WRAP }	cv::face::FacemarkAAM::Params::model_filename	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_m_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_m_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->m; END_WRAP }	cv::face::FacemarkAAM::Params::m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						
face_FacemarkAAM_Params_m_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_m_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->m = val; END_WRAP }	cv::face::FacemarkAAM::Params::m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_n_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_n_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->n; END_WRAP }	cv::face::FacemarkAAM::Params::n	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						
face_FacemarkAAM_Params_n_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_n_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->n = val; END_WRAP }	cv::face::FacemarkAAM::Params::n	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_n_iter_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_n_iter_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->n_iter; END_WRAP }	cv::face::FacemarkAAM::Params::n_iter	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						
face_FacemarkAAM_Params_n_iter_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_n_iter_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->n_iter = val; END_WRAP }	cv::face::FacemarkAAM::Params::n_iter	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_verbose_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_verbose_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->verbose ? 1 : 0; END_WRAP }	cv::face::FacemarkAAM::Params::verbose	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						
face_FacemarkAAM_Params_verbose_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_verbose_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->verbose = (val != 0); END_WRAP }	cv::face::FacemarkAAM::Params::verbose	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_save_model_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_save_model_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->save_model ? 1 : 0; END_WRAP }	cv::face::FacemarkAAM::Params::save_model	structcv_1_1face_1_1FacemarkAAM_1_1Params.html							•						

face_FacemarkAAM_Params_save_model_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_save_model_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->save_model = (val != 0); END_WRAP }	cv::face::FacemarkAAM::Params::save_model	structcv_1_1face_1_1FacemarkAAM_1_1Params.html									•				
face_FacemarkAAM_Params_max_m_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_max_m_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->max_m; END_WRAP }	cv::face::FacemarkAAM::Params::max_m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_max_m_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_max_m_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->max_m = val; END_WRAP }	cv::face::FacemarkAAM::Params::max_m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html									•				
face_FacemarkAAM_Params_max_n_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_max_n_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->max_n; END_WRAP }	cv::face::FacemarkAAM::Params::max_n	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_max_n_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_max_n_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->max_n = val; END_WRAP }	cv::face::FacemarkAAM::Params::max_n	structcv_1_1face_1_1FacemarkAAM_1_1Params.html									•				
face_FacemarkAAM_Params_texture_max_m_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_texture_max_m_get(cv::face::FacemarkAAM::Params *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->texture_max_m; END_WRAP }	cv::face::FacemarkAAM::Params::texture_max_m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_texture_max_m_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_texture_max_m_set(cv::face::FacemarkAAM::Params *obj, int val) { BEGIN_WRAP obj->texture_max_m = val; END_WRAP }	cv::face::FacemarkAAM::Params::texture_max_m	structcv_1_1face_1_1FacemarkAAM_1_1Params.html									•				
face_FacemarkAAM_Params_scales_get	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_scales_get(cv::face::FacemarkAAM::Params *obj, std::vector<float> *v) { BEGIN_WRAP std::copy(obj->scales.begin(), obj->scales.end(), std::back_inserter(*v)); END_WRAP }	cv::face::FacemarkAAM::Params::scales	structcv_1_1face_1_1FacemarkAAM_1_1Params.html								•					
face_FacemarkAAM_Params_scales_set	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_scales_set(cv::face::FacemarkAAM::Params *obj, std::vector<float> *v) { BEGIN_WRAP obj->scales.clear(); std::copy(v->begin(), v->end(), std::back_inserter(obj->scales)); END_WRAP }	cv::face::FacemarkAAM::Params::scales	structcv_1_1face_1_1FacemarkAAM_1_1Params.html									•				
face_FacemarkAAM_Params_read	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_read(cv::face::FacemarkAAM::Params *obj, cv::FileNode *fn) { BEGIN_WRAP obj->read(*fn); END_WRAP }	cv::face::FacemarkAAM::Params::read()	structcv_1_1face_1_1FacemarkAAM_1_1Params.html	Read parameters from file, currently unused.				1								
face_FacemarkAAM_Params_write	CVAPI(ExceptionStatus) face_FacemarkAAM_Params_write(cv::face::FacemarkAAM::Params *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->write(*fs); END_WRAP }	cv::face::FacemarkAAM::Params::write()	structcv_1_1face_1_1FacemarkAAM_1_1Params.html	Read parameters from file, currently unused.				1								
face_FaceRecognizer_train	CVAPI(ExceptionStatus) face_FaceRecognizer_train(cv::face::FaceRecognizer *obj, cv::Mat **src, int srcLength, int *labels, int labelsLength) { BEGIN_WRAP std::vector<cv::Mat> srcVec(srcLength); for (auto i = 0; i < srcLength; i++) srcVec[i] = *src[i]; const std::vector<int> labelsVec(labels, labels + labelsLength); obj->train(srcVec, labelsVec); END_WRAP }	cv::face::FaceRecognizer::train()	classcv_1_1face_1_1FaceRecognizer.html	Trains a FaceRecognizer with given data and associated labels.	src : The training images, that means the faces you want to learn. The data has to be given as a vector<Mat>. labels : The labels corresponding to the images have to be given either as a vector<int> or a Mat of type CV_32SC1.	The following source code snippet shows you how to learn a Fisherfaces model on a given set of images. The images are read with imread and pushed into a std::vector<Mat>. The labels of each image are stored within a std::vector<int> (you could also use a Mat of type CV_32SC1). Think of the label as the subject (the person) this image belongs to, so same subjects (persons) should have the same label. For the available FaceRecognizer you don't have to pay any attention to the order of the labels, just make sure same persons have the same label:// holds images and labelsvector<Mat> images;vector<int> labels;// using Mat of type CV_32SC1// Mat labels(number_of_samples, 1, CV_32SC1);// images for first personimages.push_back(imread("person0/0.jpg", IMREAD_GRAYSCALE));labels.push_back(0);images.push_back(imread("person0/1.jpg", IMREAD_GRAYSCALE));labels.push_back(0);images.push_back(imread("person0/2.jpg", IMREAD_GRAYSCALE)	2								
face_FaceRecognizer_update	CVAPI(ExceptionStatus) face_FaceRecognizer_update(cv::face::FaceRecognizer *obj, cv::Mat **src, int srcLength, int *labels, int labelsLength) { BEGIN_WRAP std::vector<cv::Mat> srcVec(srcLength); for (auto i = 0; i < srcLength; i++) srcVec[i] = *src[i]; const std::vector<int> labelsVec(labels, labels + labelsLength); obj->update(srcVec, labelsVec); END_WRAP }	cv::face::FaceRecognizer::update()	classcv_1_1face_1_1FaceRecognizer.html	Updates a FaceRecognizer with given data and associated labels.	src : The training images, that means the faces you want to learn. The data has to be given as a vector<Mat>. labels : The labels corresponding to the images have to be given either as a vector<int> or a Mat of type CV_32SC1.	This method updates a (probably trained) FaceRecognizer, but only if the algorithm supports it. The Local Binary Patterns Histograms (LBPH) recognizer (see createLBPHFaceRecognizer) can be updated. For the Eigenfaces and Fisherfaces method, this is algorithmically not possible and you have to re-estimate the model with FaceRecognizer::train. In any case, a call to train empties the existing model and learns a new model, while update does not delete any model data.// Create a new LBPH model (it can be updated) and use the default parameters.// this is the most common usage of this specific FaceRecognizer://Ptr<FaceRecognizer> model = LBPHFaceRecognizer::create();// This is the common interface to train all of the available cv::FaceRecognizer// implementations://model->train(images, labels);// Some containers to hold new image:vector<Mat> newImages;vector<int> newLabels;// You should add some images to the	2								
face_FaceRecognizer_predict1	CVAPI(ExceptionStatus) face_FaceRecognizer_predict1(cv::face::FaceRecognizer *obj, cv::_InputArray *src, int *returnValue) { BEGIN_WRAP *returnValue = obj->predict(*src); END_WRAP }	cv::face::FaceRecognizer::predict()	classcv_1_1face_1_1FaceRecognizer.html	Predicts a label and associated confidence (e.g. distance) for a given input image.	src : Sample image to get a prediction from. label : The predicted label for the given image. confidence : Associated confidence (e.g. distance) for the predicted label.	The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.The following example shows how to get a prediction from a trained model:using namespace cv;// Do your initialization here (create the cv::FaceRecognizer model) ...// ...// Read in a sample image:Mat img = imread("person1/3.jpg", IMREAD_GRAYSCALE);// And get a prediction from the cv::FaceRecognizer:int predicted = model->predict(img);fragmentDr to get a prediction and the associated confidence (e.g. distance):using namespace cv;// Do your initialization here (create the cv::FaceRecognizer model) ...// ...Mat img = imread("person1/3.jpg", IMREAD_GRAYSCALE);// Some variables for the predicted label and associated confidence (e.g. distance):int predicted_label = -1;double predicted_confidence = 0.0;// Get the prediction and associated confidence from the modelmodel->predict(img, predicted_label	1								

face_FaceRecognizer_predict2	CVAPI(ExceptionStatus) face_FaceRecognizer_predict2 (cv::face::FaceRecognizer *obj, cv::_InputArray *src, int *label, double *confidence) { BEGIN_WRAP obj->predict(*src, *label, *confidence); END_WRAP }	cv::face::FaceRecognizer::predict()	classcv_1_1face_1_1FaceRecognizer.html	Predicts a label and associated confidence (e.g. distance) for a given input image.	src : Sample image to get a prediction from. label : The predicted label for the given image. confidence : Associated confidence (e.g. distance) for the predicted label.	The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.The following example shows how to get a prediction from a trained model:using namespace cv;// Do your initialization here (create the cv::FaceRecognizer model) ...// ...// Read in a sample image:Mat img = imread("person1/3.jpg", IMREAD_GRAYSCALE);// And get a prediction from the cv::FaceRecognizer:int predicted = model->predict(img);fragmentOr to get a prediction and the associated confidence (e.g. distance):using namespace cv;// Do your initialization here (create the cv::FaceRecognizer model) ...// ...Mat img = imread("person1/3.jpg", IMREAD_GRAYSCALE);// Some variables for the predicted label and associated confidence (e.g. distance):int predicted_label = -1;double predicted_confidence = 0.0;// Get the prediction and associated confidence from the modelmodel->predict(img,	3								
face_FaceRecognizer_write1	CVAPI(ExceptionStatus) face_FaceRecognizer_write1(cv::face::FaceRecognizer *obj, const char *filename) { BEGIN_WRAP obj->write(filename); END_WRAP }	cv::face::FaceRecognizer::write()	classcv_1_1face_1_1FaceRecognizer.html	Saves a FaceRecognizer and its model state.	filename : The filename to store this FaceRecognizer to (either XML/YAML).	Saves this model to a given filename, either as XML or YAML.Every FaceRecognizer overwrites FaceRecognizer::save(FileStorage& fs) to save the internal model state. FaceRecognizer::save(const String& filename) saves the state of a model to the given filename.The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.	1								
face_FaceRecognizer_read1	CVAPI(ExceptionStatus) face_FaceRecognizer_read1(cv::face::FaceRecognizer *obj, const char *filename) { BEGIN_WRAP obj->read(filename); END_WRAP }	cv::face::FaceRecognizer::read()	classcv_1_1face_1_1FaceRecognizer.html	Loads a FaceRecognizer and its model state.		Loads a persisted model and state from a given XML or YAML file . Every FaceRecognizer has to overwrite FaceRecognizer::load(FileStorage& fs) to enable loading the model state. FaceRecognizer::load(FileStorage& fs) in turn gets called by FaceRecognizer::load(const String& filename), to ease saving a model.	1								
face_FaceRecognizer_write2	CVAPI(ExceptionStatus) face_FaceRecognizer_write2(cv::face::FaceRecognizer *obj, cv::FileStorage *fs) { BEGIN_WRAP obj->write(*fs); END_WRAP }	cv::face::FaceRecognizer::write()	classcv_1_1face_1_1FaceRecognizer.html	Saves a FaceRecognizer and its model state.	filename : The filename to store this FaceRecognizer to (either XML/YAML).	Saves this model to a given filename, either as XML or YAML.Every FaceRecognizer overwrites FaceRecognizer::save(FileStorage& fs) to save the internal model state. FaceRecognizer::save(const String& filename) saves the state of a model to the given filename.The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.	1								
face_FaceRecognizer_read2	CVAPI(ExceptionStatus) face_FaceRecognizer_read2(cv::face::FaceRecognizer *obj, cv::FileNode *fn) { BEGIN_WRAP obj->read(*fn); END_WRAP }	cv::face::FaceRecognizer::read()	classcv_1_1face_1_1FaceRecognizer.html	Loads a FaceRecognizer and its model state.		Loads a persisted model and state from a given XML or YAML file . Every FaceRecognizer has to overwrite FaceRecognizer::load(FileStorage& fs) to enable loading the model state. FaceRecognizer::load(FileStorage& fs) in turn gets called by FaceRecognizer::load(const String& filename), to ease saving a model.	1								
face_FaceRecognizer_setLabelInfo	CVAPI(ExceptionStatus) face_FaceRecognizer_setLabelInfo(cv::face::FaceRecognizer *obj, int label, const char *strInfo) { BEGIN_WRAP obj->setLabelInfo(label, strInfo); END_WRAP }	cv::face::FaceRecognizer::setLabelInfo()	classcv_1_1face_1_1FaceRecognizer.html	Sets string info for the specified model's label.		The string info is replaced by the provided value if it was set before for the specified label.	2								
face_FaceRecognizer_getLabelInfo	CVAPI(ExceptionStatus) face_FaceRecognizer_getLabelInfo(cv::face::FaceRecognizer *obj, int label, std::string *dst) { BEGIN_WRAP const auto result = obj->getLabelInfo(label); dst->assign(result); END_WRAP }	cv::face::FaceRecognizer::getLabelInfo()	classcv_1_1face_1_1FaceRecognizer.html	Gets string information by label.		If an unknown label id is provided or there is no label information associated with the specified label id the method returns an empty string.	1								
face_FaceRecognizer_getLabelsByString	CVAPI(ExceptionStatus) face_FaceRecognizer_getLabelsByString(cv::face::FaceRecognizer *obj, const char* str, std::vector<int> *dst) { BEGIN_WRAP const auto result = obj->getLabelsByString(str); std::copy(result.begin(), result.end(), std::back_inserter(*dst)); END_WRAP }	cv::face::FaceRecognizer::getLabelsByString()	classcv_1_1face_1_1FaceRecognizer.html	Gets vector of labels by string.		The function searches for the labels containing the specified sub-string in the associated string info.	3								
face_FaceRecognizer_getThreshold	CVAPI(ExceptionStatus) face_FaceRecognizer_getThreshold(cv::face::FaceRecognizer *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::face::FaceRecognizer::getThreshold()	classcv_1_1face_1_1FaceRecognizer.html	threshold parameter accessor - required for default BestMinDist collector		Implemented in cv::face::LBPHFaceRecognizer, and cv::face::BasicFaceRecognizer.	1								
face_FaceRecognizer_setThreshold	CVAPI(ExceptionStatus) face_FaceRecognizer_setThreshold(cv::face::FaceRecognizer *obj, double val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::face::FaceRecognizer::setThreshold()	classcv_1_1face_1_1FaceRecognizer.html	Sets threshold of model.		Implemented in cv::face::LBPHFaceRecognizer, and cv::face::BasicFaceRecognizer.	1								
face_BasicFaceRecognizer_getNumComponents	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getNumComponents(cv::face::BasicFaceRecognizer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNumComponents(); END_WRAP }	cv::face::BasicFaceRecognizer::getNumComponents()	classcv_1_1face_1_1BasicFaceRecognizer.html	See alsosetNumComponents			1								
face_BasicFaceRecognizer_setNumComponents	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_setNumComponents(cv::face::BasicFaceRecognizer *obj, int val) { BEGIN_WRAP obj->setNumComponents(val); END_WRAP }	cv::face::BasicFaceRecognizer::setNumComponents()	classcv_1_1face_1_1BasicFaceRecognizer.html	See alsogetNumComponents			1								
face_BasicFaceRecognizer_getThreshold	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getThreshold(cv::face::BasicFaceRecognizer *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::face::BasicFaceRecognizer::getThreshold()	classcv_1_1face_1_1BasicFaceRecognizer.html	See alsosetThreshold		Implements cv::face::FaceRecognizer.	1								
face_BasicFaceRecognizer_setThreshold	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_setThreshold(cv::face::BasicFaceRecognizer *obj, double val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::face::BasicFaceRecognizer::setThreshold()	classcv_1_1face_1_1BasicFaceRecognizer.html	See alsogetThreshold		Implements cv::face::FaceRecognizer.	1								
face_BasicFaceRecognizer_getProjections	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getProjections(cv::face::BasicFaceRecognizer *obj, std::vector<cv::Mat> *dst) { BEGIN_WRAP auto result = obj->getProjections(); dst->clear(); dst->reserve(result.size()); for (size_t i = 0; i < result.size(); i++) { dst->push_back(result[i]); } END_WRAP }	cv::face::BasicFaceRecognizer::getProjections()																
face_BasicFaceRecognizer_getLabels	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getLabels(cv::face::BasicFaceRecognizer *obj, cv::Mat *dst) { BEGIN_WRAP const auto result = obj->getLabels(); result.copyTo(*dst); END_WRAP }	cv::face::BasicFaceRecognizer::getLabels()																

face_BasicFaceRecognizer_getEigenValues	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getEigenValues(cv::face::BasicFaceRecognizer *obj, cv::Mat *dst) { BEGIN_WRAP const auto result = obj->getEigenValues(); result.copyTo(*dst); END_WRAP }	cv::face::BasicFaceRecognizer::getEigenValues()														
face_BasicFaceRecognizer_getEigenVectors	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getEigenVectors(cv::face::BasicFaceRecognizer *obj, cv::Mat *dst) { BEGIN_WRAP const auto result = obj->getEigenVectors(); result.copyTo(*dst); END_WRAP }	cv::face::BasicFaceRecognizer::getEigenVectors()														
face_BasicFaceRecognizer_getMean	CVAPI(ExceptionStatus) face_BasicFaceRecognizer_getMean(cv::face::BasicFaceRecognizer *obj, cv::Mat *dst) { BEGIN_WRAP const auto result = obj->getMean(); result.copyTo(*dst); END_WRAP }	cv::face::BasicFaceRecognizer::getMean()														
face_EigenFaceRecognizer_create	CVAPI(ExceptionStatus) face_EigenFaceRecognizer_create(const int numComponents, const double threshold, cv::Ptr<cv::face::EigenFaceRecognizer> **returnValue) { BEGIN_WRAP const auto r = cv::face::EigenFaceRecognizer::create(numComponents, threshold); *returnValue = clone(r); END_WRAP }	cv::face::EigenFaceRecognizer::create()	classcv_1_1face_1_1EigenFaceRecognizer.html	Notes:	num_components : The number of components (read: Eigenfaces) kept for this Principal Component Analysis. As a hint: There's no rule how many components (read: Eigenfaces) should be kept for good reconstruction capabilities. It is based on your input data, so experiment with the number. Keeping 80 components should almost always be sufficient. threshold : The threshold applied in the prediction.	Training and prediction must be done on grayscale images, use cvtColor to convert between the color spaces. THE EIGENFACES METHOD MAKES THE ASSUMPTION, THAT THE TRAINING AND TEST IMAGES ARE OF EQUAL SIZE. (caps-lock, because I got so many mails asking for this). You have to make sure your input data has the correct shape, else a meaningful exception is thrown. Use resize to resize the images. This model does not support updating.Model internal data:num_components see EigenFaceRecognizer::create. threshold see EigenFaceRecognizer::create. eigenvalues The eigenvalues for this Principal Component Analysis (ordered descending). eigenvectors The eigenvectors for this Principal Component Analysis (ordered by their eigenvalue). mean The sample mean calculated from the training data. projections The projections of the training data.	2								
face_Ptr_EigenFaceRecognizer_get	CVAPI(ExceptionStatus) face_Ptr_EigenFaceRecognizer_get(cv::Ptr<cv::face::EigenFaceRecognizer> *obj, cv::face::EigenFaceRecognizer **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::face::EigenFaceRecognizer													•	
face_Ptr_EigenFaceRecognizer_delete	CVAPI(ExceptionStatus) face_Ptr_EigenFaceRecognizer_delete(cv::Ptr<cv::face::EigenFaceRecognizer> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::EigenFaceRecognizer												•		
face_FisherFaceRecognizer_create	CVAPI(ExceptionStatus) face_FisherFaceRecognizer_create(const int numComponents, const double threshold, cv::Ptr<cv::face::FisherFaceRecognizer> **returnValue) { BEGIN_WRAP const auto r = cv::face::FisherFaceRecognizer::create(numComponents, threshold); *returnValue = clone(r); END_WRAP }	cv::face::FisherFaceRecognizer::create()	classcv_1_1face_1_1FisherFaceRecognizer.html	Notes:	num_components : The number of components (read: Fisherfaces) kept for this Linear Discriminant Analysis with the Fisherfaces criterion. It's useful to keep all components, that means the number of your classes c (read: subjects, persons you want to recognize). If you leave this at the default (0) or set it to a value less-equal 0 or greater (c-1), it will be set to the correct number (c-1) automatically. threshold : The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.	Training and prediction must be done on grayscale images, use cvtColor to convert between the color spaces. THE FISHERFACES METHOD MAKES THE ASSUMPTION, THAT THE TRAINING AND TEST IMAGES ARE OF EQUAL SIZE. (caps-lock, because I got so many mails asking for this). You have to make sure your input data has the correct shape, else a meaningful exception is thrown. Use resize to resize the images. This model does not support updating.Model internal data:num_components see FisherFaceRecognizer::create. threshold see FisherFaceRecognizer::create. eigenvalues The eigenvalues for this Linear Discriminant Analysis (ordered descending). eigenvectors The eigenvectors for this Linear Discriminant Analysis (ordered by their eigenvalue). mean The sample mean calculated from the training data. projections The projections of the training data.	2								
face_Ptr_FisherFaceRecognizer_get	CVAPI(ExceptionStatus) face_Ptr_FisherFaceRecognizer_get(cv::Ptr<cv::face::FisherFaceRecognizer> *obj, cv::face::FisherFaceRecognizer **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::face::FisherFaceRecognizer													•	
face_Ptr_FisherFaceRecognizer_delete	CVAPI(ExceptionStatus) face_Ptr_FisherFaceRecognizer_delete(cv::Ptr<cv::face::FisherFaceRecognizer> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::FisherFaceRecognizer												•		
face_LBPHFaceRecognizer_create	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_create(const int radius, const int neighbors, const int gridX, const int gridY, const double threshold, cv::Ptr<cv::face::LBPHFaceRecognizer> **returnValue) { BEGIN_WRAP const auto r = cv::face::LBPHFaceRecognizer::create(radius, neighbors, gridX, gridY, threshold); *returnValue = clone(r); END_WRAP }	cv::face::LBPHFaceRecognizer::create()	classcv_1_1face_1_1LBPHFaceRecognizer.html	Notes:	radius : The radius used for building the Circular Local Binary Pattern. The greater the radius, the smoother the image but more spatial information you can get. neighbors : The number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use 8 sample points. Keep in mind: the more sample points you include, the higher the computational cost. grid_x : The number of cells in the horizontal direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. grid_y : The number of cells in the vertical direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. threshold : The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.	The Circular Local Binary Patterns (used in training and prediction) expect the data given as grayscale images, use cvtColor to convert between the color spaces. This model supports updating.Model internal data:radius see LBPHFaceRecognizer::create. neighbors see LBPHFaceRecognizer::create. grid_x see LBPHFaceRecognizer::create. grid_y see LBPHFaceRecognizer::create. threshold see LBPHFaceRecognizer::create. histograms Local Binary Patterns Histograms calculated from the given training data (empty if none was given). labels Labels corresponding to the calculated Local Binary Patterns Histograms.	5								
face_LBPHFaceRecognizer_getGridX	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getGridX(cv::face::LBPHFaceRecognizer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getGridX(); END_WRAP }	cv::face::LBPHFaceRecognizer::getGridX()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsosetGridX			1								
face_LBPHFaceRecognizer_setGridX	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_setGridX(cv::face::LBPHFaceRecognizer *obj, int val) { BEGIN_WRAP obj->setGridX(val); END_WRAP }	cv::face::LBPHFaceRecognizer::setGridX()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsogetGridX			1								
face_LBPHFaceRecognizer_getGridY	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getGridY(cv::face::LBPHFaceRecognizer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getGridY(); END_WRAP }	cv::face::LBPHFaceRecognizer::getGridY()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsosetGridY			1								
face_LBPHFaceRecognizer_setGridY	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_setGridY(cv::face::LBPHFaceRecognizer *obj, int val) { BEGIN_WRAP obj->setGridY(val); END_WRAP }	cv::face::LBPHFaceRecognizer::setGridY()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsogetGridY			1								
face_LBPHFaceRecognizer_getRadius	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getRadius(cv::face::LBPHFaceRecognizer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getRadius(); END_WRAP }	cv::face::LBPHFaceRecognizer::getRadius()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsosetRadius			1								

face_LBPHFaceRecognizer_setRadius	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_setRadius(cv::face::LBPHFaceRecognizer *obj, int val) { BEGIN_WRAP obj->setRadius(val); END_WRAP }	cv::face::LBPHFaceRecognizer::setRadius()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsogetRadius			1										
face_LBPHFaceRecognizer_getNeighbors	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getNeighbors(cv::face::LBPHFaceRecognizer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNeighbors(); END_WRAP }	cv::face::LBPHFaceRecognizer::getNeighbors()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsosetNeighbors			1										
face_LBPHFaceRecognizer_setNeighbors	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_setNeighbors(cv::face::LBPHFaceRecognizer *obj, int val) { BEGIN_WRAP obj->setNeighbors(val); END_WRAP }	cv::face::LBPHFaceRecognizer::setNeighbors()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsogetNeighbors			1										
face_LBPHFaceRecognizer_getThreshold	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getThreshold(cv::face::LBPHFaceRecognizer *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::face::LBPHFaceRecognizer::getThreshold()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsosetThreshold		Implements cv::face::FaceRecognizer.	1									
face_LBPHFaceRecognizer_setThreshold	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_setThreshold(cv::face::LBPHFaceRecognizer *obj, double val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::face::LBPHFaceRecognizer::setThreshold()	classcv_1_1face_1_1LBPHFaceRecognizer.html	See alsogetThreshold		Implements cv::face::FaceRecognizer.	1									
face_LBPHFaceRecognizer_getHistograms	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getHistograms(cv::face::LBPHFaceRecognizer *obj, std::vector<cv::Mat> *dst) { BEGIN_WRAP auto result = obj->getHistograms(); dst->clear(); dst->reserve(result.size()); for (size_t i = 0; i < result.size(); i++) { dst->at(i) = result[i]; } END_WRAP }	cv::face::LBPHFaceRecognizer::getHistograms()															
face_LBPHFaceRecognizer_getLabels	CVAPI(ExceptionStatus) face_LBPHFaceRecognizer_getLabels(cv::face::LBPHFaceRecognizer *obj, cv::Mat *dst) { BEGIN_WRAP const auto result = obj->getLabels(); result.copyTo(*dst); END_WRAP }	cv::face::LBPHFaceRecognizer::getLabels()															
face_Ptr_LBPHFaceRecognizer_get	CVAPI(ExceptionStatus) face_Ptr_LBPHFaceRecognizer_get(cv::Ptr<cv::face::LBPHFaceRecognizer> *obj, cv::face::LBPHFaceRecognizer **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::face::LBPHFaceRecognizer														•	
face_Ptr_LBPHFaceRecognizer_delete	CVAPI(ExceptionStatus) face_Ptr_LBPHFaceRecognizer_delete(cv::Ptr<cv::face::LBPHFaceRecognizer> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::face::LBPHFaceRecognizer													•		
features2d_drawKeypoints	CVAPI(ExceptionStatus) features2d_drawKeypoints(cv::_InputArray *image, cv::KeyPoint *keypoints, int keypointsLength, cv::_InputOutputArray *outImage, MyCvScalar color, int flags) { BEGIN_WRAP const std::vector<cv::KeyPoint> keypointsVec(keypoints, keypoints + keypointsLength); cv::drawKeypoints(*image, keypointsVec, *outImage, cpp(color), static_cast<cv::DrawMatchesFlags>(flags)); END_WRAP }	cv::drawKeypoints()	group__features2d__draw.html	Draws keypoints.	image : Source image. keypoints : Keypoints from the source image. outImage : Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below. color : Color of keypoints. flags : Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags. See details above in drawMatches .	NoteFor Python API, flags are modified as cv.DRAW_MATCHES_FLAGS_DEFAULT, cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, cv.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG, cv.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS	5								
features2d_drawMatches	CVAPI(ExceptionStatus) features2d_drawMatches(cv::Mat *img1, cv::KeyPoint *keypoints1, int keypoints1Length, cv::Mat *img2, cv::KeyPoint *keypoints2, int keypoints2Length, cv::DMatch *matches1to2, int matches1to2Length, cv::Mat *outImg, MyCvScalar matchColor, MyCvScalar singlePointColor, char *matchesMask, int matchesMaskLength, int flags) { BEGIN_WRAP const std::vector<cv::KeyPoint> keypoints1Vec(keypoints1, keypoints1 + keypoints1Length); const std::vector<cv::KeyPoint> keypoints2Vec(keypoints2, keypoints2 +	cv::drawMatches()	group__features2d__draw.html	Draws the found matches of keypoints from two images.	img1 : First source image. keypoints1 : Keypoints from the first source image. img2 : Second source image. keypoints2 : Keypoints from the second source image. matches1to2 : Matches from the first image to the second one, which means that keypoints1[i] has a corresponding point in keypoints2[matches[i]] . outImg : Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below. matchColor : Color of matches (lines and connected keypoints). If matchColor==Scalar::all(-1) , the color is generated randomly. singlePointColor : Color of single keypoints (circles), which means that keypoints do not have the matches. If singlePointColor==Scalar::all(-1) , the color is generated randomly. matchesMask : Mask determining which matches are drawn. If the mask is empty, all matches are drawn. flags : Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags.	This function draws matches of keypoints from two images in the output image. Match is a line connecting two keypoints (circles). See cv::DrawMatchesFlags.	10								
features2d_drawMatchesKnn	CVAPI(ExceptionStatus) features2d_drawMatchesKnn(cv::Mat *img1, cv::KeyPoint *keypoints1, int keypoints1Length, cv::Mat *img2, cv::KeyPoint *keypoints2, int keypoints2Length, cv::DMatch **matches1to2, int matches1to2Size1, int *matches1to2Size2, cv::Mat *outImg, MyCvScalar matchColor, MyCvScalar singlePointColor, char **matchesMask, int matchesMaskSize1, int *matchesMaskSize2, int flags) { BEGIN_WRAP const std::vector<cv::KeyPoint> keypoints1Vec(keypoints1, keypoints1 + keypoints1Length); const std::vector<cv::KeyPoint>	cv::drawMatches()	group__features2d__draw.html	Draws the found matches of keypoints from two images.	img1 : First source image. keypoints1 : Keypoints from the first source image. img2 : Second source image. keypoints2 : Keypoints from the second source image. matches1to2 : Matches from the first image to the second one, which means that keypoints1[i] has a corresponding point in keypoints2[matches[i]] . outImg : Output image. Its content depends on the flags value defining what is drawn in the output image. See possible flags bit values below. matchColor : Color of matches (lines and connected keypoints). If matchColor==Scalar::all(-1) , the color is generated randomly. singlePointColor : Color of single keypoints (circles), which means that keypoints do not have the matches. If singlePointColor==Scalar::all(-1) , the color is generated randomly. matchesMask : Mask determining which matches are drawn. If the mask is empty, all matches are drawn. flags : Flags setting drawing features. Possible flags bit values are defined by DrawMatchesFlags.	This function draws matches of keypoints from two images in the output image. Match is a line connecting two keypoints (circles). See cv::DrawMatchesFlags.	10								
features2d_evaluateFeatureDetector	CVAPI(ExceptionStatus) features2d_evaluateFeatureDetector(cv::Mat *img1, cv::Mat *img2, cv::Mat *H1to2, std::vector<cv::KeyPoint> *keypoints1, std::vector<cv::KeyPoint> *keypoints2, float *repeatability, int *correspCount/*, const Ptr<FeatureDetector>& fdetector = Ptr<FeatureDetector>()*/) { BEGIN_WRAP cv::evaluateFeatureDetector(*img1, *img2, *H1to2, keypoints1, keypoints2, *repeatability, *correspCount); END_WRAP }	cv::evaluateFeatureDetector()	namespacecv.html				7								

features2d_computeRecallPrecisionCurve	CVAPI(ExceptionStatus) features2d_computeRecallPrecisionCurve(cv::DMatch **matchesIto2, int matchesIto2Size1, int *matchesIto2Size2, uchar **correctMatchesIto2Mask, int correctMatchesIto2MaskSize1, int *correctMatchesIto2MaskSize2, std::vector<cv::Point2f> *recallPrecisionCurve) { BEGIN_WRAP std::vector<std::vector<cv::DMatch> > matchesIto2Vec; std::vector<std::vector<uchar> > correctMatchesIto2MaskVec; matchesIto2Vec.reserve(mat	cv::computeRecallPrecisionCurve()	namespacecv.html					3									
features2d_getRecall	CVAPI(ExceptionStatus) features2d_getRecall(cv::Point2f *recallPrecisionCurve, int recallPrecisionCurveSize, float l_precision, float *returnValue) { BEGIN_WRAP const std::vector<cv::Point2f> recallPrecisionCurveVec(recallPrecisionCurve, recallPrecisionCurve + recallPrecisionCurveSize); *returnValue = cv::getRecall(recallPrecisionCurveVec, l_precision); END_WRAP }	cv::getRecall()	namespacecv.html					2									
features2d_getNearestPoint	CVAPI(ExceptionStatus) features2d_getNearestPoint(cv::Point2f *recallPrecisionCurve, int recallPrecisionCurveSize, float l_precision, int *returnValue) { BEGIN_WRAP const std::vector<cv::Point2f> recallPrecisionCurveVec(recallPrecisionCurve, recallPrecisionCurve + recallPrecisionCurveSize); *returnValue = cv::getNearestPoint(recallPrecisionCurveVec, l_precision); END_WRAP }	cv::getNearestPoint()	namespacecv.html					2									
features2d_KeyPointsFilter_runByImageBorder	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_runByImageBorder(std::vector<cv::KeyPoint> *keypoints, MyCvSize imageSize, int borderSize) { BEGIN_WRAP cv::KeyPointsFilter::runByImageBorder(*keypoints, cpp(imageSize), borderSize); END_WRAP }	cv::KeyPointsFilter::runByImageBorder()	classcv_1_1KeyPointsFilter.html					3									
features2d_KeyPointsFilter_runByKeypointSize	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_runByKeypointSize(std::vector<cv::KeyPoint> *keypoints, float minSize, float maxSize) { BEGIN_WRAP cv::KeyPointsFilter::runByKeypointSize(*keypoints, minSize, maxSize); END_WRAP }	cv::KeyPointsFilter::runByKeypointSize()	classcv_1_1KeyPointsFilter.html					3									
features2d_KeyPointsFilter_runByPixelsMask	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_runByPixelsMask(std::vector<cv::KeyPoint> *keypoints, cv::Mat *mask) { BEGIN_WRAP cv::KeyPointsFilter::runByPixelsMask(*keypoints, *mask); END_WRAP }	cv::KeyPointsFilter::runByPixelsMask()	classcv_1_1KeyPointsFilter.html					2									
features2d_KeyPointsFilter_removeDuplicated	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_removeDuplicated(std::vector<cv::KeyPoint> *keypoints) { BEGIN_WRAP cv::KeyPointsFilter::removeDuplicated(*keypoints); END_WRAP }	cv::KeyPointsFilter::removeDuplicated()	classcv_1_1KeyPointsFilter.html					1									
features2d_KeyPointsFilter_removeDuplicatedS	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_removeDuplicatedSorted(std::vector<cv::KeyPoint> *keypoints) { BEGIN_WRAP cv::KeyPointsFilter::removeDuplicatedSorted(*keypoints); END_WRAP }	cv::KeyPointsFilter::removeDuplicatedSorted()	classcv_1_1KeyPointsFilter.html					1									
features2d_KeyPointsFilter_retainBest	CVAPI(ExceptionStatus) features2d_KeyPointsFilter_retainBest(std::vector<cv::KeyPoint> *keypoints, int nPoints) { BEGIN_WRAP cv::KeyPointsFilter::retainBest(*keypoints, nPoints); END_WRAP }	cv::KeyPointsFilter::retainBest()	classcv_1_1KeyPointsFilter.html					2									
features2d_BOWTrainer_add	CVAPI(ExceptionStatus) features2d_BOWTrainer_add(cv::BOWTrainer *obj, cv::Mat *descriptors) { BEGIN_WRAP obj->add(*descriptors); END_WRAP }	cv::BOWTrainer::add()	classcv_1_1BOWTrainer.html	Adds descriptors to a training set.	descriptors : Descriptors to add to a training set. Each row of the descriptors matrix is a descriptor.	The training set is clustered using clustermethod to construct the vocabulary.		1									
features2d_BOWTrainer_getDescriptors	CVAPI(ExceptionStatus) features2d_BOWTrainer_getDescriptors(cv::BOWTrainer *obj, std::vector<cv::Mat> *descriptors) { BEGIN_WRAP const std::vector<cv::Mat> d = obj->getDescriptors(); std::copy(d.begin(), d.end(), std::back_inserter(*descriptors)); END_WRAP }	cv::BOWTrainer::getDescriptors()	classcv_1_1BOWTrainer.html	Returns a training set of descriptors.				3									
features2d_BOWTrainer_descriptorsCount	CVAPI(ExceptionStatus) features2d_BOWTrainer_descriptorsCount(cv::BOWTrainer *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->descriptorsCount(); END_WRAP }	cv::BOWTrainer::descriptorsCount()	classcv_1_1BOWTrainer.html	Returns the count of all descriptors stored in the training set.				1									
features2d_BOWTrainer_clear	CVAPI(ExceptionStatus) features2d_BOWTrainer_clear(cv::BOWTrainer *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::BOWTrainer::clear()	classcv_1_1BOWTrainer.html					1									
features2d_BOWKMeansTrainer_new	CVAPI(ExceptionStatus) features2d_BOWKMeansTrainer_new(int clusterCount, MyCvTermCriteria termcrit, int attempts, int flags, cv::BOWKMeansTrainer **returnValue) { BEGIN_WRAP *returnValue = new cv::BOWKMeansTrainer(clusterCount, cpp(termcrit), attempts, flags); END_WRAP }	cv::BOWKMeansTrainer	classcv_1_1BOWKMeansTrainer.html	kmeans -based class to train visual vocabulary using the bag of visual words approach. :								•					

features2d_BOWKMeansTrainer_delete	CVAPI(ExceptionStatus) features2d_BOWKMeansTrainer_delete(cv::BOWKMeansTrainer *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::BOWKMeansTrainer	classcv_1_tBOWKMeansTrainer.html	kmeans -based class to train visual vocabulary using the bag of visual words approach. :													
features2d_BOWKMeansTrainer_cluster1	CVAPI(ExceptionStatus) features2d_BOWKMeansTrainer_cluster1(cv::BOWKMeansTrainer *obj, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat m = obj->cluster(); *returnValue = new cv::Mat(m); END_WRAP }	cv::BOWKMeansTrainer::cluster()	classcv_1_tBOWKMeansTrainer.html	Clusters train descriptors.	descriptors : Descriptors to cluster. Each row of the descriptors matrix is a descriptor. Descriptors are not added to the inner train descriptor set.	The vocabulary consists of cluster centers. So, this method returns the vocabulary. In the first variant of the method, train descriptors stored in the object are clustered. In the second variant, input descriptors are clustered.Implements cv::BOWTrainer.	1									
features2d_BOWKMeansTrainer_cluster2	CVAPI(ExceptionStatus) features2d_BOWKMeansTrainer_cluster2(cv::BOWKMeansTrainer *obj, cv::Mat *descriptors, cv::Mat **returnValue) { BEGIN_WRAP const cv::Mat m = obj->cluster(*descriptors); *returnValue = new cv::Mat(m); END_WRAP }	cv::BOWKMeansTrainer::cluster()	classcv_1_tBOWKMeansTrainer.html	Clusters train descriptors.	descriptors : Descriptors to cluster. Each row of the descriptors matrix is a descriptor. Descriptors are not added to the inner train descriptor set.	The vocabulary consists of cluster centers. So, this method returns the vocabulary. In the first variant of the method, train descriptors stored in the object are clustered. In the second variant, input descriptors are clustered.Implements cv::BOWTrainer.	1									
features2d_BOWImgDescriptorExtractor_new1	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_new1_Ptr(cv::Ptr<cv::DescriptorExtractor> *dextractor, cv::Ptr<cv::DescriptorMatcher> *dmatcher, cv::BOWImgDescriptorExtractor **returnValue) { BEGIN_WRAP *returnValue = new cv::BOWImgDescriptorExtractor(*dextractor, *dmatcher); END_WRAP }	cv::BOWImgDescriptorExtractor	classcv_1_tBOWImgDescriptorExtractor.html	Class to compute an image descriptor using the bag of visual words. Such a computation consists of the following steps: Compute descriptors for a given image and its keypoints set. Find the nearest visual words from the vocabulary for each keypoint descriptor. Compute the bag-of-words image descriptor as is a normalized histogram of vocabulary words encountered in the image. The i-th bin of the histogram is a frequency of i-th word of the vocabulary in the given image.								•			
features2d_BOWImgDescriptorExtractor_new2	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_new2_Ptr(cv::Ptr<cv::DescriptorMatcher> *dmatcher, cv::BOWImgDescriptorExtractor **returnValue) { BEGIN_WRAP *returnValue = new cv::BOWImgDescriptorExtractor(*dmatcher); END_WRAP }	cv::BOWImgDescriptorExtractor	classcv_1_tBOWImgDescriptorExtractor.html	Class to compute an image descriptor using the bag of visual words. Such a computation consists of the following steps: Compute descriptors for a given image and its keypoints set. Find the nearest visual words from the vocabulary for each keypoint descriptor. Compute the bag-of-words image descriptor as is a normalized histogram of vocabulary words encountered in the image. The i-th bin of the histogram is a frequency of i-th word of the vocabulary in the given image.								•			
features2d_BOWImgDescriptorExtractor_new1	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_new1_RawPtr(cv::DescriptorExtractor *dextractor, cv::DescriptorMatcher *dmatcher, cv::BOWImgDescriptorExtractor **returnValue) { BEGIN_WRAP // do not delete dextractor and dmatcher const cv::Ptr<cv::DescriptorExtractor> dextractorPtr(dextractor, DescriptorExtractorDeleter); const cv::Ptr<cv::DescriptorMatcher> dmatcherPtr(dmatcher, DescriptorMatcherDeleter); *returnValue = new cv::BOWImgDescriptorExtractor(dextractorPtr, dmatcherPtr); END_WRAP }	cv::BOWImgDescriptorExtractor	classcv_1_tBOWImgDescriptorExtractor.html	Class to compute an image descriptor using the bag of visual words. Such a computation consists of the following steps: Compute descriptors for a given image and its keypoints set. Find the nearest visual words from the vocabulary for each keypoint descriptor. Compute the bag-of-words image descriptor as is a normalized histogram of vocabulary words encountered in the image. The i-th bin of the histogram is a frequency of i-th word of the vocabulary in the given image.								•			
features2d_BOWImgDescriptorExtractor_new2	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_new2_RawPtr(cv::DescriptorMatcher *dmatcher, cv::BOWImgDescriptorExtractor **returnValue) { BEGIN_WRAP // do not delete dmatcher const cv::Ptr<cv::DescriptorMatcher> dmatcherPtr(dmatcher, DescriptorMatcherDeleter); *returnValue = new cv::BOWImgDescriptorExtractor(dmatcherPtr); END_WRAP }	cv::BOWImgDescriptorExtractor	classcv_1_tBOWImgDescriptorExtractor.html	Class to compute an image descriptor using the bag of visual words. Such a computation consists of the following steps: Compute descriptors for a given image and its keypoints set. Find the nearest visual words from the vocabulary for each keypoint descriptor. Compute the bag-of-words image descriptor as is a normalized histogram of vocabulary words encountered in the image. The i-th bin of the histogram is a frequency of i-th word of the vocabulary in the given image.								•			
features2d_BOWImgDescriptorExtractor_delete	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_delete(cv::BOWImgDescriptorExtractor *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::BOWImgDescriptorExtractor	classcv_1_tBOWImgDescriptorExtractor.html	Class to compute an image descriptor using the bag of visual words. Such a computation consists of the following steps: Compute descriptors for a given image and its keypoints set. Find the nearest visual words from the vocabulary for each keypoint descriptor. Compute the bag-of-words image descriptor as is a normalized histogram of vocabulary words encountered in the image. The i-th bin of the histogram is a frequency of i-th word of the vocabulary in the given image.									•		
features2d_BOWImgDescriptorExtractor_setVo	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_setVocabulary(cv::BOWImgDescriptorExtractor *obj, cv::Mat *vocabulary) { BEGIN_WRAP obj->setVocabulary(*vocabulary); END_WRAP }	cv::BOWImgDescriptorExtractor::setVocabulary()	classcv_1_tBOWImgDescriptorExtractor.html	Sets a visual vocabulary.	vocabulary : Vocabulary (can be trained using the inheritor of BOWTrainer). Each row of the vocabulary is a visual word (cluster center).		1									
features2d_BOWImgDescriptorExtractor_getVo	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_getVocabulary(cv::BOWImgDescriptorExtractor *obj, cv::Mat **returnValue) { BEGIN_WRAP cv::Mat m = obj->getVocabulary(); *returnValue = new cv::Mat(m); END_WRAP }	cv::BOWImgDescriptorExtractor::getVocabulary()	classcv_1_tBOWImgDescriptorExtractor.html	Returns the set vocabulary.			1									
features2d_BOWImgDescriptorExtractor_compu	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_compute11(cv::BOWImgDescriptorExtractor *obj, cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, cv::_OutputArray *imgDescriptor, std::vector<std::vector<int> >* pointIdxsOfClusters, cv::Mat* descriptors) { BEGIN_WRAP obj->compute(*image, *keypoints, *imgDescriptor, pointIdxsOfClusters, descriptors); END_WRAP }	cv::BOWImgDescriptorExtractor::compute()	classcv_1_tBOWImgDescriptorExtractor.html	Computes an image descriptor using the set visual vocabulary.	image : Image, for which the descriptor is computed. keypoints : Keypoints detected in the input image. imgDescriptor : Computed output image descriptor. pointIdxsOfClusters : Indices of keypoints that belong to the cluster. This means that pointIdxsOfClusters[i] are keypoint indices that belong to the i -th cluster (word of vocabulary) returned if it is non-zero. descriptors : Descriptors of the image keypoints that are returned if they are non-zero.		5									
features2d_BOWImgDescriptorExtractor_compu	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_compute12(cv::BOWImgDescriptorExtractor *obj, cv::_InputArray *keypointDescriptors, cv::_OutputArray *imgDescriptor, std::vector<std::vector<int> >* pointIdxsOfClusters) { BEGIN_WRAP obj->compute(*keypointDescriptors, *imgDescriptor, pointIdxsOfClusters); END_WRAP }	cv::BOWImgDescriptorExtractor::compute()	classcv_1_tBOWImgDescriptorExtractor.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	keypointDescriptors : Computed descriptors to match with vocabulary. imgDescriptor : Computed output image descriptor. pointIdxsOfClusters : Indices of keypoints that belong to the cluster. This means that pointIdxsOfClusters[i] are keypoint indices that belong to the i -th cluster (word of vocabulary) returned if it is non-zero.		3									

features2d_BOWImgDescriptorExtractor_comp	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_compute2(cv::BOWImgDescriptorExtractor *obj, cv::Mat *image, std::vector<cv::KeyPoint> *keypoints, cv::Mat *imgDescriptor) { BEGIN_WRAP obj->compute2(*image, *keypoints, *imgDescriptor); END_WRAP }	cv::BOWImgDescriptorExtractor::compute2()	classcv_1_1BOWImgDescriptorExtractor.html				3										
features2d_BOWImgDescriptorExtractor_descri	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_descriptorSize(cv::BOWImgDescriptorExtractor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->descriptorSize(); END_WRAP }	cv::BOWImgDescriptorExtractor::descriptorSize()	classcv_1_1BOWImgDescriptorExtractor.html	Returns an image descriptor size if the vocabulary is set. Otherwise, it returns 0.			1										
features2d_BOWImgDescriptorExtractor_descri	CVAPI(ExceptionStatus) features2d_BOWImgDescriptorExtractor_descriptorType(cv::BOWImgDescriptorExtractor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->descriptorType(); END_WRAP }	cv::BOWImgDescriptorExtractor::descriptorType()	classcv_1_1BOWImgDescriptorExtractor.html	Returns an image descriptor type.			1										
features2d_DescriptorMatcher_add	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_add(cv::DescriptorMatcher *obj, cv::Mat **descriptors, int descriptorLength) { BEGIN_WRAP std::vector<cv::Mat> descriptorsVec(descriptorLength); for (int i = 0; i < descriptorLength; i++) { descriptorsVec[i] = *descriptors[i]; } obj->add(descriptorsVec); END_WRAP }	cv::DescriptorMatcher::add()	classcv_1_1DescriptorMatcher.html	Adds descriptors to train a CPU(trainDescCollectionis) or GPU(utrainDescCollectionis) descriptor collection.	descriptors : Descriptors to add. Each descriptors[i] is a set of descriptors from the same train image.	If the collection is not empty, the new descriptors are added to existing train descriptors.Reimplemented in cv::FlannBasedMatcher.	1									
features2d_DescriptorMatcher_getTrainDescrip	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_getTrainDescriptors(cv::DescriptorMatcher *obj, std::vector<cv::Mat> *dst) { BEGIN_WRAP *dst = obj->getTrainDescriptors(); END_WRAP }	cv::DescriptorMatcher::getTrainDescriptors()	classcv_1_1DescriptorMatcher.html	Returns a constant link to the train descriptor collection trainDescCollection .			1										
features2d_DescriptorMatcher_clear	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_clear(cv::DescriptorMatcher *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::DescriptorMatcher::clear()	classcv_1_1DescriptorMatcher.html	Clears the train descriptor collections.		Reimplemented from cv::Algorithm.Reimplemented in cv::FlannBasedMatcher.	1										
features2d_DescriptorMatcher_empty	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_empty(cv::DescriptorMatcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::DescriptorMatcher::empty()	classcv_1_1DescriptorMatcher.html	Returns true if there are no train descriptors in the both collections.		Reimplemented from cv::Algorithm.	1										
features2d_DescriptorMatcher_isMaskSupporte	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_isMaskSupported(cv::DescriptorMatcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isMaskSupported() ? 1 : 0; END_WRAP }	cv::DescriptorMatcher::isMaskSupported()	classcv_1_1DescriptorMatcher.html	Returns true if the descriptor matcher supports masking permissible matches.		Implemented in cv::FlannBasedMatcher, and cv::BFMatcher.	1										
features2d_DescriptorMatcher_train	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_train(cv::DescriptorMatcher *obj) { BEGIN_WRAP obj->train(); END_WRAP }	cv::DescriptorMatcher::train()	classcv_1_1DescriptorMatcher.html	Trains a descriptor matcher.		Trains a descriptor matcher (for example, the flann index). In all methods to match, the method train() is run every time before matching. Some descriptor matchers (for example, BruteForceMatcher) have an empty implementation of this method. Other matchers really train their inner structures (for example, FlannBasedMatcher trains flann::Index).Reimplemented in cv::FlannBasedMatcher.	1									
features2d_DescriptorMatcher_match1	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_match1(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, cv::Mat *trainDescriptors, std::vector<cv::DMatch> *matches, cv::Mat *mask) { BEGIN_WRAP obj->match(*queryDescriptors, *trainDescriptors, *matches, entity(mask)); END_WRAP }	cv::DescriptorMatcher::match()	classcv_1_1DescriptorMatcher.html	Finds the best match for each descriptor from a query set.	queryDescriptors : Query set of descriptors. trainDescriptors : Train set of descriptors. This set is not added to the train descriptors collection stored in the class object. matches : Matches. If a query descriptor is masked out in mask , no match is added for this descriptor. So, matches size may be smaller than the query descriptors count. mask : Mask specifying permissible matches between an input query and train matrices of descriptors.	In the first variant of this method, the train descriptors are passed as an input argument. In the second variant of the method, train descriptors collection that was set by DescriptorMatcher::add is used. Optional mask (or masks) can be passed to specify which query and training descriptors can be matched. Namely, queryDescriptors[i] can be matched with trainDescriptors[j] only if mask.at<uchar>(i,j) is non-zero.	4									
features2d_DescriptorMatcher_knnMatch1	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_knnMatch1(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, cv::Mat *trainDescriptors, std::vector<std::vector<cv::DMatch> > *matches, int k, cv::Mat *mask, int compactResult) { BEGIN_WRAP obj->knnMatch(*queryDescriptors, *trainDescriptors, *matches, k, entity(mask), compactResult != 0); END_WRAP }	cv::DescriptorMatcher::knnMatch()	classcv_1_1DescriptorMatcher.html	Finds the k best matches for each descriptor from a query set.	queryDescriptors : Query set of descriptors. trainDescriptors : Train set of descriptors. This set is not added to the train descriptors collection stored in the class object. mask : Mask specifying permissible matches between an input query and train matrices of descriptors. matches : Matches. Each matches[i] is k or less matches for the same query descriptor. k : Count of best matches found per each query descriptor or less if a query descriptor has less than k possible matches in total. compactResult : Parameter used when the mask (or masks) is not empty. If compactResult is false, the matches vector has the same size as queryDescriptors rows. If compactResult is true, the matches vector does not contain matches for fully masked-out query descriptors.	These extended variants of DescriptorMatcher::match methods find several best matches for each query descriptor. The matches are returned in the distance increasing order. See DescriptorMatcher::match for the details about query and train descriptors.	6									
features2d_DescriptorMatcher_radiusMatch1	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_radiusMatch1(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, cv::Mat *trainDescriptors, std::vector<std::vector<cv::DMatch> > *matches, float maxDistance, cv::Mat *mask, int compactResult) { BEGIN_WRAP obj->radiusMatch(*queryDescriptors, *trainDescriptors, *matches, maxDistance, entity(mask), compactResult != 0); END_WRAP }	cv::DescriptorMatcher::radiusMatch()	classcv_1_1DescriptorMatcher.html	For each query descriptor, finds the training descriptors not farther than the specified distance.	queryDescriptors : Query set of descriptors. trainDescriptors : Train set of descriptors. This set is not added to the train descriptors collection stored in the class object. matches : Found matches. compactResult : Parameter used when the mask (or masks) is not empty. If compactResult is false, the matches vector has the same size as queryDescriptors rows. If compactResult is true, the matches vector does not contain matches for fully masked-out query descriptors. maxDistance : Threshold for the distance between matched descriptors. Distance means here metric distance (e.g. Hamming distance), not the distance between coordinates (which is measured in Pixels)! mask : Mask specifying permissible matches between an input query and train matrices of descriptors.	For each query descriptor, the methods find such training descriptors that the distance between the query descriptor and the training descriptor is equal or smaller than maxDistance. Found matches are returned in the distance increasing order.	6									
features2d_DescriptorMatcher_match2	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_match2(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, std::vector<cv::DMatch> *matches, cv::Mat **masks, int masksSize) { BEGIN_WRAP std::vector<cv::Mat> masksVal; if (masksSize != 0) { masksVal = std::vector<cv::Mat>(masksSize); for (int i = 0; i < masksSize; i++) { masksVal[i] = *(masks[i]); } } obj->	cv::DescriptorMatcher::match()	classcv_1_1DescriptorMatcher.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	queryDescriptors : Query set of descriptors. matches : Matches. If a query descriptor is masked out in mask , no match is added for this descriptor. So, matches size may be smaller than the query descriptors count. masks : Set of masks. Each masks[i] specifies permissible matches between the input query descriptors and stored train descriptors from the i-th image trainDescCollection[i].		3									

features2d_DescriptorMatcher_knnMatch2	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_knnMatch2(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, std::vector<std::vector<cv::DMatch> > *matches, int k, cv::Mat **masks, int masksSize, int compactResult) { BEGIN_WRAP std::vector<cv::Mat> masksVal; if (masksSize != 0) { masksVal = std::vector<cv::Mat>(masksSize); for (int i = 0; i < masksSize; i++) { masksVal[i] = *(masks[i]); } `	cv::DescriptorMatcher::knnMatch()	classcv_1_1DescriptorMatcher.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	queryDescriptors : Query set of descriptors. matches : Matches. Each matches[i] is k or less matches for the same query descriptor. k : Count of best matches found per each query descriptor or less if a query descriptor has less than k possible matches in total. masks : Set of masks. Each masks[i] specifies permissible matches between the input query descriptors and stored train descriptors from the i-th image trainDescCollection[i]. compactResult : Parameter used when the mask (or masks) is not empty. If compactResult is false, the matches vector has the same size as queryDescriptors rows. If compactResult is true, the matches vector does not contain matches for fully masked-out query descriptors.		5								
features2d_DescriptorMatcher_radiusMatch2	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_radiusMatch2(cv::DescriptorMatcher *obj, cv::Mat *queryDescriptors, std::vector<std::vector<cv::DMatch> > *matches, float maxDistance, cv::Mat **masks, int masksSize, int compactResult) { BEGIN_WRAP std::vector<cv::Mat> masksVal; if (masksSize != 0) { masksVal = std::vector<cv::Mat>(masksSize); for (int i = 0; i < masksSize; i++) { masksVal[i] = *(masks[i]); } `	cv::DescriptorMatcher::radiusMatch()	classcv_1_1DescriptorMatcher.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	queryDescriptors : Query set of descriptors. matches : Found matches. maxDistance : Threshold for the distance between matched descriptors. Distance means here metric distance (e.g. Hamming distance), not the distance between coordinates (which is measured in Pixels)! masks : Set of masks. Each masks[i] specifies permissible matches between the input query descriptors and stored train descriptors from the i-th image trainDescCollection[i]. compactResult : Parameter used when the mask (or masks) is not empty. If compactResult is false, the matches vector has the same size as queryDescriptors rows. If compactResult is true, the matches vector does not contain matches for fully masked-out query descriptors.		5								
features2d_DescriptorMatcher_create	CVAPI(ExceptionStatus) features2d_DescriptorMatcher_create(const char *descriptorMatcherType, cv::Ptr<cv::DescriptorMatcher> **returnValue) { BEGIN_WRAP const cv::Ptr<cv::DescriptorMatcher> ret = cv::DescriptorMatcher::create(descriptorMatcherType); *returnValue = new cv::Ptr<cv::DescriptorMatcher>(ret); END_WRAP }	cv::DescriptorMatcher::create()	classcv_1_1DescriptorMatcher.html	Creates a descriptor matcher of a given type with the default parameters (using default constructor).	descriptorMatcherType : Descriptor matcher type. Now the following matcher types are supported: BruteForce (it uses L2) BruteForce-L1 BruteForce-Hamming BruteForce-Hamming(2) FlannBased		1								
features2d_Ptr_DescriptorMatcher_get	CVAPI(ExceptionStatus) features2d_Ptr_DescriptorMatcher_get(cv::Ptr<cv::DescriptorMatcher> *ptr, cv::DescriptorMatcher **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::DescriptorMatcher	classcv_1_1DescriptorMatcher.html	Abstract base class for matching keypoint descriptors. It has two groups of match methods: for matching descriptors of an image with another image or with an image set.								•	
features2d_Ptr_DescriptorMatcher_delete	CVAPI(ExceptionStatus) features2d_Ptr_DescriptorMatcher_delete(cv::Ptr<cv::DescriptorMatcher> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::DescriptorMatcher	classcv_1_1DescriptorMatcher.html	Abstract base class for matching keypoint descriptors. It has two groups of match methods: for matching descriptors of an image with another image or with an image set.							•		
features2d_BFMatcher_new	CVAPI(ExceptionStatus) features2d_BFMatcher_new(int normType, int crossCheck, cv::BFMatcher **returnValue) { BEGIN_WRAP *returnValue = new cv::BFMatcher(normType, crossCheck != 0); END_WRAP }	cv::BFMatcher	classcv_1_1BFMatcher.html	Brute-force descriptor matcher. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets.						•			
features2d_BFMatcher_delete	CVAPI(ExceptionStatus) features2d_BFMatcher_delete(cv::BFMatcher *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::BFMatcher	classcv_1_1BFMatcher.html	Brute-force descriptor matcher. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets.							•		
features2d_BFMatcher_isMaskSupported	CVAPI(ExceptionStatus) features2d_BFMatcher_isMaskSupported(cv::BFMatcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isMaskSupported() ? 1 : 0; END_WRAP }	cv::BFMatcher::isMaskSupported()	classcv_1_1BFMatcher.html	Returns true if the descriptor matcher supports masking permissible matches.		Implements cv::DescriptorMatcher.	1								
features2d_Ptr_BFMatcher_get	CVAPI(ExceptionStatus) features2d_Ptr_BFMatcher_get(cv::Ptr<cv::BFMatcher> *ptr, cv::BFMatcher **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::BFMatcher	classcv_1_1BFMatcher.html	Brute-force descriptor matcher. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets.								•	
features2d_Ptr_BFMatcher_delete	CVAPI(ExceptionStatus) features2d_Ptr_BFMatcher_delete(cv::Ptr<cv::BFMatcher> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::BFMatcher	classcv_1_1BFMatcher.html	Brute-force descriptor matcher. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one. This descriptor matcher supports masking permissible matches of descriptor sets.							•		
features2d_FlannBasedMatcher_new	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_new(cv::Ptr<cv::flann::IndexParams> *indexParams, cv::Ptr<cv::flann::SearchParams> *searchParams, cv::FlannBasedMatcher **returnValue) { BEGIN_WRAP cv::Ptr<cv::flann::IndexParams> indexParamsPtr; cv::Ptr<cv::flann::SearchParams> searchParamsPtr; if (indexParams == nullptr) indexParamsPtr = cv::makePtr<cv::flann::KDTreeIndexParams>(); else indexParamsPtr = *indexParams;	cv::FlannBasedMatcher	classcv_1_1FlannBasedMatcher.html	Flann-based descriptor matcher. This matcher trains cv::flann::Index on a train descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large train collection than the brute force matcher. FlannBasedMatcher does not support masking permissible matches of descriptor sets because flann::Index does not support this. :								•	
features2d_FlannBasedMatcher_delete	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_delete(cv::FlannBasedMatcher *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::FlannBasedMatcher	classcv_1_1FlannBasedMatcher.html	Flann-based descriptor matcher. This matcher trains cv::flann::Index on a train descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large train collection than the brute force matcher. FlannBasedMatcher does not support masking permissible matches of descriptor sets because flann::Index does not support this. :								•	
features2d_FlannBasedMatcher_add	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_add(cv::FlannBasedMatcher *obj, cv::Mat **descriptors, int descriptorsSize) { BEGIN_WRAP std::vector<cv::Mat> descriptorsVal(descriptorsSize); for (int i = 0; i < descriptorsSize; i++) { descriptorsVal[i] = *(descriptors[i]); } obj->add(descriptorsVal); END_WRAP }	cv::FlannBasedMatcher::add()	classcv_1_1FlannBasedMatcher.html	Adds descriptors to train a CPU(trainDescCollection) or GPU(utrainDescCollection) descriptor collection.	descriptors : Descriptors to add. Each descriptors[i] is a set of descriptors from the same train image.	If the collection is not empty, the new descriptors are added to existing train descriptors.Reimplemented from cv::DescriptorMatcher.	1								

features2d_FlannBasedMatcher_clear	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_clear(cv::FlannBasedMatcher *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::FlannBasedMatcher::clear()	classcv_1_1FlannBasedMatcher.html	Clears the train descriptor collections.		Reimplemented from cv::DescriptorMatcher.	1									
features2d_FlannBasedMatcher_train	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_train(cv::FlannBasedMatcher *obj) { BEGIN_WRAP obj->train(); END_WRAP }	cv::FlannBasedMatcher::train()	classcv_1_1FlannBasedMatcher.html	Trains a descriptor matcher.		Trains a descriptor matcher (for example, the flann index). In all methods to match, the method train() is run every time before matching. Some descriptor matchers (for example, BruteForceMatcher) have an empty implementation of this method. Other matchers really train their inner structures (for example, FlannBasedMatcher trains flann::Index).Reimplemented from cv::DescriptorMatcher.	1									
features2d_FlannBasedMatcher_isMaskSupport	CVAPI(ExceptionStatus) features2d_FlannBasedMatcher_isMaskSupported(cv::FlannBasedMatcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isMaskSupported() ? 1 : 0; END_WRAP }	cv::FlannBasedMatcher::isMaskSupported()	classcv_1_1FlannBasedMatcher.html	Returns true if the descriptor matcher supports masking permissible matches.		Implements cv::DescriptorMatcher.	1									
features2d_Ptr_FlannBasedMatcher_get	CVAPI(ExceptionStatus) features2d_Ptr_FlannBasedMatcher_get(cv::Ptr<cv::FlannBasedMatcher> *ptr, cv::FlannBasedMatcher **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::FlannBasedMatcher	classcv_1_1FlannBasedMatcher.html	Flann-based descriptor matcher. This matcher trains cv::flann::Index on a train descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large train collection than the brute force matcher. FlannBasedMatcher does not support masking permissible matches of descriptor sets because flann::Index does not support this. :									•		
features2d_Ptr_FlannBasedMatcher_delete	CVAPI(ExceptionStatus) features2d_Ptr_FlannBasedMatcher_delete(cv::Ptr<cv::FlannBasedMatcher> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::FlannBasedMatcher	classcv_1_1FlannBasedMatcher.html	Flann-based descriptor matcher. This matcher trains cv::flann::Index on a train descriptor collection and calls its nearest search methods to find the best matches. So, this matcher may be faster when matching a large train collection than the brute force matcher. FlannBasedMatcher does not support masking permissible matches of descriptor sets because flann::Index does not support this. :								•		
features2d_Feature2D_detect_Mat1	CVAPI(ExceptionStatus) features2d_Feature2D_detect_Mat1(cv::Feature2D *detector, cv::Mat *image, std::vector<cv::KeyPoint> *keypoints, cv::Mat *mask) { BEGIN_WRAP detector->detect(*image, *keypoints, entity(mask)); END_WRAP }	cv::Feature2D::detect()	classcv_1_1Feature2D.html	Detects keypoints in an image (first variant) or image set (second variant).	image : Image. keypoints : The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] . mask : Mask specifying where to look for keypoints (optional). It must be a 8-bit integer matrix with non-zero values in the region of interest.		3								
features2d_Feature2D_detect_Mat2	CVAPI(ExceptionStatus) features2d_Feature2D_detect_Mat2(cv::Feature2D *detector, cv::Mat **images, int imageLength, std::vector<std::vector<cv::KeyPoint> > *keypoints, cv::Mat **mask) { BEGIN_WRAP std::vector<cv::Mat> imageVec(imageLength); std::vector<cv::Mat> maskVec; for (auto i = 0; i < imageLength; i++) imageVec.push_back(*images[i]); if (mask != nullptr) {	cv::Feature2D::detect()	classcv_1_1Feature2D.html	Detects keypoints in an image (first variant) or image set (second variant).	image : Image. keypoints : The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] . mask : Mask specifying where to look for keypoints (optional). It must be a 8-bit integer matrix with non-zero values in the region of interest.		3								
features2d_Feature2D_detect_InputArray	CVAPI(ExceptionStatus) features2d_Feature2D_detect_InputArray(cv::Feature2D *obj, cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, cv::Mat *mask) { BEGIN_WRAP obj->detect(*image, *keypoints, entity(mask)); END_WRAP }	cv::Feature2D::detect()	classcv_1_1Feature2D.html	Detects keypoints in an image (first variant) or image set (second variant).	image : Image. keypoints : The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] . mask : Mask specifying where to look for keypoints (optional). It must be a 8-bit integer matrix with non-zero values in the region of interest.		3								
features2d_Feature2D_compute1	CVAPI(ExceptionStatus) features2d_Feature2D_compute1(cv::Feature2D *obj, cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, cv::_OutputArray *descriptors) { BEGIN_WRAP obj->compute(*image, *keypoints, *descriptors); END_WRAP }	cv::Feature2D::compute()	classcv_1_1Feature2D.html	Computes the descriptors for a set of keypoints detected in an image (first variant) or image set (second variant).	image : Image. keypoints : Input collection of keypoints. Keypoints for which a descriptor cannot be computed are removed. Sometimes new keypoints can be added, for example: SIFT duplicates keypoint with several dominant orientations (for each orientation). descriptors : Computed descriptors. In the second variant of the method descriptors[i] are descriptors computed for a keypoints[i]. Row j is the keypoints (or keypoints[i]) is the descriptor for keypoint j-th keypoint.	Reimplemented in cv::xfeatures2d::DAISY.	3								
features2d_Feature2D_compute2	CVAPI(ExceptionStatus) features2d_Feature2D_compute2(cv::Feature2D *detector, cv::Mat **images, int imageLength, std::vector<std::vector<cv::KeyPoint> > *keypoints, cv::Mat **descriptors, int descriptorsLength) { BEGIN_WRAP std::vector<cv::Mat> imageVec(imageLength); std::vector<cv::Mat> descriptorsVec(descriptorsLength); for (auto i = 0; i < imageLength; i++) imageVec.push_back(*images[i]); for (auto i = 0; i < descriptorsLength; i++)	cv::Feature2D::compute()	classcv_1_1Feature2D.html	Computes the descriptors for a set of keypoints detected in an image (first variant) or image set (second variant).	image : Image. keypoints : Input collection of keypoints. Keypoints for which a descriptor cannot be computed are removed. Sometimes new keypoints can be added, for example: SIFT duplicates keypoint with several dominant orientations (for each orientation). descriptors : Computed descriptors. In the second variant of the method descriptors[i] are descriptors computed for a keypoints[i]. Row j is the keypoints (or keypoints[i]) is the descriptor for keypoint j-th keypoint.	Reimplemented in cv::xfeatures2d::DAISY.	3								
features2d_Feature2D_detectAndCompute	CVAPI(ExceptionStatus) features2d_Feature2D_detectAndCompute(cv::Feature2D *detector, cv::_InputArray *image, cv::_InputArray *mask, std::vector<cv::KeyPoint> *keypoints, cv::_OutputArray *descriptors, int useProvidedKeypoints) { BEGIN_WRAP detector->detectAndCompute(entity(image), entity(mask), *keypoints, *descriptors, useProvidedKeypoints != 0); END_WRAP }	cv::Feature2D::detectAndCompute()	classcv_1_1Feature2D.html	Detects keypoints and computes the descriptors			5								
features2d_Feature2D_descriptorSize	CVAPI(ExceptionStatus) features2d_Feature2D_descriptorSize(cv::Feature2D *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->descriptorSize(); END_WRAP }	cv::Feature2D::descriptorSize()	classcv_1_1Feature2D.html				1									
features2d_Feature2D_descriptorType	CVAPI(ExceptionStatus) features2d_Feature2D_descriptorType(cv::Feature2D *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->descriptorType(); END_WRAP }	cv::Feature2D::descriptorType()	classcv_1_1Feature2D.html				1									
features2d_Feature2D_defaultNorm	CVAPI(ExceptionStatus) features2d_Feature2D_defaultNorm(cv::Feature2D *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->defaultNorm(); END_WRAP }	cv::Feature2D::defaultNorm()	classcv_1_1Feature2D.html				1									

features2d_Feature2D_empty	CVAPI(ExceptionStatus) features2d_Feature2D_empty (cv::Feature2D *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::Feature2D::empty()	classcv_1_1Feature2D.html	Return true if detector object is empty.		Reimplemented from cv::Algorithm.	1									
features2d_Feature2D_write	CVAPI(ExceptionStatus) features2d_Feature2D_write(cv::Feature2D *obj, const char *fileName) { BEGIN_WRAP const cv::String fileNameString(fileName); obj->write(fileNameString); END_WRAP }	cv::Feature2D::write()	classcv_1_1Feature2D.html				1									
features2d_Feature2D_read	CVAPI(ExceptionStatus) features2d_Feature2D_read(cv::Feature2D *obj, const char *fileName) { BEGIN_WRAP obj->read(fileName); END_WRAP }	cv::Feature2D::read()	classcv_1_1Feature2D.html				1									
features2d_Feature2D_getDefaultName	CVAPI(ExceptionStatus) features2d_Feature2D_getDefaultName(cv::Feature2D *obj, std::string *returnValue) { BEGIN_WRAP returnValue->assign(obj->getDefaultName()); END_WRAP }	cv::Feature2D::getDefaultName()	classcv_1_1Feature2D.html	Returns the algorithm string identifier. This string is used as top level xml/yml node tag when the object is saved to a file or string.		Reimplemented from cv::Algorithm.Reimplemented in cv::AKAZE, cv::KAZE, cv::SimpleBlobDetector, cv::GFTTDetector, cv::AgastFeatureDetector, cv::FastFeatureDetector, cv::MSER, cv::ORB, cv::BRISK, cv::SIFT, and cv::AffineFeature.	1									
features2d_SIFT_create	CVAPI(ExceptionStatus) features2d_SIFT_create(int nfeatures, int nOctaveLayers, double contrastThreshold, double edgeThreshold, double sigma, cv::Ptr<cv::SIFT> **returnValue) { BEGIN_WRAP const auto ptr = cv::SIFT::create(nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma); *returnValue = clone(ptr); END_WRAP }	cv::SIFT::create()	classcv_1_1SIFT.html	NoteThe contrast threshold will be divided by nOctaveLayers when the filtering is applied. When nOctaveLayers is set to default and if you want to use the value used in D. Lowe paper, 0.03, set this argument to 0.09.	nfeatures : The number of best features to retain. The features are ranked by their scores (measured in SIFT algorithm as the local contrast) nOctaveLayers : The number of layers in each octave. 3 is the value used in D. Lowe paper. The number of octaves is computed automatically from the image resolution. contrastThreshold : The contrast threshold used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. edgeThreshold : The threshold used to filter out edge-like features. Note that the its meaning is different from the contrastThreshold, i.e. the larger the edgeThreshold, the less features are filtered out (more features are retained). sigma : The sigma of the Gaussian applied to the input image at the octave #0. If your image is captured with a weak camera with soft lenses, you might want to reduce the number.		5									
features2d_Ptr_SIFT_delete	CVAPI(ExceptionStatus) features2d_Ptr_SIFT_delete(cv::Ptr<cv::SIFT> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::SIFT	classcv_1_1SIFT.html	Class for extracting keypoints and computing descriptors using the Scale Invariant Feature Transform (SIFT) algorithm by D. Lowe [153] .								•				
features2d_Ptr_SIFT_get	CVAPI(ExceptionStatus) features2d_Ptr_SIFT_get(cv::Ptr<cv::SIFT> *ptr, cv::SIFT **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::SIFT	classcv_1_1SIFT.html	Class for extracting keypoints and computing descriptors using the Scale Invariant Feature Transform (SIFT) algorithm by D. Lowe [153] .									•			
features2d_BRISK_create1	CVAPI(ExceptionStatus) features2d_BRISK_create1(int thresh, int octaves, float patternScale, cv::Ptr<cv::BRISK> **returnValue) { BEGIN_WRAP const auto ptr = cv::BRISK::create(thresh, octaves, patternScale); *returnValue = clone(ptr); END_WRAP }	cv::BRISK::create()	classcv_1_1BRISK.html	The BRISK constructor.	thresh : AGAST detection threshold score. octaves : detection octaves. Use 0 to do single scale. patternScale : apply this scale to the pattern used for sampling the neighbourhood of a keypoint.		3									
features2d_BRISK_create2	CVAPI(ExceptionStatus) features2d_BRISK_create2(float *radiusList, int radiusListLength, int *numberList, int numberListLength, float dMax, float dMin, int *indexChange, int indexChangeLength, cv::Ptr<cv::BRISK> **returnValue) { BEGIN_WRAP const std::vector<float> radiusListVec(radiusList, radiusList + radiusListLength); const std::vector<int> numberListVec(numberList, numberList + numberListLength); std::vector<int> indexChangeVec; if (indexChange != nullptr) indexChangeVec = std::vector<int>(indexChange	radiusList : defines the radii (in pixels) where the samples around a keypoint are taken (for keypoint scale 1). numberList : defines the number of sampling points on the sampling circle. Must be the same size as radiusList.. dMax : threshold for the short pairings used for descriptor formation (in pixels for keypoint scale 1). dMin : threshold for the long pairings used for orientation determination (in pixels for keypoint scale 1). indexChange : index remapping of the bits.		5									
features2d_BRISK_create3	CVAPI(ExceptionStatus) features2d_BRISK_create3(int thresh, int octaves, float *radiusList, int radiusListLength, int *numberList, int numberListLength, float dMax, float dMin, int *indexChange, int indexChangeLength, cv::Ptr<cv::BRISK> **returnValue) { BEGIN_WRAP const std::vector<float> radiusListVec(radiusList, radiusList + radiusListLength); const std::vector<int> numberListVec(numberList, numberList + numberListLength); std::vector<int> indexChangeVec; if (indexChange != nullptr) indexChangeVec =	thresh : AGAST detection threshold score. octaves : detection octaves. Use 0 to do single scale. radiusList : defines the radii (in pixels) where the samples around a keypoint are taken (for keypoint scale 1). numberList : defines the number of sampling points on the sampling circle. Must be the same size as radiusList.. dMax : threshold for the short pairings used for descriptor formation (in pixels for keypoint scale 1). dMin : threshold for the long pairings used for orientation determination (in pixels for keypoint scale 1). indexChange : index remapping of the bits.		7									
features2d_Ptr_BRISK_delete	CVAPI(ExceptionStatus) features2d_Ptr_BRISK_delete(cv::Ptr<cv::BRISK> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::BRISK	classcv_1_1BRISK.html	Class implementing the BRISK keypoint detector and descriptor extractor, described in [139] .								•				
features2d_Ptr_BRISK_get	CVAPI(ExceptionStatus) features2d_Ptr_BRISK_get(cv::Ptr<cv::BRISK> *ptr, cv::BRISK **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::BRISK	classcv_1_1BRISK.html	Class implementing the BRISK keypoint detector and descriptor extractor, described in [139] .									•			
features2d_ORB_create	CVAPI(ExceptionStatus) features2d_ORB_create(int nFeatures, float scaleFactor, int nlevels, int edgeThreshold, int firstLevel, int wtaK, int scoreType, int patchSize, int fastThreshold, cv::Ptr<cv::ORB> **returnValue) { BEGIN_WRAP const auto ptr = cv::ORB::create(nFeatures, scaleFactor, nlevels, edgeThreshold, firstLevel, wtaK, static_cast<cv::ORB::ScoreType>(scoreType), patchSize, fastThreshold); *returnValue = clone(ptr); END_WRAP }	cv::ORB::create()	classcv_1_1ORB.html	The ORB constructor.	nfeatures : The maximum number of features to retain. scaleFactor : Pyramid decimation ratio, greater than 1. scaleFactor==2 means the classical pyramid, where each next level has 4x less pixels than the previous, but such a big scale factor will degrade feature matching scores dramatically. On the other hand, too close to 1 scale factor will mean that to cover certain scale range you will need more pyramid levels and so the speed will suffer. nlevels : The number of pyramid levels. The smallest level will have linear size equal to input_image_linear_size/pow(scaleFactor, nlevels - firstLevel). edgeThreshold : This is size of the border where the features are not detected. It should roughly match the patchSize parameter. firstLevel : The level of pyramid to put source image to. Previous layers are filled with upscaled source image. WTA_K : The number of points that produce each element of the oriented BRIEF descriptor. The default value 2 means the BRIEF where we take a random point pair and compare their brightnesses, so we get 0/1 response. Other possible values are 3 and 4. For example, 3 means that we take 3 random points (of		9									
features2d_Ptr_ORB_delete	CVAPI(ExceptionStatus) features2d_Ptr_ORB_delete(cv::Ptr<cv::ORB> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::ORB	classcv_1_1ORB.html	Class implementing the ORB (oriented BRIEF) keypoint detector and descriptor extractor. described in [206] . The algorithm uses FAST in pyramids to detect stable keypoints, selects the strongest features using FAST or Harris response, finds their orientation using first-order moments and computes the descriptors using BRIEF (where the coordinates of random point pairs (or k-tuples) are rotated according to the measured orientation).								•				

features2d_Ptr_ORB_get	CVAPI(ExceptionStatus) features2d_Ptr_ORB_get(cv::Ptr<cv::ORB> *ptr, cv::ORB **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ORB	classcv_1_1ORB.html	Class implementing the ORB (oriented BRIEF) keypoint detector and descriptor extractor. described in [206] . The algorithm uses FAST in pyramids to detect stable keypoints, selects the strongest features using FAST or Harris response, finds their orientation using first-order moments and computes the descriptors using BRIEF (where the coordinates of random point pairs (or k-tuples) are rotated according to the measured orientation).									•	
features2d_ORB_setMaxFeatures	CVAPI(ExceptionStatus) features2d_ORB_setMaxFeatures(cv::ORB *obj, int val) { BEGIN_WRAP obj->setMaxFeatures(val); END_WRAP }	cv::ORB::setMaxFeatures()	classcv_1_1ORB.html				1									
features2d_ORB_getMaxFeatures	CVAPI(ExceptionStatus) features2d_ORB_getMaxFeatures(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxFeatures(); END_WRAP }	cv::ORB::getMaxFeatures()	classcv_1_1ORB.html				1									
features2d_ORB_setScaleFactor	CVAPI(ExceptionStatus) features2d_ORB_setScaleFactor(cv::ORB *obj, double val) { BEGIN_WRAP obj->setScaleFactor(val); END_WRAP }	cv::ORB::setScaleFactor()	classcv_1_1ORB.html				1									
features2d_ORB_getScaleFactor	CVAPI(ExceptionStatus) features2d_ORB_getScaleFactor(cv::ORB *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getScaleFactor(); END_WRAP }	cv::ORB::getScaleFactor()	classcv_1_1ORB.html				1									
features2d_ORB_setNLevels	CVAPI(ExceptionStatus) features2d_ORB_setNLevels(cv::ORB *obj, int val) { BEGIN_WRAP obj->setNLevels(val); END_WRAP }	cv::ORB::setNLevels()	classcv_1_1ORB.html				1									
features2d_ORB_getNLevels	CVAPI(ExceptionStatus) features2d_ORB_getNLevels(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNLevels(); END_WRAP }	cv::ORB::getNLevels()	classcv_1_1ORB.html				1									
features2d_ORB_setEdgeThreshold	CVAPI(ExceptionStatus) features2d_ORB_setEdgeThreshold(cv::ORB *obj, int val) { BEGIN_WRAP obj->setEdgeThreshold(val); END_WRAP }	cv::ORB::setEdgeThreshold()	classcv_1_1ORB.html				1									
features2d_ORB_getEdgeThreshold	CVAPI(ExceptionStatus) features2d_ORB_getEdgeThreshold(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getEdgeThreshold(); END_WRAP }	cv::ORB::getEdgeThreshold()	classcv_1_1ORB.html				1									
features2d_ORB_setFirstLevel	CVAPI(ExceptionStatus) features2d_ORB_setFirstLevel(cv::ORB *obj, int val) { BEGIN_WRAP obj->setFirstLevel(val); END_WRAP }	cv::ORB::setFirstLevel()	classcv_1_1ORB.html				1									
features2d_ORB_getFirstLevel	CVAPI(ExceptionStatus) features2d_ORB_getFirstLevel(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getFirstLevel(); END_WRAP }	cv::ORB::getFirstLevel()	classcv_1_1ORB.html				1									
features2d_ORB_setWTA_K	CVAPI(ExceptionStatus) features2d_ORB_setWTA_K(cv::ORB *obj, int val) { BEGIN_WRAP obj->setWTA_K(val); END_WRAP }	cv::ORB::setWTA_K()	classcv_1_1ORB.html				1									
features2d_ORB_getWTA_K	CVAPI(ExceptionStatus) features2d_ORB_getWTA_K(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getWTA_K(); END_WRAP }	cv::ORB::getWTA_K()	classcv_1_1ORB.html				1									
features2d_ORB_setScoreType	CVAPI(ExceptionStatus) features2d_ORB_setScoreType(cv::ORB *obj, int val) { BEGIN_WRAP obj->setScoreType(static_cast<cv::ORB::ScoreType>(val)); END_WRAP }	cv::ORB::setScoreType()	classcv_1_1ORB.html				1									
features2d_ORB_getScoreType	CVAPI(ExceptionStatus) features2d_ORB_getScoreType(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getScoreType()); END_WRAP }	cv::ORB::getScoreType()	classcv_1_1ORB.html				1									
features2d_ORB_setPatchSize	CVAPI(ExceptionStatus) features2d_ORB_setPatchSize(cv::ORB *obj, int val) { BEGIN_WRAP obj->setPatchSize(val); END_WRAP }	cv::ORB::setPatchSize()	classcv_1_1ORB.html				1									
features2d_ORB_getPatchSize	CVAPI(ExceptionStatus) features2d_ORB_getPatchSize(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getPatchSize(); END_WRAP }	cv::ORB::getPatchSize()	classcv_1_1ORB.html				1									
features2d_ORB_setFastThreshold	CVAPI(ExceptionStatus) features2d_ORB_setFastThreshold(cv::ORB *obj, int val) { BEGIN_WRAP obj->setFastThreshold(val); END_WRAP }	cv::ORB::setFastThreshold()	classcv_1_1ORB.html				1									
features2d_ORB_getFastThreshold	CVAPI(ExceptionStatus) features2d_ORB_getFastThreshold(cv::ORB *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getFastThreshold(); END_WRAP }	cv::ORB::getFastThreshold()	classcv_1_1ORB.html				1									

features2d_MSER_create	CVAPI(ExceptionStatus) features2d_MSER_create(int delta, int minArea, int maxArea, double maxVariation, double minDiversity, int maxEvolution, double areaThreshold, double minMargin, int edgeBlurSize, cv::Ptr<cv::MSER> **returnValue) { BEGIN_WRAP const auto ptr = cv::MSER::create(delta, minArea, maxArea, maxVariation, minDiversity, maxEvolution, areaThreshold, minMargin, edgeBlurSize); *returnValue = clone(ptr); END_WRAP }	cv::MSER::create()	classcv_1_1MSER.html	Full constructor for MSER detector.	delta : it compares \((size_{i}-size_{i-delta})/size_{i-delta}\) min_area : prune the area which smaller than minArea max_area : prune the area which bigger than maxArea max_variation : prune the area have similar size to its children min_diversity : for color image, trace back to cut off mser with diversity less than min_diversity max_evolution : for color image, the evolution steps area_threshold : for color image, the area threshold to cause re-initialize min_margin : for color image, ignore too small margin edge_blur_size : for color image, the aperture size for edge blur		9								
features2d_Ptr_MSER_delete	CVAPI(ExceptionStatus) features2d_Ptr_MSER_delete(cv::Ptr<cv::MSER> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::MSER	classcv_1_1MSER.html	Maximally stable extremal region extractor. The class encapsulates all the parameters of the MSER extraction algorithm (see wiki article). there are two different implementation of MSER: one for grey image, one for color image the grey image algorithm is taken from: [185] ; the paper claims to be faster than union-find method; it actually get 1.5~2m/s on my centrino L7200 1.2GHz laptop. the color image algorithm is taken from: [80] ; it should be much slower than grey image method (3~4 times) (Python) A complete example showing the use of the MSER detector can be found at samples/python/mser.py							•		
features2d_Ptr_MSER_get	CVAPI(ExceptionStatus) features2d_Ptr_MSER_get(cv::Ptr<cv::MSER> *ptr, cv::MSER **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::MSER	classcv_1_1MSER.html	Maximally stable extremal region extractor. The class encapsulates all the parameters of the MSER extraction algorithm (see wiki article). there are two different implementation of MSER: one for grey image, one for color image the grey image algorithm is taken from: [185] ; the paper claims to be faster than union-find method; it actually get 1.5~2m/s on my centrino L7200 1.2GHz laptop. the color image algorithm is taken from: [80] ; it should be much slower than grey image method (3~4 times) (Python) A complete example showing the use of the MSER detector can be found at samples/python/mser.py							•		
features2d_MSER_detectRegions	CVAPI(ExceptionStatus) features2d_MSER_detectRegions(cv::MSER *obj, cv::_InputArray *image, std::vector<std::vector<cv::Point> > *msers, std::vector<cv::Rect> *bboxes) { BEGIN_WRAP obj->detectRegions(*image, *msers, *bboxes); END_WRAP }	cv::MSER::detectRegions()	classcv_1_1MSER.html	Detect MSER regions.	image : input image (8UC1, 8UC3 or 8UC4, must be greater or equal than 3x3) msers : resulting list of point sets bboxes : resulting bounding boxes		3								
features2d_MSER_setDelta	CVAPI(ExceptionStatus) features2d_MSER_setDelta(cv::MSER *obj, int delta) { BEGIN_WRAP obj->setDelta(delta); END_WRAP }	cv::MSER::setDelta()	classcv_1_1MSER.html				1									
features2d_MSER_getDelta	CVAPI(ExceptionStatus) features2d_MSER_getDelta(cv::MSER *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getDelta(); END_WRAP }	cv::MSER::getDelta()	classcv_1_1MSER.html				1									
features2d_MSER_setMinArea	CVAPI(ExceptionStatus) features2d_MSER_setMinArea(cv::MSER *obj, int minArea) { BEGIN_WRAP obj->setMinArea(minArea); END_WRAP }	cv::MSER::setMinArea()	classcv_1_1MSER.html				1									
features2d_MSER_getMinArea	CVAPI(ExceptionStatus) features2d_MSER_getMinArea(cv::MSER *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMinArea(); END_WRAP }	cv::MSER::getMinArea()	classcv_1_1MSER.html				1									
features2d_MSER_setMaxArea	CVAPI(ExceptionStatus) features2d_MSER_setMaxArea(cv::MSER *obj, int maxArea) { BEGIN_WRAP obj->setMaxArea(maxArea); END_WRAP }	cv::MSER::setMaxArea()	classcv_1_1MSER.html				1									
features2d_MSER_getMaxArea	CVAPI(ExceptionStatus) features2d_MSER_getMaxArea(cv::MSER *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxArea(); END_WRAP }	cv::MSER::getMaxArea()	classcv_1_1MSER.html				1									
features2d_MSER_setPass2Only	CVAPI(ExceptionStatus) features2d_MSER_setPass2Only(cv::MSER *obj, int f) { BEGIN_WRAP obj->setPass2Only(f != 0); END_WRAP }	cv::MSER::setPass2Only()	classcv_1_1MSER.html				1									
features2d_MSER_getPass2Only	CVAPI(ExceptionStatus) features2d_MSER_getPass2Only(cv::MSER *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getPass2Only() ? 1 : 0; END_WRAP }	cv::MSER::getPass2Only()	classcv_1_1MSER.html				1									
features2d_FAST1	CVAPI(ExceptionStatus) features2d_FAST1(cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, int threshold, int nonmaxSupression) { BEGIN_WRAP cv::FAST(*image, *keypoints, threshold, nonmaxSupression != 0); END_WRAP }	cv::FAST()	group__features2d__main.html	Detects corners using the FAST algorithm.	image : grayscale image where keypoints (corners) are detected. keypoints : keypoints detected on the image. threshold : threshold on difference between intensity of the central pixel and pixels of a circle around this pixel. nonmaxSuppression : if true, non-maximum suppression is applied to detected corners (keypoints). type : one of the three neighborhoods as defined in the paper: FastFeatureDetector::TYPE_9_16, FastFeatureDetector::TYPE_7_12, FastFeatureDetector::TYPE_5_8	Detects corners using the FAST algorithm by [205] .Notaln Python API, types are given as cv.FAST_FEATURE_DETECTOR_TYPE_5_8 cv.FAST_FEATURE_DETECTOR_TYPE_7_12 and cv.FAST_FEATURE_DETECTOR_TYPE_9_16. For corner detection, use cv.FAST.detect() method.	4							
features2d_FAST2	CVAPI(ExceptionStatus) features2d_FAST2(cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, int threshold, int nonmaxSupression, int type) { BEGIN_WRAP cv::FAST(*image, *keypoints, threshold, nonmaxSupression != 0, static_cast<cv::FastFeatureDetector::DetectorType>(type)); END_WRAP }	cv::FAST()	group__features2d__main.html	Detects corners using the FAST algorithm.	image : grayscale image where keypoints (corners) are detected. keypoints : keypoints detected on the image. threshold : threshold on difference between intensity of the central pixel and pixels of a circle around this pixel. nonmaxSuppression : if true, non-maximum suppression is applied to detected corners (keypoints). type : one of the three neighborhoods as defined in the paper: FastFeatureDetector::TYPE_9_16, FastFeatureDetector::TYPE_7_12, FastFeatureDetector::TYPE_5_8	Detects corners using the FAST algorithm by [205] .Notaln Python API, types are given as cv.FAST_FEATURE_DETECTOR_TYPE_5_8 cv.FAST_FEATURE_DETECTOR_TYPE_7_12 and cv.FAST_FEATURE_DETECTOR_TYPE_9_16. For corner detection, use cv.FAST.detect() method.	5							
features2d_FastFeatureDetector_create	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_create(int threshold, int nonmaxSuppression, cv::Ptr<cv::FastFeatureDetector> **returnValue) { BEGIN_WRAP const auto ptr = cv::FastFeatureDetector::create(threshold, nonmaxSuppression != 0); *returnValue = clone(ptr); END_WRAP }	cv::FastFeatureDetector::create()	classcv_1_1FastFeatureDetector.html				2									

features2d_Ptr_FastFeatureDetector_delete	CVAPI(ExceptionStatus) features2d_Ptr_FastFeatureDetector_delete(cv::Ptr<cv::FastFeatureDetector> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::FastFeatureDetector	classcv_1_1FastFeatureDetector.html	Wrapping class for feature detection using the FAST method. :									•			
features2d_Ptr_FastFeatureDetector_get	CVAPI(ExceptionStatus) features2d_Ptr_FastFeatureDetector_get(cv::Ptr<cv::FastFeatureDetector> *ptr, cv::FastFeatureDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::FastFeatureDetector	classcv_1_1FastFeatureDetector.html	Wrapping class for feature detection using the FAST method. :										•		
features2d_FastFeatureDetector_setThreshold	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_setThreshold(cv::FastFeatureDetector *obj, int threshold) { BEGIN_WRAP obj->setThreshold(threshold); END_WRAP }	cv::FastFeatureDetector::setThreshold()	classcv_1_1FastFeatureDetector.html					1								
features2d_FastFeatureDetector_getThreshold	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_getThreshold(cv::FastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::FastFeatureDetector::getThreshold()	classcv_1_1FastFeatureDetector.html					1								
features2d_FastFeatureDetector_setNonmaxSu	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_setNonmaxSuppression(cv::FastFeatureDetector *obj, int f) { BEGIN_WRAP obj->setNonmaxSuppression(f != 0); END_WRAP }	cv::FastFeatureDetector::setNonmaxSuppression()	classcv_1_1FastFeatureDetector.html					1								
features2d_FastFeatureDetector_getNonmaxSu	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_getNonmaxSuppression(cv::FastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNonmaxSuppression() ? 1 : 0; END_WRAP }	cv::FastFeatureDetector::getNonmaxSuppression()	classcv_1_1FastFeatureDetector.html					1								
features2d_FastFeatureDetector_setType	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_setType(cv::FastFeatureDetector *obj, int type) { BEGIN_WRAP obj->setType(static_cast<cv::FastFeatureDetector::DetectorType>(type)); END_WRAP }	cv::FastFeatureDetector::setType()	classcv_1_1FastFeatureDetector.html					1								
features2d_FastFeatureDetector_getType	CVAPI(ExceptionStatus) features2d_FastFeatureDetector_getType(cv::FastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getType()); END_WRAP }	cv::FastFeatureDetector::getType()	classcv_1_1FastFeatureDetector.html					1								
features2d_AGAST	CVAPI(ExceptionStatus) features2d_AGAST(cv::_InputArray *image, std::vector<cv::KeyPoint> *keypoints, int threshold, int nonmaxSuppression, int type) { BEGIN_WRAP cv::AGAST(entity(image), *keypoints, threshold, nonmaxSuppression != 0, static_cast<cv::AgastFeatureDetector::DetectorType>(type)); END_WRAP }	cv::AGAST()	group__features2d__main.html	Detects corners using the AGAST algorithm.	image : grayscale image where keypoints (corners) are detected. keypoints : keypoints detected on the image. threshold : threshold on difference between intensity of the central pixel and pixels of a circle around this pixel. nonmaxSuppression : if true, non-maximum suppression is applied to detected corners (keypoints). type : one of the four neighborhoods as defined in the paper: AgastFeatureDetector::AGAST_5_8, AgastFeatureDetector::AGAST_7_12d, AgastFeatureDetector::AGAST_7_12s, AgastFeatureDetector::OAST_9_16	For non-Intel platforms, there is a tree optimised variant of AGAST with same numerical results. The 32-bit binary tree tables were generated automatically from original code using perl script. The perl script and examples of tree generation are placed in features2d/doc folder. Detects corners using the AGAST algorithm by [159] .	5							
features2d_AgastFeatureDetector_create	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_create(int threshold, int nonmaxSuppression, int type, cv::Ptr<cv::AgastFeatureDetector> **returnValue) { BEGIN_WRAP const auto ptr = cv::AgastFeatureDetector::create(threshold, nonmaxSuppression != 0, static_cast<cv::AgastFeatureDetector::DetectorType>(type)); *returnValue = clone(ptr); END_WRAP }	cv::AgastFeatureDetector::create()	classcv_1_1AgastFeatureDetector.html					3								
features2d_Ptr_AgastFeatureDetector_delete	CVAPI(ExceptionStatus) features2d_Ptr_AgastFeatureDetector_delete(cv::Ptr<cv::AgastFeatureDetector> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::AgastFeatureDetector	classcv_1_1AgastFeatureDetector.html	Wrapping class for feature detection using the AGAST method. :										•		
features2d_Ptr_AgastFeatureDetector_get	CVAPI(ExceptionStatus) features2d_Ptr_AgastFeatureDetector_get(cv::Ptr<cv::AgastFeatureDetector> *ptr, cv::AgastFeatureDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::AgastFeatureDetector	classcv_1_1AgastFeatureDetector.html	Wrapping class for feature detection using the AGAST method. :										•		
features2d_AgastFeatureDetector_setThreshol	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_setThreshold(cv::AgastFeatureDetector *obj, int val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::AgastFeatureDetector::setThreshold()	classcv_1_1AgastFeatureDetector.html					1								
features2d_AgastFeatureDetector_getThreshol	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_getThreshold(cv::AgastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::AgastFeatureDetector::getThreshold()	classcv_1_1AgastFeatureDetector.html					1								
features2d_AgastFeatureDetector_setNonmax	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_setNonmaxSuppression(cv::AgastFeatureDetector *obj, int val) { BEGIN_WRAP obj->setNonmaxSuppression(val != 0); END_WRAP }	cv::AgastFeatureDetector::setNonmaxSuppression()	classcv_1_1AgastFeatureDetector.html					1								
features2d_AgastFeatureDetector_getNonmax	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_getNonmaxSuppression(cv::AgastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNonmaxSuppression() ? 1 : 0; END_WRAP }	cv::AgastFeatureDetector::getNonmaxSuppression()	classcv_1_1AgastFeatureDetector.html					1								

features2d_AgastFeatureDetector_setType	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_setType(cv::AgastFeatureDetector *obj, int val) { BEGIN_WRAP obj->setType(static_cast<cv::AgastFeatureDetector::DetectorType>(val)); END_WRAP }	cv::AgastFeatureDetector::setType()	classcv_1_1AgastFeatureDetector.html				1									
features2d_AgastFeatureDetector_getType	CVAPI(ExceptionStatus) features2d_AgastFeatureDetector_getType(cv::AgastFeatureDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getType()); END_WRAP }	cv::AgastFeatureDetector::getType()	classcv_1_1AgastFeatureDetector.html				1									
features2d_GFTTDetector_create	CVAPI(ExceptionStatus) features2d_GFTTDetector_create(int maxCorners, double qualityLevel, double minDistance, int blockSize, int useHarrisDetector, double k, cv::Ptr<cv::GFTTDetector> **returnValue) { BEGIN_WRAP const auto ptr = cv::GFTTDetector::create(maxCorners, qualityLevel, minDistance, blockSize, useHarrisDetector != 0, k); *returnValue = clone(ptr); END_WRAP }	cv::GFTTDetector::create()	classcv_1_1GFTTDetector.html				6									
features2d_Ptr_GFTTDetector_get	CVAPI(ExceptionStatus) features2d_Ptr_GFTTDetector_get(cv::Ptr<cv::GFTTDetector> *ptr, cv::GFTTDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::GFTTDetector	classcv_1_1GFTTDetector.html	Wrapping class for feature detection using the goodFeaturesToTrack function. :									•		
features2d_Ptr_GFTTDetector_delete	CVAPI(ExceptionStatus) features2d_Ptr_GFTTDetector_delete(cv::Ptr<cv::GFTTDetector> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::GFTTDetector	classcv_1_1GFTTDetector.html	Wrapping class for feature detection using the goodFeaturesToTrack function. :								•			
features2d_GFTTDetector_setMaxFeatures	CVAPI(ExceptionStatus) features2d_GFTTDetector_setMaxFeatures(cv::GFTTDetector *obj, int maxFeatures) { BEGIN_WRAP obj->setMaxFeatures(maxFeatures); END_WRAP }	cv::GFTTDetector::setMaxFeatures()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getMaxFeatures	CVAPI(ExceptionStatus) features2d_GFTTDetector_getMaxFeatures(cv::GFTTDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxFeatures(); END_WRAP }	cv::GFTTDetector::getMaxFeatures()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_setQualityLevel	CVAPI(ExceptionStatus) features2d_GFTTDetector_setQualityLevel(cv::GFTTDetector *obj, double qlevel) { BEGIN_WRAP obj->setQualityLevel(qlevel); END_WRAP }	cv::GFTTDetector::setQualityLevel()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getQualityLevel	CVAPI(ExceptionStatus) features2d_GFTTDetector_getQualityLevel(cv::GFTTDetector *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getQualityLevel(); END_WRAP }	cv::GFTTDetector::getQualityLevel()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_setMinDistance	CVAPI(ExceptionStatus) features2d_GFTTDetector_setMinDistance(cv::GFTTDetector *obj, double minDistance) { BEGIN_WRAP obj->setMinDistance(minDistance); END_WRAP }	cv::GFTTDetector::setMinDistance()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getMinDistance	CVAPI(ExceptionStatus) features2d_GFTTDetector_getMinDistance(cv::GFTTDetector *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMinDistance(); END_WRAP }	cv::GFTTDetector::getMinDistance()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_setBlockSize	CVAPI(ExceptionStatus) features2d_GFTTDetector_setBlockSize(cv::GFTTDetector *obj, int blockSize) { BEGIN_WRAP obj->setBlockSize(blockSize); END_WRAP }	cv::GFTTDetector::setBlockSize()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getBlockSize	CVAPI(ExceptionStatus) features2d_GFTTDetector_getBlockSize(cv::GFTTDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getBlockSize(); END_WRAP }	cv::GFTTDetector::getBlockSize()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_setHarrisDetector	CVAPI(ExceptionStatus) features2d_GFTTDetector_setHarrisDetector(cv::GFTTDetector *obj, int val) { BEGIN_WRAP obj->setHarrisDetector(val != 0); END_WRAP }	cv::GFTTDetector::setHarrisDetector()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getHarrisDetector	CVAPI(ExceptionStatus) features2d_GFTTDetector_getHarrisDetector(cv::GFTTDetector *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getHarrisDetector() ? 1 : 0; END_WRAP }	cv::GFTTDetector::getHarrisDetector()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_setK	CVAPI(ExceptionStatus) features2d_GFTTDetector_setK(cv::GFTTDetector *obj, double k) { BEGIN_WRAP obj->setK(k); END_WRAP }	cv::GFTTDetector::setK()	classcv_1_1GFTTDetector.html				1									
features2d_GFTTDetector_getK	CVAPI(ExceptionStatus) features2d_GFTTDetector_getK(cv::GFTTDetector *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getK(); END_WRAP }	cv::GFTTDetector::getK()	classcv_1_1GFTTDetector.html				1									

features2d_SimpleBlobDetector_create	CVAPI(ExceptionStatus) features2d_SimpleBlobDetector_create(SimpleBlobDetector_Params *p, cv::Ptr<cv::SimpleBlobDetector> **returnValue) { BEGIN_WRAP cv::SimpleBlobDetector::Params p2; if (p != nullptr) { p2.thresholdStep = p->thresholdStep; p2.minThreshold = p->minThreshold; p2.maxThreshold = p->maxThreshold; p2.minRepeatability = static_cast<size_t>(p->minRepeatability); p2.minDistBetweenBlobs = p->minDistBetweenBlobs; p2.filterByColor = p->	cv::SimpleBlobDetector::create()	classcv_1_1SimpleBlobDetector.html				1								
features2d_Ptr_SimpleBlobDetector_delete	CVAPI(ExceptionStatus) features2d_Ptr_SimpleBlobDetector_delete(cv::Ptr<cv::SimpleBlobDetector> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::SimpleBlobDetector	classcv_1_1SimpleBlobDetector.html	Class for extracting blobs from an image. : The class implements a simple algorithm for extracting blobs from an image: Convert the source image to binary images by applying thresholding with several thresholds from minThreshold (inclusive) to maxThreshold (exclusive) with distance thresholdStep between neighboring thresholds. Extract connected components from every binary image by findContours and calculate their centers. Group centers from several binary images by their coordinates. Close centers form one group that corresponds to one blob, which is controlled by the minDistBetweenBlobs parameter. From the groups, estimate final centers of blobs and their radiuses and return as locations and sizes of keypoints. This class performs several filtrations of returned blobs. You should set filterBy* to true/false to turn on/off corresponding filtration. Available filtrations: By color. This filter compares the intensity of a binary image at the center of a blob to blobColor. If they differ, the blob is filtered out. Use blobColor = 0 to extract dark blobs and blobColor							•		
features2d_Ptr_SimpleBlobDetector_get	CVAPI(ExceptionStatus) features2d_Ptr_SimpleBlobDetector_get(cv::Ptr<cv::SimpleBlobDetector> *ptr, cv::SimpleBlobDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::SimpleBlobDetector	classcv_1_1SimpleBlobDetector.html	Class for extracting blobs from an image. : The class implements a simple algorithm for extracting blobs from an image: Convert the source image to binary images by applying thresholding with several thresholds from minThreshold (inclusive) to maxThreshold (exclusive) with distance thresholdStep between neighboring thresholds. Extract connected components from every binary image by findContours and calculate their centers. Group centers from several binary images by their coordinates. Close centers form one group that corresponds to one blob, which is controlled by the minDistBetweenBlobs parameter. From the groups, estimate final centers of blobs and their radiuses and return as locations and sizes of keypoints. This class performs several filtrations of returned blobs. You should set filterBy* to true/false to turn on/off corresponding filtration. Available filtrations: By color. This filter compares the intensity of a binary image at the center of a blob to blobColor. If they differ, the blob is filtered out. Use blobColor = 0 to extract dark blobs and blobColor								•	
features2d_KAZE_create	CVAPI(ExceptionStatus) features2d_KAZE_create(int extended, int upright, float threshold, int nOctaves, int nOctaveLayers, int diffusivity, cv::Ptr<cv::KAZE> **returnValue) { BEGIN_WRAP const auto ptr = cv::KAZE::create(extended != 0, upright != 0, threshold, nOctaves, nOctaveLayers, static_cast<cv::KAZE::DiffusivityType>(diffusivity)); *returnValue = clone(ptr); END_WRAP }	cv::KAZE::create()	classcv_1_1KAZE.html	The KAZE constructor.	extended : Set to enable extraction of extended (128-byte) descriptor. upright : Set to enable use of upright descriptors (non rotation-invariant). threshold : Detector response threshold to accept point nOctaves : Maximum octave evolution of the image nOctaveLayers : Default number of sublevels per scale level diffusivity : Diffusivity type. DIFF_PM_G1, DIFF_PM_G2, DIFF_WEICKERT or DIFF_CHARBONNIER		6							
features2d_Ptr_KAZE_delete	CVAPI(ExceptionStatus) features2d_Ptr_KAZE_delete(cv::Ptr<cv::KAZE> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::KAZE	classcv_1_1KAZE.html	Class implementing the KAZE keypoint detector and descriptor extractor, described in [10] . NoteAKAZE descriptor can only be used with KAZE or AKAZE keypoints .. [ABD12] KAZE Features. Pablo F. Alcantarilla, Adrien Bartoli and Andrew J. Davison. In European Conference on Computer Vision (ECCV), Florenze, Italy, October 2012.							•		
features2d_Ptr_KAZE_get	CVAPI(ExceptionStatus) features2d_Ptr_KAZE_get(cv::Ptr<cv::KAZE> *ptr, cv::KAZE **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::KAZE	classcv_1_1KAZE.html	Class implementing the KAZE keypoint detector and descriptor extractor, described in [10] . NoteAKAZE descriptor can only be used with KAZE or AKAZE keypoints .. [ABD12] KAZE Features. Pablo F. Alcantarilla, Adrien Bartoli and Andrew J. Davison. In European Conference on Computer Vision (ECCV), Florenze, Italy, October 2012.								•	
features2d_KAZE_setDiffusivity	CVAPI(ExceptionStatus) features2d_KAZE_setDiffusivity(cv::KAZE *obj, int val) { BEGIN_WRAP obj->setDiffusivity(static_cast<cv::KAZE::DiffusivityType>(val)); END_WRAP }	cv::KAZE::setDiffusivity()	classcv_1_1KAZE.html				1									
features2d_KAZE_getDiffusivity	CVAPI(ExceptionStatus) features2d_KAZE_getDiffusivity(cv::KAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getDiffusivity()); END_WRAP }	cv::KAZE::getDiffusivity()	classcv_1_1KAZE.html				1									
features2d_KAZE_setExtended	CVAPI(ExceptionStatus) features2d_KAZE_setExtended(cv::KAZE *obj, int val) { BEGIN_WRAP obj->setExtended(val != 0); END_WRAP }	cv::KAZE::setExtended()	classcv_1_1KAZE.html				1									
features2d_KAZE_getExtended	CVAPI(ExceptionStatus) features2d_KAZE_getExtended(cv::KAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getExtended() ? 1 : 0; END_WRAP }	cv::KAZE::getExtended()	classcv_1_1KAZE.html				1									
features2d_KAZE_setNOctaveLayers	CVAPI(ExceptionStatus) features2d_KAZE_setNOctaveLayers(cv::KAZE *obj, int val) { BEGIN_WRAP obj->setNOctaveLayers(val); END_WRAP }	cv::KAZE::setNOctaveLayers()	classcv_1_1KAZE.html				1									
features2d_KAZE_getNOctaveLayers	CVAPI(ExceptionStatus) features2d_KAZE_getNOctaveLayers(cv::KAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaveLayers(); END_WRAP }	cv::KAZE::getNOctaveLayers()	classcv_1_1KAZE.html				1									
features2d_KAZE_setNOctaves	CVAPI(ExceptionStatus) features2d_KAZE_setNOctaves(cv::KAZE *obj, int val) { BEGIN_WRAP obj->setNOctaves(val); END_WRAP }	cv::KAZE::setNOctaves()	classcv_1_1KAZE.html				1									
features2d_KAZE_getNOctaves	CVAPI(ExceptionStatus) features2d_KAZE_getNOctaves(cv::KAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaves(); END_WRAP }	cv::KAZE::getNOctaves()	classcv_1_1KAZE.html				1									

features2d_KAZE_setThreshold	CVAPI(ExceptionStatus) features2d_KAZE_setThreshold(cv::KAZE *obj, double val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::KAZE::setThreshold()	classcv_1_1KAZE.html				1										
features2d_KAZE_getThreshold	CVAPI(ExceptionStatus) features2d_KAZE_getThreshold(cv::KAZE *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::KAZE::getThreshold()	classcv_1_1KAZE.html				1										
features2d_KAZE_setUpright	CVAPI(ExceptionStatus) features2d_KAZE_setUpright(cv::KAZE *obj, int val) { BEGIN_WRAP obj->setUpright(val != 0); END_WRAP }	cv::KAZE::setUpright()	classcv_1_1KAZE.html				1										
features2d_KAZE_getUpright	CVAPI(ExceptionStatus) features2d_KAZE_getUpright(cv::KAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getUpright() ? 1 : 0; END_WRAP }	cv::KAZE::getUpright()	classcv_1_1KAZE.html				1										
features2d_AKAZE_create	CVAPI(ExceptionStatus) features2d_AKAZE_create(int descriptor_type, int descriptor_size, int descriptor_channels, float threshold, int nOctaves, int nOctaveLayers, int diffusivity, cv::Ptr<cv::AKAZE> **returnValue) { BEGIN_WRAP const auto ptr = cv::AKAZE::create(static_cast<cv::AKAZE::DescriptorType>(descriptor_type), descriptor_size, descriptor_channels, threshold, nOctaves, nOctaveLayers, static_cast<cv::KAZE::DiffusivityType>(diffusivity)); *returnValue = clone(ptr); END_WRAP }	cv::AKAZE::create()	classcv_1_1AKAZE.html	The AKAZE constructor.	descriptor_type : Type of the extracted descriptor: DESCRIPTOR_KAZE, DESCRIPTOR_KAZE_UPRIGHT, DESCRIPTOR_MLDB or DESCRIPTOR_MLDB_UPRIGHT. descriptor_size : Size of the descriptor in bits. 0 -> Full size descriptor_channels : Number of channels in the descriptor (1, 2, 3) threshold : Detector response threshold to accept point nOctaves : Maximum octave evolution of the image nOctaveLayers : Default number of sublevels per scale level diffusivity : Diffusivity type. DIFF_PM_G1, DIFF_PM_G2, DIFF_WEICKERT or DIFF_CHARBONNIER		7								
features2d_Ptr_AKAZE_delete	CVAPI(ExceptionStatus) features2d_Ptr_AKAZE_delete(cv::Ptr<cv::AKAZE> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::AKAZE	classcv_1_1AKAZE.html	Class implementing the AKAZE keypoint detector and descriptor extractor, described in [9]. AKAZE descriptors can only be used with KAZE or AKAZE keypoints. This class is thread-safe. NoteWhen you need descriptors use Feature2D::detectAndCompute, which provides better performance. When using Feature2D::detect followed by Feature2D::compute scale space pyramid is computed twice. AKAZE implements T-API. When image is passed as UMat some parts of the algorithm will use OpenCL. [ANB13] Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. Pablo F. Alcantarilla, Jesús Nuevo and Adrien Bartoli. In British Machine Vision Conference (BMVC), Bristol, UK, September 2013.							•		
features2d_Ptr_AKAZE_get	CVAPI(ExceptionStatus) features2d_Ptr_AKAZE_get(cv::Ptr<cv::AKAZE> *ptr, cv::AKAZE **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::AKAZE	classcv_1_1AKAZE.html	Class implementing the AKAZE keypoint detector and descriptor extractor, described in [9]. AKAZE descriptors can only be used with KAZE or AKAZE keypoints. This class is thread-safe. NoteWhen you need descriptors use Feature2D::detectAndCompute, which provides better performance. When using Feature2D::detect followed by Feature2D::compute scale space pyramid is computed twice. AKAZE implements T-API. When image is passed as UMat some parts of the algorithm will use OpenCL. [ANB13] Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. Pablo F. Alcantarilla, Jesús Nuevo and Adrien Bartoli. In British Machine Vision Conference (BMVC), Bristol, UK, September 2013.								•	
features2d_AKAZE_setDescriptorType	CVAPI(ExceptionStatus) features2d_AKAZE_setDescriptorType(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setDescriptorType(static_cast<cv::AKAZE::DescriptorType>(val)); END_WRAP }	cv::AKAZE::setDescriptorType()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_getDescriptorType	CVAPI(ExceptionStatus) features2d_AKAZE_getDescriptorType(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getDescriptorType()); END_WRAP }	cv::AKAZE::getDescriptorType()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_setDescriptorSize	CVAPI(ExceptionStatus) features2d_AKAZE_setDescriptorSize(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setDescriptorSize(val); END_WRAP }	cv::AKAZE::setDescriptorSize()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_getDescriptorSize	CVAPI(ExceptionStatus) features2d_AKAZE_getDescriptorSize(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getDescriptorSize(); END_WRAP }	cv::AKAZE::getDescriptorSize()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_setDescriptorChannels	CVAPI(ExceptionStatus) features2d_AKAZE_setDescriptorChannels(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setDescriptorChannels(val); END_WRAP }	cv::AKAZE::setDescriptorChannels()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_getDescriptorChannels	CVAPI(ExceptionStatus) features2d_AKAZE_getDescriptorChannels(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getDescriptorChannels(); END_WRAP }	cv::AKAZE::getDescriptorChannels()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_setThreshold	CVAPI(ExceptionStatus) features2d_AKAZE_setThreshold(cv::AKAZE *obj, double val) { BEGIN_WRAP obj->setThreshold(val); END_WRAP }	cv::AKAZE::setThreshold()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_getThreshold	CVAPI(ExceptionStatus) features2d_AKAZE_getThreshold(cv::AKAZE *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::AKAZE::getThreshold()	classcv_1_1AKAZE.html				1										
features2d_AKAZE_setNOctaves	CVAPI(ExceptionStatus) features2d_AKAZE_setNOctaves(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setNOctaves(val); END_WRAP }	cv::AKAZE::setNOctaves()	classcv_1_1AKAZE.html				1										

features2d_AKAZE_getNOctaves	CVAPI(ExceptionStatus) features2d_AKAZE_getNOctaves(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaves(); END_WRAP }	cv::AKAZE::getNOctaves()	classcv_1_1AKAZE.html				1								
features2d_AKAZE_setNOctaveLayers	CVAPI(ExceptionStatus) features2d_AKAZE_setNOctaveLayers(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setNOctaveLayers(val); END_WRAP }	cv::AKAZE::setNOctaveLayers()	classcv_1_1AKAZE.html				1								
features2d_AKAZE_getNOctaveLayers	CVAPI(ExceptionStatus) features2d_AKAZE_getNOctaveLayers(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getNOctaveLayers(); END_WRAP }	cv::AKAZE::getNOctaveLayers()	classcv_1_1AKAZE.html				1								
features2d_AKAZE_setDiffusivity	CVAPI(ExceptionStatus) features2d_AKAZE_setDiffusivity(cv::AKAZE *obj, int val) { BEGIN_WRAP obj->setDiffusivity((static_cast<cv::AKAZE::DiffusivityType>(val)); END_WRAP }	cv::AKAZE::setDiffusivity()	classcv_1_1AKAZE.html				1								
features2d_AKAZE_getDiffusivity	CVAPI(ExceptionStatus) features2d_AKAZE_getDiffusivity(cv::AKAZE *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->getDiffusivity()); END_WRAP }	cv::AKAZE::getDiffusivity()	classcv_1_1AKAZE.html				1								
imgproc_getGaussianKernel	CVAPI(ExceptionStatus) imgproc_getGaussianKernel(int ksize, double sigma, int ktype, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::getGaussianKernel(ksize, sigma, ktype); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getGaussianKernel()	group__imgproc__filter.html	Returns Gaussian filter coefficients.	ksize : Aperture size. It should be odd (¥(¥texttt{ksize} ¥mod 2 = 1¥)) and positive. sigma : Gaussian standard deviation. If it is non-positive, it is computed from ksize as sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8. ktype : Type of filter coefficients. It can be CV_32F or CV_64F .	The function computes and returns the ¥(¥texttt{ksize} ¥times 1¥) matrix of Gaussian filter coefficients:¥[G_i= ¥alpha *e^{-(i¥(¥texttt{ksize} -1)/2)^2/2* ¥texttt{sigma}^2}),¥]where ¥(i=0.. ¥texttt{ksize}-1¥) and ¥(¥alpha¥) is the scale factor chosen so that ¥(¥sum_j G_j=1¥).Two of such generated kernels can be passed to sepFilter2D. Those functions automatically recognize smoothing kernels (a symmetrical kernel with sum of weights equal to 1) and handle them accordingly. You may also use the higher-level GaussianBlur.See alsosepFilter2D, getDerivKernels, getStructuringElement, GaussianBlur	3								
imgproc_getDerivKernels	CVAPI(ExceptionStatus) imgproc_getDerivKernels(cv::_OutputArray *kx, cv::_OutputArray *ky, int dx, int dy, int ksize, int normalize, int ktype) { BEGIN_WRAP cv::getDerivKernels(*kx, *ky, dx, dy, ksize, normalize != 0, ktype); END_WRAP }	cv::getDerivKernels()	group__imgproc__filter.html	Returns filter coefficients for computing spatial image derivatives.	kx : Output matrix of row filter coefficients. It has the type ktype . ky : Output matrix of column filter coefficients. It has the type ktype . dx : Derivative order in respect of x. dy : Derivative order in respect of y. ksize : Aperture size. It can be FILTER_SCHARR, 1, 3, 5, or 7. normalize : Flag indicating whether to normalize (scale down) the filter coefficients or not. Theoretically, the coefficients should have the denominator ¥(=2^{ksize*2-dx-dy-2}¥). If you are going to filter floating-point images, you are likely to use the normalized kernels. But if you compute derivatives of an 8-bit image, store the results in a 16-bit image, and wish to preserve all the fractional bits, you may want to set normalize=false . ktype : Type of filter coefficients. It can be CV_32f or CV_64F .	The function computes and returns the filter coefficients for spatial image derivatives. When ksize=FILTER_SCHARR, the Scharr ¥(3 ¥times 3¥) kernels are generated (see Scharr). Otherwise, Sobel kernels are generated (see Sobel). The filters are normally passed to sepFilter2D or to	7								
imgproc_getGaborKernel	CVAPI(ExceptionStatus) imgproc_getGaborKernel(MyCvSize ksize, double sigma, double theta, double lambd, double gamma, double psi, int ktype, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::getGaborKernel(cpp(ksize), sigma, theta, lambd, gamma, psi, ktype); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getGaborKernel()	group__imgproc__filter.html	Returns Gabor filter coefficients.	ksize : Size of the filter returned. sigma : Standard deviation of the gaussian envelope. theta : Orientation of the normal to the parallel stripes of a Gabor function. lambd : Wavelength of the sinusoidal factor. gamma : Spatial aspect ratio. psi : Phase offset. ktype : Type of filter coefficients. It can be CV_32F or CV_64F .	For more details about gabor filter equations and parameters, see: Gabor Filter.	7								
imgproc_getStructuringElement	CVAPI(ExceptionStatus) imgproc_getStructuringElement(int shape, MyCvSize ksize, MyCvPoint anchor, cv::Mat **returnValue) { BEGIN_WRAP const auto ret = cv::getStructuringElement(shape, cpp(ksize), cpp(anchor)); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getStructuringElement()	group__imgproc__filter.html	Returns a structuring element of the specified size and shape for morphological operations.	shape : Element shape that could be one of MorphShapes ksize : Size of the structuring element. anchor : Anchor position within the element. The default value ¥(-1, -1¥) means that the anchor is at the center. Note that only the shape of a cross-shaped element depends on the anchor position. In other cases the anchor just regulates how much the result of the morphological operation is shifted.	The function constructs and returns the structuring element that can be further passed to erode, dilate or morphologyEx. But you can also construct an arbitrary binary mask yourself and use it as the structuring element.Examples: samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, and samples/cpp/tutorial_code/ImgProc/Morphology_2.cpp.	3								
imgproc_medianBlur	CVAPI(ExceptionStatus) imgproc_medianBlur(cv::_InputArray *src, cv::_OutputArray *dst, int ksize) { BEGIN_WRAP cv::medianBlur(*src, *dst, ksize); END_WRAP }	cv::medianBlur()	group__imgproc__filter.html	Blurs an image using the median filter.	src : input 1-, 3-, or 4-channel image; when ksize is 3 or 5, the image depth should be CV_8U, CV_16U, or CV_32F, for larger aperture sizes, it can only be CV_8U. dst : destination array of the same size and type as src. ksize : aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...	The function smoothes an image using the median filter with the ¥(¥texttt{ksize} ¥times ¥texttt{ksize}¥) aperture. Each channel of a multi-channel image is processed independently. In-place operation is supported.NoteThe median filter uses BORDER_REPLICATE internally to cope with border pixels, see BorderTypesSee alsobilateralFilter, blur, boxFilter, GaussianBlurExamples: samples/cpp/laplace.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, and samples/cpp/tutorial_code/ImgTrans/houghcircles.cpp.	3								
imgproc_GaussianBlur	CVAPI(ExceptionStatus) imgproc_GaussianBlur(cv::_InputArray *src, cv::_OutputArray *dst, MyCvSize ksize, double sigmaX, double sigmaY, int borderType) { BEGIN_WRAP cv::GaussianBlur(*src, *dst, cpp(ksize), sigmaX, sigmaY, borderType); END_WRAP }	cv::GaussianBlur()	group__imgproc__filter.html	Blurs an image using a Gaussian filter.	src : input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. dst : output image of the same size and type as src. ksize : Gaussian kernel size. ksize.width and ksize.height can differ but they both must be positive and odd. Or, they can be zero's and then they are computed from sigma. sigmaX : Gaussian kernel standard deviation in X direction. sigmaY : Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed from ksize.width and ksize.height, respectively (see getGaussianKernel for details); to fully control the result regardless of possible future modifications of all this semantics, it is recommended to specify all of ksize, sigmaX, and sigmaY. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	The function convolves the source image with the specified Gaussian kernel. In-place filtering is supported.See alsosepFilter2D, filter2D, blur, boxFilter, bilateralFilter, medianBlurExamples: samples/cpp/laplace.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, and samples/cpp/tutorial_code/ImgTrans/Sobel_Demo.cpp.	6								
imgproc_bilateralFilter	CVAPI(ExceptionStatus) imgproc_bilateralFilter(cv::_InputArray *src, cv::_OutputArray *dst, int d, double sigmaColor, double sigmaSpace, int borderType) { BEGIN_WRAP cv::bilateralFilter(*src, *dst, d, sigmaColor, sigmaSpace, borderType); END_WRAP }	cv::bilateralFilter()	group__imgproc__filter.html	Applies the bilateral filter to an image.	src : Source 8-bit or floating-point, 1-channel or 3-channel image. dst : Destination image of the same size and type as src . d : Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace. sigmaColor : Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color. sigmaSpace : Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace. borderType : border mode used to extrapolate pixels outside of the image, see BorderTypes	The function applies bilateral filtering to the input image, as described in http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html bilateralFilter can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters.Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small (< 10), the filter will not have much effect, whereas if they are large (> 150), they will have a very strong effect, making the image look "cartoonish".Filter size: Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.This filter does not work inplace.Examples: samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp	6								
imgproc_boxFilter	CVAPI(ExceptionStatus) imgproc_boxFilter(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, MyCvSize ksize, MyCvPoint anchor, int normalize, int borderType) { BEGIN_WRAP cv::boxFilter(*src, *dst, ddepth, cpp(ksize), cpp(anchor), normalize != 0, borderType); END_WRAP }	cv::boxFilter()	group__imgproc__filter.html	Blurs an image using the box filter.	src : input image. dst : output image of the same size and type as src. ddepth : the output image depth (-1 to use src.depth()). ksize : blurring kernel size. anchor : anchor point; default value Point(-1,-1) means that the anchor is at the kernel center. normalize : flag, specifying whether the kernel is normalized by its area or not. borderType : border mode used to extrapolate pixels outside of the image, see BorderTypes. BORDER_WRAP is not supported.	The function smooths an image using the kernel:¥[¥texttt{K} = ¥alpha ¥begin{bmatrix} 1 & 1 & 1 & ¥cdots & 1 & 1 ¥¥ 1 & 1 & 1 & ¥cdots & 1 & 1 ¥¥ ¥hdotsfor{6} ¥¥ 1 & 1 & 1 & ¥cdots & 1 & 1 ¥end{bmatrix}¥]where¥[¥alpha = ¥begin{cases} ¥frac{1}{¥texttt{ksize.width*ksize.height}} & ¥texttt{when } ¥texttt{normalize=true} ¥¥ 1 & ¥texttt{otherwise}¥end{cases}¥]Unnormalized box filter is useful for computing various integral characteristics over each pixel neighborhood, such as covariance matrices of image derivatives (used in dense optical flow algorithms, and so on). If you need to compute pixel sums over variable-size windows, use integral.See alsoblur, bilateralFilter, GaussianBlur, medianBlur, integral	7								

imgproc_sqrBoxFilter	CVAPI(ExceptionStatus) imgproc_sqrBoxFilter(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, MyCvSize ksize, MyCvPoint anchor, int normalize, int borderType) { BEGIN_WRAP cv::sqrBoxFilter(*src, *dst, ddepth, cpp(ksize), cpp(anchor), normalize != 0, borderType); END_WRAP }	cv::sqrBoxFilter()	group__imgproc__filter.html	Calculates the normalized sum of squares of the pixel values overlapping the filter.	src : input image dst : output image of the same size and type as src ddepth : the output image depth (-1 to use src.depth()) ksize : kernel size anchor : kernel anchor point. The default value of Point(-1, -1) denotes that the anchor is at the kernel center. normalize : flag, specifying whether the kernel is to be normalized by it's area or not. borderType : border mode used to extrapolate pixels outside of the image, see BorderTypes. BORDER_WRAP is not supported.	For every pixel ¥((x, y) ¥) in the source image, the function calculates the sum of squares of those neighboring pixel values which overlap the filter placed over the pixel ¥((x, y) ¥).The unnormalized square box filter can be useful in computing local image statistics such as the the local variance and standard deviation around the neighborhood of a pixel.See alsoboxFilter	7								
imgproc_blur	CVAPI(ExceptionStatus) imgproc_blur(cv::_InputArray *src, cv::_OutputArray *dst, CvSize ksize, CvPoint anchor, int borderType) { BEGIN_WRAP cv::blur(*src, *dst, ksize, anchor, borderType); END_WRAP }	cv::blur()	group__imgproc__filter.html	Blurs an image using the normalized box filter.	src : input image; it can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. dst : output image of the same size and type as src. ksize : blurring kernel size. anchor : anchor point; default value Point(-1,-1) means that the anchor is at the kernel center. borderType : border mode used to extrapolate pixels outside of the image, see BorderTypes. BORDER_WRAP is not supported.	The function smooths an image using the kernel:¥[¥texttt{K} = ¥frac{1}{¥texttt{ksize.width*ksize.height }} ¥begin{bmatrix} 1 & 1 & 1 & ¥cdots & 1 & 1 ¥¥ 1 & 1 & 1 & ¥cdots & 1 & 1 ¥¥ ¥hdotsfor{6} ¥¥ 1 & 1 & 1 & ¥cdots & 1 & 1 ¥¥ ¥end{bmatrix}¥]The call blur(src, dst, ksize, anchor, borderType) is equivalent to boxFilter(src, dst, src.type(), ksize, anchor, true, borderType).See alsoboxFilter, bilateralFilter, GaussianBlur, medianBlurExamples: samples/cpp/edge.cpp, samples/cpp/laplace.cpp, and samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp.	5								
imgproc_filter2D	CVAPI(ExceptionStatus) imgproc_filter2D(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, cv::_InputArray *kernel, MyCvPoint anchor, double delta, int borderType) { BEGIN_WRAP cv::filter2D(*src, *dst, ddepth, *kernel, cpp(anchor), delta, borderType); END_WRAP }	cv::filter2D()	group__imgproc__filter.html	Convolves an image with the kernel.	src : input image. dst : output image of the same size and the same number of channels as src. ddepth : desired depth of the destination image, see combinations kernel : convolution kernel (or rather a correlation kernel), a single-channel floating point matrix; if you want to apply different kernels to different channels, split the image into separate color planes using split and process them individually. anchor : anchor of the kernel that indicates the relative position of a filtered point within the kernel; the anchor should lie within the kernel; default value (-1,-1) means that the anchor is at the kernel center. delta : optional value added to the filtered pixels before storing them in dst. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	The function applies an arbitrary linear filter to an image. In-place operation is supported. When the aperture is partially outside the image, the function interpolates outlier pixel values according to the specified border mode.The function does actually compute correlation, not the convolution:¥[¥texttt{dst} (x,y) = ¥sum _{ ¥substack{0¥leq x' < ¥texttt{kernel.cols}¥¥{0¥leq y' < ¥texttt{kernel.rows}}}} ¥texttt{kernel} (x',y')* ¥texttt{src} (x+x'- ¥texttt{anchor.x} ,y+y'- ¥texttt{anchor.y})¥]That is, the kernel is not mirrored around the anchor point. If you need a real convolution, flip the kernel using flip and set the new anchor to (kernel.cols - anchor.x - 1, kernel.rows - anchor.y - 1).The function uses the DFT-based algorithm in case of sufficiently large kernels (~11 x 11 or larger) and the direct algorithm for small kernels.See alsosepFilter2D, dft, matchTemplate	7								
imgproc_sepFilter2D	CVAPI(ExceptionStatus) imgproc_sepFilter2D(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, cv::_InputArray *kernelX, cv::_InputArray *kernelY, MyCvPoint anchor, double delta, int borderType) { BEGIN_WRAP cv::sepFilter2D(*src, *dst, ddepth, *kernelX, *kernelY, cpp(anchor), delta, borderType); END_WRAP }	cv::sepFilter2D()	group__imgproc__filter.html	Applies a separable linear filter to an image.	src : Source image. dst : Destination image of the same size and the same number of channels as src . ddepth : Destination image depth, see combinations kernelX : Coefficients for filtering each row. kernelY : Coefficients for filtering each column. anchor : Anchor position within the kernel. The default value ¥((-1,-1)¥) means that the anchor is at the kernel center. delta : Value added to the filtered results before storing them. borderType : Pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	The function applies a separable linear filter to the image. That is, first, every row of src is filtered with the 1D kernel kernelX. Then, every column of the result is filtered with the 1D kernel kernelY. The final result shifted by delta is stored in dst .See alsofilter2D, Sobel, GaussianBlur, boxFilter, blur	8								
imgproc_Sobel	CVAPI(ExceptionStatus) imgproc_Sobel(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, int dx, int dy, int ksize, double scale, double delta, int borderType) { BEGIN_WRAP cv::Sobel(*src, *dst, ddepth, dx, dy, ksize, scale, delta, borderType); END_WRAP }	cv::Sobel()	group__imgproc__filter.html	Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator.	src : input image. dst : output image of the same size and the same number of channels as src . ddepth : output image depth, see combinations; in the case of 8-bit input images it will result in truncated derivatives. dx : order of the derivative x. dy : order of the derivative y. ksize : size of the extended Sobel kernel; it must be 1, 3, 5, or 7. scale : optional scale factor for the computed derivative values; by default, no scaling is applied (see getDerivKernels for details). delta : optional delta value that is added to the results prior to storing them in dst. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	In all cases except one, the ¥(¥texttt{ksize} ¥times ¥texttt{ksize}¥) separable kernel is used to calculate the derivative. When ¥(¥texttt{ksize = 1}¥), the ¥(3 ¥times 1¥) or ¥(1 ¥times 3¥) kernel is used (that is, no Gaussian smoothing is done). ksize = 1 can only be used for the first or the second x- or y- derivatives.There is also the special value ksize = FILTER_SCHARR (-1) that corresponds to the ¥(3¥times3¥) Scharr filter that may give more accurate results than the ¥(3¥times3¥) Sobel. The Scharr aperture is¥[¥vecthreethree{-3}{0}{3}{-10}{0}{10}{-3}{0}{3}¥]for the x-derivative, or transposed for the y-derivative.The function calculates an image derivative by convolving the image with the appropriate kernel:¥[¥texttt{dst} = ¥frac{¥partial^{xorder+yorder} ¥texttt{src}}{¥partial x^{xorder} ¥partial y^{yorder}}¥]The Sobel operators combine Gaussian smoothing and differentiation, so the result is more or less resistant to the noise. Most often,	9								
imgproc_spatialGradient	CVAPI(ExceptionStatus) imgproc_spatialGradient(cv::_InputArray *src, cv::_OutputArray *dx, cv::_OutputArray *dy, int ksize, int borderType) { BEGIN_WRAP cv::spatialGradient(*src, *dx, *dy, ksize, borderType); END_WRAP }	cv::spatialGradient()	group__imgproc__filter.html	Calculates the first order image derivative in both x and y using a Sobel operator.	src : input image. dx : output image with first-order derivative in x. dy : output image with first-order derivative in y. ksize : size of Sobel kernel. It must be 3. borderType : pixel extrapolation method, see BorderTypes. Only BORDER_DEFAULT=BORDER_REFLECT_101 and BORDER_REPLICATE are supported.	Equivalent to calling:Sobel(src, dx, CV_16SC1, 1, 0, 3);Sobel(src, dy, CV_16SC1, 0, 1, 3);fragmentSee alsoSobel	5								
imgproc_Scharr	CVAPI(ExceptionStatus) imgproc_Scharr(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, int dx, int dy, double scale, double delta, int borderType) { BEGIN_WRAP cv::Scharr(*src, *dst, ddepth, dx, dy, scale, delta, borderType); END_WRAP }	cv::Scharr()	group__imgproc__filter.html	Calculates the first x- or y- image derivative using Scharr operator.	src : input image. dst : output image of the same size and the same number of channels as src. ddepth : output image depth, see combinations dx : order of the derivative x. dy : order of the derivative y. scale : optional scale factor for the computed derivative values; by default, no scaling is applied (see getDerivKernels for details). delta : optional delta value that is added to the results prior to storing them in dst. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	The function computes the first x- or y- spatial image derivative using the Scharr operator. The call¥[¥texttt{Scharr(src, dst, ddepth, dx, dy, scale, delta, borderType)}¥]is equivalent to¥[¥texttt{Sobel(src, dst, ddepth, dx, dy, FILTER_SCHARR, scale, delta, borderType)} .¥]See alsocartToPolarExamples: samples/cpp/edge.cpp.	8								
imgproc_Laplacian	CVAPI(ExceptionStatus) imgproc_Laplacian(cv::_InputArray *src, cv::_OutputArray *dst, int ddepth, int ksize, double scale, double delta, int borderType) { BEGIN_WRAP cv::Laplacian(*src, *dst, ddepth, ksize, scale, delta, borderType); END_WRAP }	cv::Laplacian()	group__imgproc__filter.html	Calculates the Laplacian of an image.	src : Source image. dst : Destination image of the same size and the same number of channels as src . ddepth : Desired depth of the destination image. ksize : Aperture size used to compute the second-derivative filters. See getDerivKernels for details. The size must be positive and odd. scale : Optional scale factor for the computed Laplacian values. By default, no scaling is applied. See getDerivKernels for details. delta : Optional delta value that is added to the results prior to storing them in dst . borderType : Pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported.	The function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator:¥[¥texttt{dst} = ¥Delta ¥texttt{src} = ¥frac{¥partial^2 ¥texttt{src}}{¥partial x^2} + ¥frac{¥partial^2 ¥texttt{src}}{¥partial y^2}¥]This is done when ksize > 1. When ksize == 1, the Laplacian is computed by filtering the image with the following ¥(3 ¥times 3¥) aperture:¥[¥vecthreethree {0}{1}{0}{1}{-4}{1}{0}{1}{0}¥]See alsoSobel, ScharrExamples: samples/cpp/laplace.cpp.	7								
imgproc_Canny1	CVAPI(ExceptionStatus) imgproc_Canny1(cv::_InputArray *src, cv::_OutputArray *edges, double threshold1, double threshold2, int apertureSize, int L2gradient) { BEGIN_WRAP cv::Canny(*src, *edges, threshold1, threshold2, apertureSize, L2gradient != 0); END_WRAP }	cv::Canny()	group__imgproc__feature.html	Finds edges in an image using the Canny algorithm [41] .	image : 8-bit input image. edges : output edge map; single channels 8-bit image, which has the same size as image . threshold1 : first threshold for the hysteresis procedure. threshold2 : second threshold for the hysteresis procedure. apertureSize : aperture size for the Sobel operator. L2gradient : a flag, indicating whether a more accurate ¥(L_2¥) norm ¥(=¥sqrt{(dI/dx)^2 + (dI/dy)^2}¥) should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default ¥(L_1¥) norm ¥(=|dI/dx|+|dI/dy|¥) is enough (L2gradient=false).	The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detectorExamples: samples/cpp/edge.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ImgTrans/houghlines.cpp, and samples/tapi/squares.cpp.	6								
imgproc_Canny2	CVAPI(ExceptionStatus) imgproc_Canny2(cv::_InputArray *dx, cv::_InputArray *dy, cv::_OutputArray *edges, double threshold1, double threshold2, int L2gradient = false) { BEGIN_WRAP cv::Canny(*dx, *dy, *edges, threshold1, threshold2, L2gradient != 0); END_WRAP }	cv::Canny()	group__imgproc__feature.html	Finds edges in an image using the Canny algorithm [41] .	image : 8-bit input image. edges : output edge map; single channels 8-bit image, which has the same size as image . threshold1 : first threshold for the hysteresis procedure. threshold2 : second threshold for the hysteresis procedure. apertureSize : aperture size for the Sobel operator. L2gradient : a flag, indicating whether a more accurate ¥(L_2¥) norm ¥(=¥sqrt{(dI/dx)^2 + (dI/dy)^2}¥) should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default ¥(L_1¥) norm ¥(=|dI/dx|+|dI/dy|¥) is enough (L2gradient=false).	The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detectorExamples: samples/cpp/edge.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ImgTrans/houghlines.cpp, and samples/tapi/squares.cpp.	6								
imgproc_cornerMinEigenVal	CVAPI(ExceptionStatus) imgproc_cornerMinEigenVal(cv::_InputArray *src, cv::_OutputArray *dst, int blockSize, int ksize, int borderType) { BEGIN_WRAP cv::cornerMinEigenVal(*src, *dst, blockSize, ksize, borderType); END_WRAP }	cv::cornerMinEigenVal()	group__imgproc__feature.html	Calculates the minimal eigenvalue of gradient matrices for corner detection.	src : Input single-channel 8-bit or floating-point image. dst : Image to store the minimal eigenvalues. It has the type CV_32FC1 and the same size as src . blockSize : Neighborhood size (see the details on cornerEigenValsAndVecs). ksize : Aperture parameter for the Sobel operator. borderType : Pixel extrapolation method. See BorderTypes. BORDER_WRAP is not supported.	The function is similar to cornerEigenValsAndVecs but it calculates and stores only the minimal eigenvalue of the covariance matrix of derivatives, that is, ¥(¥min(¥lambda_1, ¥lambda_2)¥) in terms of the formulae in the cornerEigenValsAndVecs description.	5								
imgproc_cornerHarris	CVAPI(ExceptionStatus) imgproc_cornerHarris(cv::_InputArray *src, cv::_OutputArray *dst, int blockSize, int ksize, double k, int borderType) { BEGIN_WRAP cv::cornerHarris(*src, *dst, blockSize, ksize, k, borderType); END_WRAP }	cv::cornerHarris()	group__imgproc__feature.html	Harris corner detector.	src : Input single-channel 8-bit or floating-point image. dst : Image to store the Harris detector responses. It has the type CV_32FC1 and the same size as src . blockSize : Neighborhood size (see the details on cornerEigenValsAndVecs). ksize : Aperture parameter for the Sobel operator. k : Harris detector free parameter. See the formula above. borderType : Pixel extrapolation method. See BorderTypes. BORDER_WRAP is not supported.	The function runs the Harris corner detector on the image. Similarly to cornerMinEigenVal and cornerEigenValsAndVecs , for each pixel ¥((x, y)¥) it calculates a ¥(2¥times2¥) gradient covariance matrix ¥(M^{(x,y)}¥) over a ¥(¥texttt{blockSize} ¥times ¥texttt{blockSize}¥) neighborhood. Then, it computes the following characteristic:¥[¥texttt{dst} (x,y) = ¥mathrm{det} M^{(x,y)} - k ¥cdot ¥left (¥mathrm{tr} M^{(x,y)} ¥right)^2¥]Corners in the image can be found as the local maxima of this response map.	6								

imgproc_cornerEigenValsAndVecs	CVAPI(ExceptionStatus) imgproc_cornerEigenValsAndVecs(cv::_InputArray *src, cv::_OutputArray *dst, int blockSize, int ksize, int borderType) { BEGIN_WRAP cv::cornerEigenValsAndVecs(*src, *dst, blockSize, ksize, borderType); END_WRAP }	cv::cornerEigenValsAndVecs()	group__imgproc__feature.html	Calculates eigenvalues and eigenvectors of image blocks for corner detection.	src : Input single-channel 8-bit or floating-point image. dst : Image to store the results. It has the same size as src and the type CV_32FC(6) . blockSize : Neighborhood size (see details below). ksize : Aperture parameter for the Sobel operator. borderType : Pixel extrapolation method. See BorderTypes. BORDER_WRAP is not supported.	For every pixel ¥(p¥) , the function cornerEigenValsAndVecs considers a blockSize ¥(¥times¥) blockSize neighborhood ¥(S(p)¥) . It calculates the covariation matrix of derivatives over the neighborhood as:¥[M = ¥begin{bmatrix} ¥sum _{S(p)}(dI/dx)^2 & ¥sum _{S(p)}dI/dx dI/dy ¥¥ ¥sum _{S(p)}dI/dx dI/dy & ¥sum _{S(p)}(dI/dy)^2 ¥end{bmatrix}¥]where the derivatives are computed using the Sobel operator.After that, it finds eigenvectors and eigenvalues of ¥(M¥) and stores them in the destination image as ¥(¥lambda_1, ¥lambda_2, x_1, y_1, x_2, y_2¥) where¥(¥lambda_1, ¥lambda_2¥) are the non-sorted eigenvalues of ¥(M¥)¥(x_1, y_1¥) are the eigenvectors corresponding to ¥(¥lambda_1¥)¥(x_2, y_2¥) are the eigenvectors corresponding to ¥(¥lambda_2¥)The output of the function can be used for robust edge or corner detection.See alsocornerMinEigenVal, cornerHarris, preCornerDetect	5								
imgproc_preCornerDetect	CVAPI(ExceptionStatus) imgproc_preCornerDetect(cv::_InputArray *src, cv::_OutputArray *dst, int ksize, int borderType) { BEGIN_WRAP cv::preCornerDetect(*src, *dst, ksize, borderType); END_WRAP }	cv::preCornerDetect()	group__imgproc__feature.html	Calculates a feature map for corner detection.	src : Source single-channel 8-bit of floating-point image. dst : Output image that has the type CV_32F and the same size as src . ksize : Aperture size of the Sobel . borderType : Pixel extrapolation method. See BorderTypes. BORDER_WRAP is not supported.	The function calculates the complex spatial derivative-based function of the source image¥[¥texttt{dst} = (D_x ¥texttt{src})^2 ¥cdot D_{yy} ¥texttt{src} + (D_y ¥texttt{src})^2 ¥cdot D_{xx} ¥texttt{src} - 2 D_x ¥texttt{src} ¥cdot D_y ¥texttt{src} ¥cdot D_{xy} ¥texttt{src}¥]where ¥(D_x¥), ¥(D_y¥) are the first image derivatives, ¥(D_{xx}¥), ¥(D_{yy}¥) are the second image derivatives, and ¥(D_{xy}¥) is the mixed derivative.The corners can be found as local maximums of the functions, as shown below:Mat corners, dilated_corners;preCornerDetect(image, corners, 3);// dilation with 3x3 rectangular structuring elementdilate(corners, dilated_corners, Mat(), 1);Mat corner_mask = corners ==	4								
imgproc_cornerSubPix	CVAPI(ExceptionStatus) imgproc_cornerSubPix(cv::_InputArray *image, std::vector<cv::Point2f> *corners, CvSize winSize, CvSize zeroZone, MyCvTermCriteria criteria) { BEGIN_WRAP cv::cornerSubPix(*image, *corners, winSize, zeroZone, cpp(criteria)); END_WRAP }	cv::cornerSubPix()	group__imgproc__feature.html	Refines the corner locations.	image : Input single-channel, 8-bit or float image. corners : Initial coordinates of the input corners and refined coordinates provided for output. winSize : Half of the side length of the search window. For example, if winSize=Size(5,5) , then a ¥((5*2+1) ¥times (5*2+1) = 11 ¥times 11¥) search window is used. zeroZone : Half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done. It is used sometimes to avoid possible singularities of the autocorrelation matrix. The value of (-1,-1) indicates that there is no such a size. criteria : Criteria for termination of the iterative process of corner refinement. That is, the process of corner position refinement stops either after criteria.maxCount iterations or when the corner position moves by less than criteria.epsilon on some iteration.	The function iterates to find the sub-pixel accurate location of corners or radial saddle points as described in [81], and as shown on the figure below.imageSub-pixel accurate corner locator is based on the observation that every vector from the center ¥(q¥) to a point ¥(p¥) located within a neighborhood of ¥(q¥) is orthogonal to the image gradient at ¥(p¥) subject to image and measurement noise. Consider the expression:¥[¥epsilon _i = {DI_{p_i}}^T ¥cdot (q - p_i)¥]where ¥(DI_{p_i}¥) is an image gradient at one of the points ¥(p_i¥) in a neighborhood of ¥(q¥) . The value of ¥(q¥) is to be found so that ¥(¥epsilon_i¥) sat to zero:¥[¥sum _i(DI_{p_i} ¥cdot {DI_{p_i}}^T) ¥cdot q - ¥sum _i(DI_{p_i} ¥cdot {DI_{p_i}}^T ¥cdot p_i)¥]where the gradients are summed within a neighborhood ("search window") of ¥(q¥) . Calling the first gradient term ¥(G¥) and the second gradient term ¥(b¥) gives:¥[q = G^{-1} ¥cdot b¥]The algorithm sets the center of	5								
imgproc_goodFeaturesToTrack	CVAPI(ExceptionStatus) imgproc_goodFeaturesToTrack(cv::_InputArray *src, std::vector<cv::Point2f> *corners, int maxCorners, double qualityLevel, double minDistance, cv::_InputArray *mask, int blockSize, int useHarrisDetector, double k) { BEGIN_WRAP cv::goodFeaturesToTrack(*src, *corners, maxCorners, qualityLevel, minDistance, entity(mask), blockSize, useHarrisDetector != 0, k); END_WRAP }	cv::goodFeaturesToTrack()	group__imgproc__feature.html	Determines strong corners on an image.	image : Input 8-bit or floating-point 32-bit, single-channel image. corners : Output vector of detected corners. maxCorners : Maximum number of corners to return. If there are more corners than are found, the strongest of them is returned. maxCorners <= 0 implies that no limit on the maximum is set and all detected corners are returned. qualityLevel : Parameter characterizing the minimal accepted quality of image corners. The parameter value is multiplied by the best corner quality measure, which is the minimal eigenvalue (see cornerMinEigenVal) or the Harris function response (see cornerHarris). The corners with the quality measure less than the product are rejected. For example, if the best corner has the quality measure = 1500, and the qualityLevel=0.01 , then all the corners with the quality measure less than 15 are rejected. minDistance : Minimum possible Euclidean distance between the returned corners. mask : Optional region of interest. If the image is not empty (it needs to have the type CV_8UC1 and the same size as image), it specifies the region in which the corners are detected. blockSize : Size of an average block for computing a derivative covariation matrix over each pixel	The function finds the most prominent corners in the image or in the specified image region, as described in [222]Function calculates the corner quality measure at every source image pixel using the cornerMinEigenVal or cornerHarris . Function performs a non-maximum suppression (the local maximums in 3 x 3 neighborhood are retained). The corners with the minimal eigenvalue less than ¥(¥texttt{qualityLevel} ¥cdot ¥max_{x,y} qualityMeasureMap(x,y)¥) are rejected. The remaining corners are sorted by the quality measure in the descending order. Function throws away each corner for which there is a stronger corner at a distance less than maxDistance.The function can be used to initialize a point-based tracker of an object.Notelf the function is called with different values A and B of the parameter qualityLevel , and A > B, the vector of returned corners with qualityLevel=A will be the prefix of the output vector with qualityLevel=B .See	9								
imgproc_HoughLines	CVAPI(ExceptionStatus) imgproc_HoughLines(cv::_InputArray *src, std::vector<cv::Vec2f> *lines, double rho, double theta, int threshold, double srn, double stn) { BEGIN_WRAP cv::HoughLines(*src, *lines, rho, theta, threshold, srn, stn); END_WRAP }	cv::HoughLines()	group__imgproc__feature.html	Finds lines in a binary image using the standard Hough transform.	image : 8-bit, single-channel binary source image. The image may be modified by the function. lines : Output vector of lines. Each line is represented by a 2 or 3 element vector ¥((¥rho, ¥theta)¥) or ¥((¥rho, ¥theta, ¥vote¥m{votes})¥) . ¥(¥rho¥) is the distance from the coordinate origin ¥((0,0)¥) (top-left corner of the image). ¥(¥theta¥) is the line rotation angle in radians (¥(0 ¥sim ¥textrm{vertical line}¥), ¥(pi/2 ¥sim ¥textrm{horizontal line}¥)). ¥(¥textrm{votes}¥) is the value of accumulator. rho : Distance resolution of the accumulator in pixels. theta : Angle resolution of the accumulator in radians. threshold : Accumulator threshold parameter. Only those lines are returned that get enough votes (¥(>¥texttt{threshold}¥)). srn : For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive. stn : For the multi-scale Hough transform, it is a divisor for the distance resolution theta. min_theta : For standard and multi-scale Hough transform, minimum angle to check for lines. Must fall	The function implements the standard or standard multi-scale Hough transform algorithm for line detection. See http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm for a good explanation of Hough transform.Examples: samples/cpp/tutorial_code/ImgTrans/houghlines.cpp.	7								
imgproc_HoughLinesP	CVAPI(ExceptionStatus) imgproc_HoughLinesP(cv::_InputArray *src, std::vector<cv::Vec4i> *lines, double rho, double theta, int threshold, double minLineLength, double maxLineGap) { BEGIN_WRAP cv::HoughLinesP(*src, *lines, rho, theta, threshold, minLineLength, maxLineGap); END_WRAP }	cv::HoughLinesP()	group__imgproc__feature.html	Finds line segments in a binary image using the probabilistic Hough transform.	image : 8-bit, single-channel binary source image. The image may be modified by the function. lines : Output vector of lines. Each line is represented by a 4-element vector ¥((x_1, y_1, x_2, y_2)¥) , where ¥((x_1,y_1)¥) and ¥((x_2, y_2)¥) are the ending points of each detected line segment. rho : Distance resolution of the accumulator in pixels. theta : Angle resolution of the accumulator in radians. threshold : Accumulator threshold parameter. Only those lines are returned that get enough votes (¥(>¥texttt{threshold}¥)). minLineLength : Minimum line length. Line segments shorter than that are rejected. maxLineGap : Maximum allowed gap between points on the same line to link them.	The function implements the probabilistic Hough transform algorithm for line detection, described in [164]See the line detection example below:#include <opencv2/imgproc.hpp>#include <opencv2/highgui.hpp>using namespace cv;using namespace std;int main(int argc, char** argv){ Mat src, dst, color_dst; if(argc != 2 || !(src=imread(argv[1], 0)).data) return -1; Canny(src, dst, 50, 200, 3); cvtColor(dst, color_dst, COLOR_GRAY2BGR); vector<Vec4i> lines; HoughLinesP(dst, lines, 1, CV_PI/180, 80, 30, 10); for(size_t i = 0; i < lines.size(); i++) { line(color_dst, Point(lines[i][0], lines[i][1]), Point(lines[i][2], lines[i][3]), Scalar(0,0,255), 3, 8); } namedWindow("Source", 1); imshow("Source", src); namedWindow("Detected Lines", 1); imshow("Detected Lines", color_dst); waitKey(0); return 0;}fragmentThis is a sample picture the function parameters have been tuned for:imageAnd this is the output of the above program in case of the probabilistic Hough	7								
imgproc_HoughLinesPointSet	CVAPI(ExceptionStatus) imgproc_HoughLinesPointSet(cv::_InputArray *_point, cv::_OutputArray *_lines, int lines_max, int threshold, double min_rho, double max_rho, double rho_step, double min_theta, double max_theta, double theta_step) { BEGIN_WRAP cv::HoughLinesPointSet(*_point, *_lines, lines_max, threshold, min_rho, max_rho, rho_step, min_theta, max_theta, theta_step); END_WRAP }	cv::HoughLinesPointSet()	group__imgproc__feature.html	Finds lines in a set of points using the standard Hough transform.	point : Input vector of points. Each vector must be encoded as a Point vector ¥((x,y)¥). Type must be CV_32FC2 or CV_32SC2. lines : Output vector of found lines. Each vector is encoded as a vector<Vec3d> ¥((votes, rho, theta)¥). The larger the value of 'votes', the higher the reliability of the Hough line. lines_max : Max count of hough lines. threshold : Accumulator threshold parameter. Only those lines are returned that get enough votes (¥(>¥texttt{threshold}¥)) min_rho : Minimum Distance value of the accumulator in pixels. max_rho : Maximum Distance value of the accumulator in pixels. rho_step : Distance resolution of the accumulator in pixels. min_theta : Minimum angle value of the accumulator in radians. max_theta : Maximum angle value of the accumulator in radians. theta_step : Angle resolution of the accumulator in radians.	The function finds lines in a set of points using a modification of the Hough transform.#include <opencv2/core.hpp>#include <opencv2/imgproc.hpp>using namespace cv;using namespace std;int main(){ Mat lines; vector<Vec3d> line3d; vector<Point2f> point; const static float Points[20][2] = { { 0.0f, 369.0f }, { 10.0f, 364.0f }, { 20.0f, 358.0f }, { 30.0f, 352.0f }, { 40.0f, 346.0f }, { 50.0f, 341.0f }, { 60.0f, 335.0f }, { 70.0f, 329.0f }, { 80.0f, 323.0f }, { 90.0f, 318.0f }, { 100.0f, 312.0f }, { 110.0f, 306.0f }, { 120.0f, 300.0f }, { 130.0f, 295.0f }, { 140.0f, 289.0f }, { 150.0f, 284.0f }, { 160.0f, 277.0f }, { 170.0f, 271.0f }, { 180.0f, 266.0f }, { 190.0f, 260.0f } }; for (int i = 0; i < 20; i++) { point.push_back(Point2f(Points[i][0],Points[i][1])); } double rhoMin = 0.0f, rhoMax = 360.0f, rhoStep = 1; double thetaMin = 0.0f, thetaMax = CV_PI / 2.0f, thetaStep = CV_PI / 180.0f; HoughLinesPointSet(point, lines, 20, 1, rhoMin, rhoMax, rhoStep,	10								
imgproc_HoughCircles	CVAPI(ExceptionStatus) imgproc_HoughCircles(cv::_InputArray *src, std::vector<cv::Vec3f> *circles, int method, double dp, double minDist, double param1, double param2, int minRadius, int maxRadius) { BEGIN_WRAP cv::HoughCircles(*src, *circles, method, dp, minDist, param1, param2, minRadius, maxRadius); END_WRAP }	cv::HoughCircles()	group__imgproc__feature.html	Finds circles in a grayscale image using the Hough transform.	image : 8-bit, single-channel, grayscale input image. circles : Output vector of found circles. Each vector is encoded as 3 or 4 element floating-point vector ¥((x, y, radius)¥) or ¥((x, y, radius, votes)¥) . method : Detection method, see HoughModes. The available methods are HOUGH_GRADIENT and HOUGH_GRADIENT_ALT. dp : Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height. For HOUGH_GRADIENT_ALT the recommended value is dp=1.5, unless some small very circles need to be detected. minDist : Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed. param1 : First method-specific parameter. In case of HOUGH_GRADIENT and HOUGH_GRADIENT_ALT, it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller). Note that HOUGH_GRADIENT_ALT uses Scharr algorithm to compute image derivatives, so the threshold value ¥hough normally be higher, such as 300 or normally exposed and contrasty images	The function finds circles in a grayscale image using a modification of the Hough transform.Example: :#include <opencv2/imgproc.hpp>#include <opencv2/highgui.hpp>#include <math.h><using namespace cv;using namespace std;int main(int argc, char** argv){ Mat img, gray; if(argc != 2 || !(img=imread(argv[1], 1)).data) return -1; cvtColor(img, gray, COLOR_BGR2GRAY); // smooth it, otherwise a lot of false circles may be detected GaussianBlur(gray, gray, Size(9, 9), 2, 2); vector<Vec3f> circles; HoughCircles(gray, circles, HOUGH_GRADIENT, 2, gray.rows/4, 200, 100); for(size_t i = 0; i < circles.size(); i++) { Point center(cvRound(circles[i][0]), cvRound(circles[i][1])); int radius = cvRound(circles[i][2]); // draw the circle center circle(img, center, 3, Scalar(0,255,0), -1, 8, 0); // draw the circle outline circle(img, center, radius, Scalar(0,0,255), 3, 8, 0); } namedWindow("circles", 1); imshow("circles", img); waitKey(0); return	9								

imgproc_erode	CVAPI(ExceptionStatus) imgproc_erode(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *kernel, MyCvPoint anchor, int iterations, int borderType, MyCvScalar borderValue) { BEGIN_WRAP cv::erode(*src, *dst, entity(kernel), cpp(anchor), iterations, borderType, cpp(borderValue)); END_WRAP }	cv::erode()	group__imgproc__filter.html	Erodes an image by using a specific structuring element.	src : input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. dst : output image of the same size and type as src. kernel : structuring element used for erosion; if element=Mat(), a 3 x 3 rectangular structuring element is used. Kernel can be created using getStructuringElement. anchor : position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center. iterations : number of times erosion is applied. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported. borderValue : border value in case of a constant border	The function erodes the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum is taken:¥[¥texttt{dst} (x,y) = ¥min _{(x',y') ¥, ¥texttt{element} (x',y') ¥neq0 } ¥texttt{src} (x+x',y+y')¥]The function supports the in-place mode. Erosion can be applied several (iterations) times. In case of multi-channel images, each channel is processed independently.See alsodilate, morphologyEx, getStructuringElementExamples: samples/cpp/segment_objects.cpp, and samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp.	7								
imgproc_dilate	CVAPI(ExceptionStatus) imgproc_dilate(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *kernel, CvPoint anchor, int iterations, int borderType, CvScalar borderValue) { BEGIN_WRAP cv::dilate(*src, *dst, entity(kernel), anchor, iterations, borderType, borderValue); END_WRAP }	cv::dilate()	group__imgproc__filter.html	Dilates an image by using a specific structuring element.	src : input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. dst : output image of the same size and type as src. kernel : structuring element used for dilation; if element=Mat(), a 3 x 3 rectangular structuring element is used. Kernel can be created using getStructuringElement anchor : position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center iterations : number of times dilation is applied. borderType : pixel extrapolation method, see BorderTypes. BORDER_WRAP is not suported. borderValue : border value in case of a constant border	The function dilates the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum is taken:¥[¥texttt{dst} (x,y) = ¥max _{(x',y') ¥, ¥texttt{element} (x',y') ¥neq0 } ¥texttt{src} (x+x',y+y')¥]The function supports the in-place mode. Dilation can be applied several (iterations) times. In case of multi-channel images, each channel is processed independently.See alsoerode, morphologyEx, getStructuringElementExamples: samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/stitching_detailed.cpp, samples/cpp/tutorial_code/ImgProc/Morphology_1.cpp, and samples/tapi/squares.cpp.	7								
imgproc_morphologyEx	CVAPI(ExceptionStatus) imgproc_morphologyEx(cv::_InputArray *src, cv::_OutputArray *dst, int op, cv::_InputArray *kernel, MyCvPoint anchor, int iterations, int borderType, MyCvScalar borderValue) { BEGIN_WRAP cv::morphologyEx(*src, *dst, op, entity(kernel), cpp(anchor), iterations, borderType, cpp(borderValue)); END_WRAP }	cv::morphologyEx()	group__imgproc__filter.html	Performs advanced morphological transformations.	src : Source image. The number of channels can be arbitrary. The depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F. dst : Destination image of the same size and type as source image. op : Type of a morphological operation, see MorphTypes kernel : Structuring element. It can be created using getStructuringElement. anchor : Anchor position with the kernel. Negative values mean that the anchor is at the kernel center. iterations : Number of times erosion and dilation are applied. borderType : Pixel extrapolation method, see BorderTypes. BORDER_WRAP is not supported. borderValue : Border value in case of a constant border. The default value has a special meaning.	The function cv::morphologyEx can perform advanced morphological transformations using an erosion and dilation as basic operations.Any of the operations can be done in-place. In case of multi-channel images, each channel is processed independently.See alsodilate, erode, getStructuringElementNoteThe number of iterations is the number of times erosion or dilatation operation will be applied. For instance, an opening operation (MORPH_OPEN) with two iterations is equivalent to apply successively: erode -> erode -> dilate -> dilate (and not erode -> dilate -> erode -> dilate).Examples: samples/cpp/tutorial_code/ImgProc/Morphology_2.cpp.	8								
imgproc_resize	CVAPI(ExceptionStatus) imgproc_resize(cv::_InputArray* src, cv::_OutputArray* dst, MyCvSize dsize, double fx, double fy, int interpolation) { BEGIN_WRAP cv::resize(*src, *dst, cpp(dsize), fx, fy, interpolation); END_WRAP }	cv::resize()	group__imgproc__transform.html	Resizes an image.	src : input image. dst : output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is the same as of src. dsize : output image size; if it equals zero, it is computed as: ¥[¥texttt{dsize = Size(round(fx*src.cols), round(fy*src.rows))}¥] Either dsize or both fx and fy must be non-zero. fx : scale factor along the horizontal axis; when it equals 0, it is computed as ¥[¥texttt{(double)dsize.width/src.cols}¥] fy : scale factor along the vertical axis; when it equals 0, it is computed as ¥[¥texttt{(double)dsize.height/src.rows}¥] interpolation : interpolation method, see InterpolationFlags	The function resize resizes the image src down to or up to the specified size. Note that the initial dst type or size are not taken into account. Instead, the size and type are derived from the src,dsize,fx, and fy. If you want to resize src so that it fits the pre-created dst, you may call the function as follows:// explicitly specify dsize=dst.size(); fx and fy will be computed from that.resize(src, dst, dst.size(), 0, 0, interpolation);//fragmentIf you want to dicrmate the image by factor of 2 in each direction, you can call the function this way:// specify fx and fy and let the function compute the destination image size.resize(src, dst, Size(), 0.5, 0.5, interpolation);//fragmentTo shrink an image, it will generally look best with INTER_AREA interpolation, whereas to enlarge an image, it will generally look best with c::INTER_CUBIC (slow) or INTER_LINEAR (faster but still looks OK).See alsowarpAffine, warpPerspective, remapExamples: samples/cpp/image_alignment.cpp, samples/cpp/train_HOG.cpp, samples/dnn/classification.cpp	6								
imgproc_warpAffine	CVAPI(ExceptionStatus) imgproc_warpAffine(cv::_InputArray* src, cv::_OutputArray* dst, cv::_InputArray* M, MyCvSize dsize, int flags, int borderMode, MyCvScalar borderValue) { BEGIN_WRAP cv::warpAffine(*src, *dst, *M, cpp(dsize), flags, borderMode, cpp(borderValue)); END_WRAP }	cv::warpAffine()	group__imgproc__transform.html	Applies an affine transformation to an image.	src : input image. dst : output image that has the size dsize and the same type as src . M : ¥(2¥times 3¥) transformation matrix. dsize : size of the output image. flags : combination of interpolation methods (see InterpolationFlags) and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation (¥[¥texttt{dst}¥rightarrow¥texttt{src}¥]). borderMode : pixel extrapolation method (see BorderTypes); when borderMode=BORDER_TRANSPARENT, it means that the pixels in the destination image corresponding to the "outliers" in the source image are not modified by the function. borderValue : value used in case of a constant border; by default, it is 0.	The function warpAffine transforms the source image using the specified matrix:¥[¥texttt{dst} (x,y) = ¥texttt{src} (¥texttt{M} _{11} x + ¥texttt{M} _{12} y + ¥texttt{M} _{13}, ¥texttt{M} _{21} x + ¥texttt{M} _{22} y + ¥texttt{M} _{23})¥]when the flag WARP_INVERSE_MAP is set. Otherwise, the transformation is first inverted with invertAffineTransform and then put in the formula above instead of M. The function cannot operate in-place.See alsowarpPerspective, resize, remap, getRectSubPix, transformExamples: samples/cpp/image_alignment.cpp.	7								
imgproc_warpPerspective_MisInputArray	CVAPI(ExceptionStatus) imgproc_warpPerspective_MisInputArray(cv::_InputArray* src, cv::_OutputArray* dst, cv::_InputArray* m, MyCvSize dsize, int flags, int borderMode, MyCvScalar borderValue) { BEGIN_WRAP cv::warpPerspective(*src, *dst, *m, cpp(dsize), flags, borderMode, cpp(borderValue)); END_WRAP }	cv::warpPerspective()	group__imgproc__transform.html	Applies a perspective transformation to an image.	src : input image. dst : output image that has the size dsize and the same type as src . M : ¥(3¥times 3¥) transformation matrix. dsize : size of the output image. flags : combination of interpolation methods (INTER_LINEAR or INTER_NEAREST) and the optional flag WARP_INVERSE_MAP, that sets M as the inverse transformation (¥[¥texttt{dst}¥rightarrow¥texttt{src}¥]). borderMode : pixel extrapolation method (BORDER_CONSTANT or BORDER_REPLICATE). borderValue : value used in case of a constant border; by default, it equals 0.	The function warpPerspective transforms the source image using the specified matrix:¥[¥texttt{dst} (x,y) = ¥texttt{src} ¥left (¥frac{M_{11} x + M_{12} y + M_{13}}{M_{31} x + M_{32} y + M_{33}} , ¥frac{M_{21} x + M_{22} y + M_{23}}{M_{31} x + M_{32} y + M_{33}} ¥right)¥]when the flag WARP_INVERSE_MAP is set. Otherwise, the transformation is first inverted with invert and then put in the formula above instead of M. The function cannot operate in-place.See alsowarpAffine, resize, remap, getRectSubPix, perspectiveTransformExamples: samples/cpp/image_alignment.cpp, and samples/dnn/text_detection.cpp.	7								
imgproc_warpPerspective_MisArray	CVAPI(ExceptionStatus) imgproc_warpPerspective_MisArray(cv::_InputArray* src, cv::_OutputArray* dst, float* m, int mRow, int mCol, MyCvSize dsize, int flags, int borderMode, MyCvScalar borderValue) { BEGIN_WRAP const cv::Mat mmat(mRow, mCol, CV_32FC1, m); cv::warpPerspective(*src, *dst, mmat, cpp(dsize), flags, borderMode, cpp(borderValue)); END_WRAP }	cv::warpPerspective()	group__imgproc__transform.html	Applies a perspective transformation to an image.	src : input image. dst : output image that has the size dsize and the same type as src . M : ¥(3¥times 3¥) transformation matrix. dsize : size of the output image. flags : combination of interpolation methods (INTER_LINEAR or INTER_NEAREST) and the optional flag WARP_INVERSE_MAP, that sets M as the inverse transformation (¥[¥texttt{dst}¥rightarrow¥texttt{src}¥]). borderMode : pixel extrapolation method (BORDER_CONSTANT or BORDER_REPLICATE). borderValue : value used in case of a constant border; by default, it equals 0.	The function warpPerspective transforms the source image using the specified matrix:¥[¥texttt{dst} (x,y) = ¥texttt{src} ¥left (¥frac{M_{11} x + M_{12} y + M_{13}}{M_{31} x + M_{32} y + M_{33}} , ¥frac{M_{21} x + M_{22} y + M_{23}}{M_{31} x + M_{32} y + M_{33}} ¥right)¥]when the flag WARP_INVERSE_MAP is set. Otherwise, the transformation is first inverted with invert and then put in the formula above instead of M. The function cannot operate in-place.See alsowarpAffine, resize, remap, getRectSubPix, perspectiveTransformExamples: samples/cpp/image_alignment.cpp, and samples/dnn/text_detection.cpp.	7								
imgproc_remap	CVAPI(ExceptionStatus) imgproc_remap(cv::_InputArray* src, cv::_OutputArray* dst, cv::_InputArray* map1, cv::_InputArray* map2, int interpolation, int borderMode, MyCvScalar borderValue) { BEGIN_WRAP cv::remap(*src, *dst, *map1, *map2, interpolation, borderMode, cpp(borderValue)); END_WRAP }	cv::remap()	group__imgproc__transform.html	Applies a generic geometrical transformation to an image.	src : Source image. dst : Destination image. It has the same size as map1 and the same type as src . map1 : The first map of either (x,y) points or just x values having the type CV_16SC2 , CV_32FC1, or CV_32FC2. See convertMaps for details on converting a floating point representation to fixed-point for speed. map2 : The second map of y values having the type CV_16UC1, CV_32FC1, or none (empty map if map1 is (x,y) points), respectively. interpolation : Interpolation method (see InterpolationFlags). The methods INTER_AREA and INTER_LINEAR_EXACT are not supported by this function. borderMode : Pixel extrapolation method (see BorderTypes). When borderMode=BORDER_TRANSPARENT, it means that the pixels in the destination image that corresponds to the "outliers" in the source image are not modified by the function. borderValue : Value used in case of a constant border. By default, it is 0.	The function remap transforms the source image using the specified map:¥[¥texttt{dst} (x,y) = ¥texttt{src} (map_x(x,y),map_y(x,y))¥]where values of pixels with non-integer coordinates are computed using one of available interpolation methods. ¥(map_x¥) and ¥(map_y¥) can be encoded as separate floating-point maps in ¥(map_1¥) and ¥(map_2¥) respectively, or interleaved floating-point maps of ¥((x,y)¥) in ¥(map_1¥), or fixed-point maps created by using convertMaps. The reason you might want to convert from floating to fixed-point representations of a map is that they can yield much faster (2×) remapping operations. In the converted case, ¥(map_1¥) contains pairs (cvFloor(x), cvFloor(y)) and ¥(map_2¥) contains indices in a table of interpolation coefficients.This function cannot operate in-place.NotesDue to current implementation limitations the size of an input and output images should be less	7								
imgproc_convertMaps	CVAPI(ExceptionStatus) imgproc_convertMaps(cv::_InputArray* map1, cv::_InputArray* map2, cv::_OutputArray* dstmap1, cv::_OutputArray* dstmap2, int dstmap1type, int nnInterpolation) { BEGIN_WRAP cv::convertMaps(*map1, *map2, *dstmap1, *dstmap2, dstmap1type, nnInterpolation != 0); END_WRAP }	cv::convertMaps()	group__imgproc__transform.html	Converts image transformation maps from one representation to another.	map1 : The first input map of type CV_16SC2, CV_32FC1, or CV_32FC2 . map2 : The second input map of type CV_16UC1, CV_32FC1, or none (empty matrix), respectively. dstmap1 : The first output map that has the type dstmap1type and the same size as src . dstmap2 : The second output map. dstmap1type : Type of the first output map that should be CV_16SC2, CV_32FC1, or CV_32FC2 . nninterpolation : Flag indicating whether the fixed-point maps are used for the nearest-neighbor or for a more complex interpolation.	The function converts a pair of maps for remap from one representation to another. The following options ((map1.type(), map2.type()) ¥[¥rightarrow¥] (dstmap1.type(), dstmap2.type())) are supported:¥(¥texttt{(CV_32FC1, CV_32FC1)} ¥rightarrow ¥texttt{(CV_16SC2, CV_16UC1)}¥). This is the most frequently used conversion operation, in which the original floating-point maps (see remap) are converted to a more compact and much faster fixed-point representation. The first output array contains the rounded coordinates and the second array (created only when nninterpolation=false) contains indices in the interpolation tables. ¥[¥texttt{(CV_32FC2)} ¥rightarrow ¥texttt{(CV_16SC2, CV_16UC1)}¥]. The same as above but the original maps are stored in one 2-channel matrix. Reverse conversion. Obviously, the reconstructed floating-point maps will not be exactly the same as the originals.See alsoremap, undistort, initUndistortRectifyMap	6								
imgproc_getRotationMatrix2D	CVAPI(ExceptionStatus) imgproc_getRotationMatrix2D(MyCvPoint2D32f center, double angle, double scale, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::getRotationMatrix2D(cpp(center), angle, scale); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getRotationMatrix2D()	group__imgproc__transform.html	Calculates an affine matrix of 2D rotation.	center : Center of the rotation in the source image. angle : Rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner). scale : Isotropic scale factor.	The function calculates the following matrix:¥[¥begin{bmatrix} ¥alpha & ¥beta & (1- ¥alpha) ¥cdot ¥texttt{center.x} - ¥beta ¥cdot ¥texttt{center.y} ¥¥ - ¥beta & ¥alpha & ¥beta ¥cdot ¥texttt{center.x} + (1- ¥alpha) ¥cdot ¥texttt{center.y} ¥end{bmatrix}¥]where¥[¥begin{array}{l} ¥alpha = ¥texttt{scale} ¥cdot ¥cos ¥texttt{angle} , ¥¥ ¥beta = ¥texttt{scale} ¥cdot ¥sin ¥texttt{angle} ¥end{array}¥]The transformation maps the rotation center to itself. If this is not the target, adjust the shift.See alsogetAffineTransform, warpAffine, transform	3								

imgproc_invertAffineTransform	CVAPI(ExceptionStatus) imgproc_invertAffineTransform(cv::_InputArray* M, cv::_OutputArray *IM) { BEGIN_WRAP cv::invertAffineTransform(*M, *IM); END_WRAP }	cv::invertAffineTransform()	group__imgproc__transform.html	Inverts an affine transformation.	M : Original affine transformation. iM : Output reverse affine transformation.	The function computes an inverse affine transformation represented by ¥(2 ¥times 3¥) matrix M:¥[¥begin{bmatrix} a_{11} & a_{12} & b_1 ¥¥ a_{21} & a_{22} & b_2 ¥end{bmatrix}¥]The result is also a ¥(2 ¥times 3¥) matrix of the same type as M.	2								
imgproc_getPerspectiveTransform1	CVAPI(ExceptionStatus) imgproc_getPerspectiveTransform1(cv::Point2f *src, cv::Point2f *dst, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::getPerspectiveTransform(src, dst); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getPerspectiveTransform()	group__imgproc__transform.html	Calculates a perspective transform from four pairs of the corresponding points.	src : Coordinates of quadrangle vertices in the source image. dst : Coordinates of the corresponding quadrangle vertices in the destination image. solveMethod : method passed to cv::solve (DecompTypes)	The function calculates the ¥(3 ¥times 3¥) matrix of a perspective transform so that:¥[¥begin{bmatrix} t_i x'_i ¥¥ t_i y'_i ¥¥ t_i ¥end{bmatrix} = ¥texttt{map_matrix} ¥cdot ¥begin{bmatrix} x_i ¥¥ y_i ¥¥ 1 ¥end{bmatrix}¥]where¥[dst[i]=(x'_i,y'_i), src[i]=(x_i, y_i), i=0,1,2,3¥]See alsofindHomography, warpPerspective, perspectiveTransformExamples: samples/cpp/warpPerspective_demo.cpp, and samples/dnn/text_detection.cpp.	2								
imgproc_getPerspectiveTransform2	CVAPI(ExceptionStatus) imgproc_getPerspectiveTransform2(cv::_InputArray *src, cv::_InputArray *dst, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::getPerspectiveTransform(*src, *dst); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getPerspectiveTransform()	group__imgproc__transform.html	Calculates a perspective transform from four pairs of the corresponding points.	src : Coordinates of quadrangle vertices in the source image. dst : Coordinates of the corresponding quadrangle vertices in the destination image. solveMethod : method passed to cv::solve (DecompTypes)	The function calculates the ¥(3 ¥times 3¥) matrix of a perspective transform so that:¥[¥begin{bmatrix} t_i x'_i ¥¥ t_i y'_i ¥¥ t_i ¥end{bmatrix} = ¥texttt{map_matrix} ¥cdot ¥begin{bmatrix} x_i ¥¥ y_i ¥¥ 1 ¥end{bmatrix}¥]where¥[dst[i]=(x'_i,y'_i), src[i]=(x_i, y_i), i=0,1,2,3¥]See alsofindHomography, warpPerspective, perspectiveTransformExamples: samples/cpp/warpPerspective_demo.cpp, and samples/dnn/text_detection.cpp.	2								
imgproc_getAffineTransform1	CVAPI(ExceptionStatus) imgproc_getAffineTransform1(cv::Point2f *src, cv::Point2f *dst, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::getAffineTransform(src, dst); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getAffineTransform()	group__imgproc__transform.html	Calculates an affine transform from three pairs of the corresponding points.	src : Coordinates of triangle vertices in the source image. dst : Coordinates of the corresponding triangle vertices in the destination image.	The function calculates the ¥(2 ¥times 3¥) matrix of an affine transform so that:¥[¥begin{bmatrix} x'_i ¥¥ y'_i ¥end{bmatrix} = ¥texttt{map_matrix} ¥cdot ¥begin{bmatrix} x_i ¥¥ y_i ¥¥ 1 ¥end{bmatrix}¥]where¥[dst[i]=(x'_i,y'_i), src[i]=(x_i, y_i), i=0,1,2¥]See alsowarpAffine, transform	2								
imgproc_getAffineTransform2	CVAPI(ExceptionStatus) imgproc_getAffineTransform2(cv::_InputArray *src, cv::_InputArray *dst, cv::Mat** returnValue) { BEGIN_WRAP const auto ret = cv::getAffineTransform(*src, *dst); *returnValue = new cv::Mat(ret); END_WRAP }	cv::getAffineTransform()	group__imgproc__transform.html	Calculates an affine transform from three pairs of the corresponding points.	src : Coordinates of triangle vertices in the source image. dst : Coordinates of the corresponding triangle vertices in the destination image.	The function calculates the ¥(2 ¥times 3¥) matrix of an affine transform so that:¥[¥begin{bmatrix} x'_i ¥¥ y'_i ¥end{bmatrix} = ¥texttt{map_matrix} ¥cdot ¥begin{bmatrix} x_i ¥¥ y_i ¥¥ 1 ¥end{bmatrix}¥]where¥[dst[i]=(x'_i,y'_i), src[i]=(x_i, y_i), i=0,1,2¥]See alsowarpAffine, transform	2								
imgproc_getRectSubPix	CVAPI(ExceptionStatus) imgproc_getRectSubPix(cv::_InputArray *image, MyCvSize patchSize, MyCvPoint2D32f center, cv::_OutputArray *patch, int patchType) { BEGIN_WRAP cv::getRectSubPix(*image, cpp(patchSize), cpp(center), *patch, patchType); END_WRAP }	cv::getRectSubPix()	group__imgproc__transform.html	Retrieves a pixel rectangle from an image with sub-pixel accuracy.	image : Source image. patchSize : Size of the extracted patch. center : Floating point coordinates of the center of the extracted rectangle within the source image. The center must be inside the image. patch : Extracted patch that has the size patchSize and the same number of channels as src . patchType : Depth of the extracted pixels. By default, they have the same depth as src .	The function getRectSubPix extracts pixels from src:¥[patch(x, y) = src(x + ¥texttt{center.x} - (¥texttt{dst.cols} - 1)*0.5, y + ¥texttt{center.y} - (¥texttt{dst.rows} -1)*0.5)¥]where the values of the pixels at non-integer coordinates are retrieved using bilinear interpolation. Every channel of multi-channel images is processed independently. Also the image should be a single channel or three channel image. While the center of the rectangle must be inside the image, parts of the rectangle may be outside.See alsowarpAffine, warpPerspective.	5								
imgproc_logPolar	CVAPI(ExceptionStatus) imgproc_logPolar(cv::_InputArray *src, cv::_OutputArray *dst, MyCvPoint2D32f center, double M, int flags) { BEGIN_WRAP cv::logPolar(*src, *dst, cpp(center), M, flags); END_WRAP }	cv::logPolar()	group__imgproc__transform.html	Remaps an image to semilog-polar coordinates space.		Deprecated:This function produces same result as cv::warpPolar(src, dst, src.size(), center, maxRadius, flags+WARP_POLAR_LOG);Examples: samples/cpp/polar_transforms.cpp.	5								
imgproc_linearPolar	CVAPI(ExceptionStatus) imgproc_linearPolar(cv::_InputArray *src, cv::_OutputArray *dst, MyCvPoint2D32f center, double maxRadius, int flags) { BEGIN_WRAP cv::linearPolar(*src, *dst, cpp(center), maxRadius, flags); END_WRAP }	cv::linearPolar()	group__imgproc__transform.html	Remaps an image to polar coordinates space.		Deprecated:This function produces same result as cv::warpPolar(src, dst, src.size(), center, maxRadius, flags)Examples: samples/cpp/polar_transforms.cpp.	5								
imgproc_warpPolar	CVAPI(ExceptionStatus) imgproc_warpPolar(cv::_InputArray *src, cv::_OutputArray *dst, MyCvSize dsize, MyCvPoint2D32f center, double maxRadius, int flags) { BEGIN_WRAP cv::warpPolar(*src, *dst, cpp(dsize), cpp(center), maxRadius, flags); END_WRAP }	cv::warpPolar()	group__imgproc__transform.html	Remaps an image to polar or semilog-polar coordinates space.	src : Source image. dst : Destination image. It will have same type as src. dsize : The destination image size (see description for valid options). center : The transformation center. maxRadius : The radius of the bounding circle to transform. It determines the inverse magnitude scale parameter too. flags : A combination of interpolation methods, InterpolationFlags + WarpPolarMode. Add WARP_POLAR_LINEAR to select linear polar mapping (default) Add WARP_POLAR_LOG to select semilog polar mapping Add WARP_INVERSE_MAP for reverse mapping.	Polar remaps referenceTransform the source image using the following transformation:¥[dst(¥rho , ¥phi) = src(x,y) ¥]where¥[¥begin{array}{l} ¥vec{I} = (x - center.x, ¥;y - center.y) ¥¥ ¥phi = ¥angle ¥cdot ¥texttt{angle} ¥¥ ¥rho = ¥left(¥begin{matrix} Klin ¥cdot ¥texttt{magnitude} ¥¥ Klog ¥cdot log_e(¥texttt{magnitude}) ¥end{matrix}¥right. & if ¥; semilog ¥¥ ¥end{array} ¥]and¥[¥begin{array}{l} ¥angle = dsize.height / 2¥Pi ¥¥ Klin = dsize.width / maxRadius ¥¥ Klog = dsize.width / log_e(maxRadius) ¥¥ ¥end{array} ¥]Linear vs semilog mappingPolar mapping can be linear or semi-log. Add one of WarpPolarMode to flags to specify the polar mapping mode.Linear is the default mode.The semilog mapping emulates the human "foveal" vision that permit very high acuity on the line of sight (central vision) in contrast to peripheral vision where acuity is minor.Option on dsize:if both values in dsize <=0 (default), the destination image will have (almost)	6								
imgproc_integral1	CVAPI(ExceptionStatus) imgproc_integral1(cv::_InputArray *src, cv::_OutputArray *sum, int sdepth) { BEGIN_WRAP cv::integral(*src, *sum, sdepth); END_WRAP }	cv::integral()	group__imgproc__misc.html	Calculates the integral of an image.	src : input image as ¥(W ¥times H¥), 8-bit or floating-point (32f or 64f). sum : integral image as ¥((W+1)¥times (H+1)¥) , 32-bit integer or floating-point (32f or 64f). sqsum : integral image for squared pixel values; it is ¥((W+1)¥times (H+1)¥), double-precision floating-point (64f) array. tilted : integral for the image rotated by 45 degrees; it is ¥((W+1)¥times (H+1)¥) array with the same data type as sum. sdepth : desired depth of the integral and the tilted integral images, CV_32S, CV_32F, or CV_64F. sdpdepth : desired depth of the integral image of squared pixel values, CV_32F or CV_64F.	The function calculates one or more integral images for the source image as follows:¥[¥texttt{sum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)¥]¥[¥texttt{sqsum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)^2¥]¥[¥texttt{tilted} (X,Y) = ¥sum _{y<Y,abs(x-X+1) ¥leq Y-y-1} ¥texttt{image} (x,y)¥]Using these integral images, you can calculate sum, mean, and standard deviation over a specific up-right or rotated rectangular region of the image in a constant time, for example:¥[¥sum _{x_1 ¥leq x < x_2, ¥, y_1 ¥leq y < y_2} ¥texttt{image} (x,y) = ¥texttt{sum} (x_2,y_2)- ¥texttt{sum} (x_1,y_2)- ¥texttt{sum} (x_2,y_1)+ ¥texttt{sum} (x_1,y_1)¥]It makes possible to do a fast blurring or fast block correlation with a variable window size, for example. In case of multi-channel images, sums for each channel are accumulated independently.As a practical example, the next figure shows the calculation of the integral of a straight rectangle Rect(3,3,3,2) and of a tilted rectangle Rect(5,1,2,3). The selected	3								
imgproc_integral2	CVAPI(ExceptionStatus) imgproc_integral2(cv::_InputArray *src, cv::_OutputArray *sum, cv::_OutputArray *sqsum, int sdepth) { BEGIN_WRAP cv::integral(*src, *sum, *sqsum, sdepth); END_WRAP }	cv::integral()	group__imgproc__misc.html	Calculates the integral of an image.	src : input image as ¥(W ¥times H¥), 8-bit or floating-point (32f or 64f). sum : integral image as ¥((W+1)¥times (H+1)¥) , 32-bit integer or floating-point (32f or 64f). sqsum : integral image for squared pixel values; it is ¥((W+1)¥times (H+1)¥), double-precision floating-point (64f) array. tilted : integral for the image rotated by 45 degrees; it is ¥((W+1)¥times (H+1)¥) array with the same data type as sum. sdepth : desired depth of the integral and the tilted integral images, CV_32S, CV_32F, or CV_64F. sdpdepth : desired depth of the integral image of squared pixel values, CV_32F or CV_64F.	The function calculates one or more integral images for the source image as follows:¥[¥texttt{sum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)¥]¥[¥texttt{sqsum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)^2¥]¥[¥texttt{tilted} (X,Y) = ¥sum _{y<Y,abs(x-X+1) ¥leq Y-y-1} ¥texttt{image} (x,y)¥]Using these integral images, you can calculate sum, mean, and standard deviation over a specific up-right or rotated rectangular region of the image in a constant time, for example:¥[¥sum _{x_1 ¥leq x < x_2, ¥, y_1 ¥leq y < y_2} ¥texttt{image} (x,y) = ¥texttt{sum} (x_2,y_2)- ¥texttt{sum} (x_1,y_2)- ¥texttt{sum} (x_2,y_1)+ ¥texttt{sum} (x_1,y_1)¥]It makes possible to do a fast blurring or fast block correlation with a variable window size, for example. In case of multi-channel images, sums for each channel are accumulated independently.As a practical example, the next figure shows the calculation of the integral of a straight rectangle Rect(3,3,3,2) and of a tilted rectangle Rect(5,1,2,3). The selected	4								

imgproc_integral3	CVAPI(ExceptionStatus) imgproc_integral3(cv::_InputArray *src, cv::_OutputArray *sum, cv::_OutputArray *sqsum, cv::_OutputArray *tilted, int sdepth, int sqdepth) { BEGIN_WRAP cv::integral(*src, *sum, *sqsum, *tilted, sdepth, sqdepth); END_WRAP }	cv::integral()	group__imgproc__misc.html	Calculates the integral of an image.	src : input image as ¥(W ¥times H¥), 8-bit or floating-point (32f or 64f). sum : integral image as ¥((W+1)¥times (H+1)¥) , 32-bit integer or floating-point (32f or 64f). sqsum : integral image for squared pixel values; it is ¥((W+1)¥times (H+1)¥), double-precision floating-point (64f) array. tilted : integral for the image rotated by 45 degrees; it is ¥((W+1)¥times (H+1)¥) array with the same data type as sum. sdepth : desired depth of the integral and the tilted integral images, CV_32S, CV_32F, or CV_64F. sqdepth : desired depth of the integral image of squared pixel values, CV_32F or CV_64F.	The function calculates one or more integral images for the source image as follows:¥[¥texttt{sum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)¥]¥[¥texttt{sqsum} (X,Y) = ¥sum _{x<X,y<Y} ¥texttt{image} (x,y)^2¥]¥[¥texttt{tilted} (X,Y) = ¥sum _{y<Y,abs(x-X+1) ¥leq Y-y-1} ¥texttt{image} (x,y)¥]Using these integral images, you can calculate sum, mean, and standard deviation over a specific up-right or rotated rectangular region of the image in a constant time, for example:¥[¥sum _{x_1 ¥leq x < x_2, ¥ y_1 ¥leq y < y_2} ¥texttt{image} (x,y) = ¥texttt{sum} (x_2,y_2)- ¥texttt{sum} (x_1,y_2)- ¥texttt{sum} (x_2,y_1)+ ¥texttt{sum} (x_1,y_1)¥]It makes possible to do a fast blurring or fast block correlation with a variable window size, for example. In case of multi-channel images, sums for each channel are accumulated independently.As a practical example, the next figure shows the calculation of the integral of a straight rectangle Rect(3,3,3,2) and of a tilted	6								
imgproc_accumulate	CVAPI(ExceptionStatus) imgproc_accumulate(cv::_InputArray *src, cv::_InputOutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::accumulate(*src, *dst, entity(mask)); END_WRAP }	cv::accumulate()	group__imgproc__motion.html	Adds an image to the accumulator image.	src : Input image of type CV_8UC(n), CV_16UC(n), CV_32FC(n) or CV_64FC(n), where n is a positive integer. dst : Accumulator image with the same number of channels as input image, and a depth of CV_32F or CV_64F. mask : Optional operation mask.	The function adds src or some of its elements to dst :¥[¥texttt{dst} (x,y) ¥leftarrow ¥texttt{dst} (x,y) + ¥texttt{src} (x,y) ¥quad ¥text{if} ¥quad ¥texttt{mask} (x,y) ¥ne 0¥]The function supports multi-channel images. Each channel is processed independently.The function cv::accumulate can be used, for example, to collect statistics of a scene background viewed by a still camera and for the further foreground-background segmentation.See alsoaccumulateSquare, accumulateProduct, accumulateWeighted	3								
imgproc_accumulateSquare	CVAPI(ExceptionStatus) imgproc_accumulateSquare(cv::_InputArray* src, cv::_InputOutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::accumulateSquare(*src, *dst, entity(mask)); END_WRAP }	cv::accumulateSquare()	group__imgproc__motion.html	Adds the square of a source image to the accumulator image.	src : Input image as 1- or 3-channel, 8-bit or 32-bit floating point. dst : Accumulator image with the same number of channels as input image, 32-bit or 64-bit floating-point. mask : Optional operation mask.	The function adds the input image src or its selected region, raised to a power of 2, to the accumulator dst :¥[¥texttt{dst} (x,y) ¥leftarrow ¥texttt{dst} (x,y) + ¥texttt{src} (x,y)^2 ¥quad ¥text{if} ¥quad ¥texttt{mask} (x,y) ¥ne 0¥]The function supports multi-channel images. Each channel is processed independently.See alsoaccumulateSquare, accumulateProduct, accumulateWeighted	3								
imgproc_accumulateProduct	CVAPI(ExceptionStatus) imgproc_accumulateProduct(cv::_InputArray *src1, cv::_InputArray *src2, cv::_InputOutputArray *dst, cv::_InputArray *mask) { BEGIN_WRAP cv::accumulateProduct(*src1, *src2, *dst, entity(mask)); END_WRAP }	cv::accumulateProduct()	group__imgproc__motion.html	Adds the per-element product of two input images to the accumulator image.	src1 : First input image, 1- or 3-channel, 8-bit or 32-bit floating point. src2 : Second input image of the same type and the same size as src1 . dst : Accumulator image with the same number of channels as input images, 32-bit or 64-bit floating-point. mask : Optional operation mask.	The function adds the product of two images or their selected regions to the accumulator dst :¥[¥texttt{dst} (x,y) ¥leftarrow ¥texttt{dst} (x,y) + ¥texttt{src1} (x,y) ¥cdot ¥texttt{src2} (x,y) ¥quad ¥text{if} ¥quad ¥texttt{mask} (x,y) ¥ne 0¥]The function supports multi-channel images. Each channel is processed independently.See alsoaccumulate, accumulateSquare, accumulateWeighted	4								
imgproc_accumulateWeighted	CVAPI(ExceptionStatus) imgproc_accumulateWeighted(cv::_InputArray *src, cv::_InputOutputArray *dst, double alpha, cv::_InputArray *mask) { BEGIN_WRAP cv::accumulateWeighted(*src, *dst, alpha, entity(mask)); END_WRAP }	cv::accumulateWeighted()	group__imgproc__motion.html	Updates a running average.	src : Input image as 1- or 3-channel, 8-bit or 32-bit floating point. dst : Accumulator image with the same number of channels as input image, 32-bit or 64-bit floating-point. alpha : Weight of the input image. mask : Optional operation mask.	The function calculates the weighted sum of the input image src and the accumulator dst so that dst becomes a running average of a frame sequence:¥[¥texttt{dst} (x,y) ¥leftarrow (1- ¥texttt{alpha}) ¥cdot ¥texttt{dst} (x,y) + ¥texttt{alpha} ¥cdot ¥texttt{src} (x,y) ¥quad ¥text{if} ¥quad ¥texttt{mask} (x,y) ¥ne 0¥]That is, alpha regulates the update speed (how fast the accumulator "forgets" about earlier images). The function supports multi-channel images. Each channel is processed independently.See alsoaccumulate, accumulateSquare, accumulateProduct	4								
imgproc_phaseCorrelate	CVAPI(ExceptionStatus) imgproc_phaseCorrelate(cv::_InputArray *src1, cv::_InputArray *src2, cv::_InputArray *window, double* response, MyCvPoint2D64f* returnValue) { BEGIN_WRAP const auto p = cv::phaseCorrelate(*src1, *src2, *window, response); *returnValue = { p.x, p.y }; END_WRAP }	cv::phaseCorrelate()	group__imgproc__motion.html	The function is used to detect translational shifts that occur between two images.	src1 : Source floating point array (CV_32FC1 or CV_64FC1) src2 : Source floating point array (CV_32FC1 or CV_64FC1) window : Floating point array with windowing coefficients to reduce edge effects (optional). response : Signal power within the 5x5 centroid around the peak, between 0 and 1 (optional).	The operation takes advantage of the Fourier shift theorem for detecting the translational shift in the frequency domain. It can be used for fast image registration as well as motion estimation. For more information please see http://en.wikipedia.org/wiki/Phase_correlationCalculates the cross-power spectrum of two supplied source arrays. The arrays are padded if needed with getOptimalDFTSize.The function performs the following equations:First it applies a Hanning window (see http://en.wikipedia.org/wiki/Hann_function) to each image to remove possible edge effects. This window is cached until the array size changes to speed up processing time.Next it computes the forward DFTs of each source array:¥[¥mathbf{G}_a = ¥mathcal{F}¥(src_1¥), ¥; ¥mathbf{G}_b = ¥mathcal{F}¥(src_2¥)¥] where ¥(¥mathcal{F}¥) is the forward DFT. It then computes the cross-power	4								
imgproc_createHanningWindow	CVAPI(ExceptionStatus) imgproc_createHanningWindow(cv::_OutputArray *dst, MyCvSize winSize, int type) { BEGIN_WRAP cv::createHanningWindow(*dst, cpp(winSize), type); END_WRAP }	cv::createHanningWindow()	group__imgproc__motion.html	This function computes a Hanning window coefficients in two dimensions.	dst : Destination array to place Hann coefficients in winSize : The window size specifications (both width and height must be > 1) type : Created array type	See (http://en.wikipedia.org/wiki/Hann_function) and (http://en.wikipedia.org/wiki/Window_function) for more information.An example is shown below:// create hanning window of size 100x100 and type CV_32FMat hann;createHanningWindow(hann, Size(100, 100), CV_32F);fragment	3								
imgproc_threshold	CVAPI(ExceptionStatus) imgproc_threshold(cv::_InputArray *src, cv::_OutputArray *dst, double thresh, double maxVal, int type, double *returnValue) { BEGIN_WRAP *returnValue = cv::threshold(*src, *dst, thresh, maxVal, type); END_WRAP }	cv::threshold()	group__imgproc__misc.html	Applies a fixed-level threshold to each array element.	src : input array (multiple-channel, 8-bit or 32-bit floating point). dst : output array of the same size and type and the same number of channels as src. thresh : threshold value. maxval : maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV thresholding types. type : thresholding type (see ThresholdTypes).	The function applies fixed-level thresholding to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image (compare could be also used for this purpose) or for removing a noise, that is, filtering out pixels with too small or too large values. There are several types of thresholding supported by the function. They are determined by type parameter.Also, the special values THRESH_OTSU or THRESH_TRIANGLE may be combined with one of the above values. In these cases, the function determines the optimal threshold value using the Otsu's or Triangle algorithm and uses it instead of the specified thresh.NoteCurrently, the Otsu's and Triangle methods are implemented only for 8-bit single-channel images.See alsoadaptiveThreshold, findContours, compare, min, maxExamples: samples/cpp/ffilldemo.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, and	5								
imgproc_adaptiveThreshold	CVAPI(ExceptionStatus) imgproc_adaptiveThreshold(cv::_InputArray *src, cv::_OutputArray *dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C) { BEGIN_WRAP cv::adaptiveThreshold(*src, *dst, maxValue, adaptiveMethod, thresholdType, blockSize, C); END_WRAP }	cv::adaptiveThreshold()	group__imgproc__misc.html	Applies an adaptive threshold to an array.	src : Source 8-bit single-channel image. dst : Destination image of the same size and the same type as src. maxValue : Non-zero value assigned to the pixels for which the condition is satisfied adaptiveMethod : Adaptive thresholding algorithm to use, see AdaptiveThresholdTypes. The BORDER_REPLICATE | BORDER_ISOLATED is used to process boundaries. thresholdType : Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV, see ThresholdTypes. blockSize : Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on. C : Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.	The function transforms a grayscale image to a binary image according to the formulae:THRESH_BINARY ¥[dst(x,y) = ¥fork{¥texttt{maxValue}}{if ¥(src(x,y) > T(x,y)¥)}{0}{otherwise}¥] THRESH_BINARY_INV ¥[dst(x,y) = ¥fork{0}{if ¥(src(x,y) > T(x,y)¥)}{¥texttt{maxValue}}{otherwise}¥] where ¥(T(x,y)¥) is a threshold calculated individually for each pixel (see adaptiveMethod parameter).The function can process the image in-place.See alsothreshold, blur, GaussianBlur	7								
imgproc_pyrDown	CVAPI(ExceptionStatus) imgproc_pyrDown(cv::_InputArray *src, cv::_OutputArray *dst, MyCvSize dstSize, int borderType) { BEGIN_WRAP cv::pyrDown(*src, *dst, cpp(dstSize), borderType); END_WRAP }	cv::pyrDown()	group__imgproc__filter.html	Blurs an image and downsamples it.	src : input image. dst : output image; it has the specified size and the same type as src. dstsize : size of the output image. borderType : Pixel extrapolation method, see BorderTypes (BORDER_CONSTANT isn't supported)	By default, size of the output image is computed as Size((src.cols+1)/2, (src.rows+1)/2), but in any case, the following conditions should be satisfied:¥[¥begin{array}{l} | ¥texttt{dstsize.width} *2-src.cols| ¥leq 2 ¥¥ | ¥texttt{dstsize.height} *2-src.rows| ¥leq 2 ¥end{array}¥]The function performs the downsampling step of the Gaussian pyramid construction. First, it convolves the source image with the kernel:¥[¥frac{1}{256} ¥begin{bmatrix} 1 & 4 & 6 & 4 & 1 ¥¥ 4 & 16 & 24 & 16 & 4 ¥¥ 6 & 24 & 36 & 24 & 6 ¥¥ 4 & 16 & 24 & 16 & 4 ¥¥ 1 & 4 & 6 & 4 & 1 ¥end{bmatrix}¥]Then, it downsamples the image by rejecting even rows and columns.Examples: samples/cpp/squares.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, and	4								
imgproc_pyrUp	CVAPI(ExceptionStatus) imgproc_pyrUp(cv::_InputArray *src, cv::_OutputArray *dst, MyCvSize dstSize, int borderType) { BEGIN_WRAP cv::pyrUp(*src, *dst, cpp(dstSize), borderType); END_WRAP }	cv::pyrUp()	group__imgproc__filter.html	Upsamples an image and then blurs it.	src : input image. dst : output image. It has the specified size and the same type as src . dstsize : size of the output image. borderType : Pixel extrapolation method, see BorderTypes (only BORDER_DEFAULT is supported)	By default, size of the output image is computed as Size(src.cols*2, (src.rows*2), but in any case, the following conditions should be satisfied:¥[¥begin{array}{l} | ¥texttt{dstsize.width} -src.cols*2| ¥leq (¥texttt{dstsize.width} ¥mod 2) ¥¥ | ¥texttt{dstsize.height} -src.rows*2| ¥leq (¥texttt{dstsize.height} ¥mod 2) ¥end{array}¥]The function performs the upsampling step of the Gaussian pyramid construction, though it can actually be used to construct the Laplacian pyramid. First, it upsamples the source image by injecting even zero rows and columns and then convolves the result with the same kernel as in pyrDown multiplied by 4.Examples: samples/cpp/squares.cpp, samples/cpp/tutorial_code/ImgProc/Pyramids/Pyramids.cpp, and	4								

imgproc_calcHist	CVAPI(ExceptionStatus) imgproc_calcHist(cv::Mat **images, int nimages, const int* channels, cv::_InputArray *mask, cv::_OutputArray *hist, int dims, const int* histSize, const float** ranges, int uniform, int accumulate) { BEGIN_WRAP std::vector<cv::Mat> imagesVec(nimages); for (auto i = 0; i < nimages; i++) imagesVec[i] = *((images[i]); cv::calcHist(&imagesVec[0], nimages, channels, entity(mask), *hist, dims, histSize, ranges, uniform != 0, accumulate != 0); END_WRAP }	cv::calcHist()	group__imgproc__hist.html	Calculates a histogram of a set of arrays.	images : Source arrays. They all should have the same depth, CV_8U, CV_16U or CV_32F , and the same size. Each of them can have an arbitrary number of channels. nimages : Number of source images. channels : List of the dims channels used to compute the histogram. The first array channels are numerated from 0 to images[0].channels()-1 , the second array channels are counted from images[0].channels() to images[0].channels() + images[1].channels()-1, and so on. mask : Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as images[0] . The non-zero mask elements mark the array elements counted in the histogram. hist : Output histogram, which is a dense or sparse dims -dimensional array. dims : Histogram dimensionality that must be positive and not greater than CV_MAX_DIMS (equal to 32 in the current OpenCV version). histSize : Array of histogram sizes in each dimension. ranges : Array of the dims arrays of the histogram bin boundaries in each dimension. When the histogram is uniform (uniform =true), then for each dimension i it is enough to specify the lower (inclusive) boundary ¥(L_0¥) of the 0-th histogram bin and the upper [exclusive] boundary ¥(L_{¥texttt{histSize}[i]-1}¥)	The function cv::calcHist calculates the histogram of one or more arrays. The elements of a tuple used to increment a histogram bin are taken from the corresponding input arrays at the same location. The sample below shows how to compute a 2D Hue-Saturation histogram for a color image. : #include <opencv2/imgproc.hpp> #include <opencv2/highgui.hpp> using namespace cv; int main(int argc, char** argv){ Mat src, hsv; if(argc != 2 || !(src=imread(argv[1], 1)).data) return -1; cvtColor(src, hsv, COLOR_BGR2HSV); // Quantize the hue to 30 levels // and the saturation to 32 levels int hbins = 30, sbins = 32; int histSize[] = {hbins, sbins}; // hue varies from 0 to 179, see cvtColor float hranges[] = { 0, 180 }; // saturation varies from 0 (black-gray-white) to // 255 (pure spectrum color) float sranges[] = { 0, 256 }; const float* ranges[] = { hranges, sranges }; MatND hist; // we compute the histogram from the 0-th and 1-st [illegible]	10									
imgproc_calcBackProject	CVAPI(ExceptionStatus) imgproc_calcBackProject(cv::Mat **images, int nimages, const int* channels, cv::_InputArray *hist, cv::_OutputArray *backProject, const float** ranges, int uniform) { BEGIN_WRAP std::vector<cv::Mat> imagesVec(nimages); for (auto i = 0; i < nimages; i++) imagesVec[i] = *((images[i]); cv::calcBackProject(&imagesVec[0], nimages, channels, *hist, *backProject, ranges, uniform != 0); END_WRAP }	cv::calcBackProject()	group__imgproc__hist.html	Calculates the back projection of a histogram.	images : Source arrays. They all should have the same depth, CV_8U, CV_16U or CV_32F , and the same size. Each of them can have an arbitrary number of channels. nimages : Number of source images. channels : The list of channels used to compute the back projection. The number of channels must match the histogram dimensionality. The first array channels are numerated from 0 to images[0].channels()-1 , the second array channels are counted from images[0].channels() to images[0].channels() + images[1].channels()-1, and so on. hist : Input histogram that can be dense or sparse. backProject : Destination back projection array that is a single-channel array of the same size and depth as images[0] . ranges : Array of arrays of the histogram bin boundaries in each dimension. See calcHist . scale : Optional scale factor for the output back projection. uniform : Flag indicating whether the histogram is uniform or not (see above).	The function cv::calcBackProject calculates the back project of the histogram. That is, similarly to calcHist , at each location (x, y) the function collects the values from the selected channels in the input images and finds the corresponding histogram bin. But instead of incrementing it, the function reads the bin value, scales it by scale , and stores in backProject(x,y) . In terms of statistics, the function computes probability of each element value in respect with the empirical probability distribution represented by the histogram. See how, for example, you can find and track a bright-colored object in a scene:Before tracking, show the object to the camera so that it covers almost the whole frame. Calculate a hue histogram. The histogram may have strong maximums, corresponding to the dominant colors in the object. When tracking, calculate a back projection of a hue plane of each input video frame using that pre-computed histogram. Threshold the back projection to suppress [illegible]	7								
imgproc_compareHist	CVAPI(ExceptionStatus) imgproc_compareHist(cv::_InputArray *h1, cv::_InputArray *h2, int method, double *returnValue) { BEGIN_WRAP *returnValue = cv::compareHist(*h1, *h2, method); END_WRAP }	cv::compareHist()	group__imgproc__hist.html	Compares two histograms.	H1 : First compared histogram. H2 : Second compared histogram of the same size as H1 . method : Comparison method, see HistCompMethods	The function cv::compareHist compares two dense or two sparse histograms using the specified method.The function returns ¥(d(H_1, H_2)¥) .While the function works well with 1-, 2-, 3-dimensional dense histograms, it may not be suitable for high-dimensional sparse histograms. In such histograms, because of aliasing and sampling problems, the coordinates of non-zero histogram bins can slightly shift. To compare such histograms or more general sparse configurations of weighted points, consider using the EMD function.	3								
imgproc_equalizeHist	CVAPI(ExceptionStatus) imgproc_equalizeHist(cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP cv::equalizeHist(*src, *dst); END_WRAP }	cv::equalizeHist()	group__imgproc__hist.html	Equalizes the histogram of a grayscale image.	src : Source 8-bit single channel image. dst : Destination image of the same size and type as src .	The function equalizes the histogram of the input image using the following algorithm:Calculate the histogram ¥(H¥) for src . Normalize the histogram so that the sum of histogram bins is 255. Compute the integral of the histogram: ¥[H'_i = ¥sum _{0 ¥le j < i} H[j]¥] Transform the image using ¥(H'¥) as a look-up table: ¥(¥texttt{dst}(x,y) = H'{¥texttt{src}(x,y)}¥)The algorithm normalizes the brightness and increases the contrast of the image.Examples: samples/cpp/tapedetect.cpp	2								
imgproc_EMD	CVAPI(ExceptionStatus) imgproc_EMD(cv::_InputArray *signature1, cv::_InputArray *signature2, int distType, cv::_InputArray *cost, float* lowerBound, cv::_OutputArray *flow, float* returnValue) { BEGIN_WRAP *returnValue = cv::EMD(*signature1, *signature2, distType, entity(cost), lowerBound, entity(flow)); END_WRAP }	cv::EMD()	group__imgproc__hist.html	Computes the "minimal work" distance between two weighted point configurations.	signature1 : First signature, a ¥(¥texttt{size1}¥times ¥texttt{dims}+1¥) floating-point matrix. Each row stores the point weight followed by the point coordinates. The matrix is allowed to have a single column (weights only) if the user-defined cost matrix is used. The weights must be non-negative and have at least one non-zero value. signature2 : Second signature of the same format as signature1 , though the number of rows may be different. The total weights may be different. In this case an extra "dummy" point is added to either signature1 or signature2. The weights must be non-negative and have at least one non-zero value. distType : Used metric. See DistanceTypes. cost : User-defined ¥(¥texttt{size1}¥times ¥texttt{size2}¥) cost matrix. Also, if a cost matrix is used, lower boundary lowerBound cannot be calculated because it needs a metric function. lowerBound : Optional input/output parameter: lower boundary of a distance between the two signatures that is a distance between mass centers. The lower boundary may not be calculated if the user-defined cost matrix is used, the total weights of point configurations are not equal, or if the signatures consist of weights only (the signature matrices have a single column). You must** initialize *lowerBound . If the calculated distance between mass centers is	The function computes the earth mover distance and/or a lower boundary of the distance between the two weighted point configurations. One of the applications described in [207], [208] is multi-dimensional histogram comparison for image retrieval. EMD is a transportation problem that is solved using some modification of a simplex algorithm, thus the complexity is exponential in the worst case, though, on average it is much faster. In the case of a real metric the lower boundary can be calculated even faster (using linear-time algorithm) and it can be used to determine roughly whether the two signatures are far enough so that they cannot relate to the same object.	6								
imgproc_watershed	CVAPI(ExceptionStatus) imgproc_watershed(cv::_InputArray *image, cv::_InputOutputArray *markers) { BEGIN_WRAP cv::watershed(*image, *markers); END_WRAP }	cv::watershed()	group__imgproc__segmentation.html	Performs a marker-based image segmentation using the watershed algorithm.	image : Input 8-bit 3-channel image. markers : Input/output 32-bit single-channel image (map) of markers. It should have the same size as image .	The function implements one of the variants of watershed, non-parametric marker-based segmentation algorithm, described in [169] .Before passing the image to the function, you have to roughly outline the desired regions in the image markers with positive (>0) indices. So, every region is represented as one or more connected components with the pixel values 1, 2, 3, and so on. Such markers can be retrieved from a binary mask using findContours and drawContours (see the watershed.cpp demo). The markers are "seeds" of the future image regions. All the other pixels in markers , whose relation to the outlined regions is not known and should be defined by the algorithm, should be set to 0's. In the function output, each pixel in markers is set to a value of the "seed" components or to -1 at boundaries between the regions.NoteAny two neighbor connected components are not necessarily separated by a watershed boundary (-1's pixels); for example, they can touch each other in the initial marker image passed to the function. For	2								
imgproc_pyrMeanShiftFiltering	CVAPI(ExceptionStatus) imgproc_pyrMeanShiftFiltering(cv::_InputArray *src, cv::_InputOutputArray *dst, double sp, double sr, int maxLevel, MyCvTermCriteria termCrit) { BEGIN_WRAP cv::pyrMeanShiftFiltering(*src, *dst, sp, sr, maxLevel, cpp(termCrit)); END_WRAP }	cv::pyrMeanShiftFiltering()	group__imgproc__filter.html	Performs initial step of meanshift segmentation of an image.	src : The source 8-bit, 3-channel image. dst : The destination image of the same format and the same size as the source. sp : The spatial window radius. sr : The color window radius. maxLevel : Maximum level of the pyramid for the segmentation. termcrit : Termination criteria: when to stop meanshift iterations.	The function implements the filtering stage of meanshift segmentation, that is, the output of the function is the filtered "posterized" image with color gradients and fine-grain texture flattened. At every pixel (X,Y) of the input image (or down-sized input image, see below) the function executes meanshift iterations, that is, the pixel (X,Y) neighborhood in the joint space-color hyperspace is considered:¥[(x,y): X- ¥texttt{sp} ¥le x ¥le X+ ¥texttt{sp} , Y- ¥texttt{sp} ¥le y ¥le Y+ ¥texttt{sp} , ||(R,G,B)-(r,g,b)|| ¥le ¥texttt{sr}¥]¥where (R,G,B) and (r,g,b) are the vectors of color components at (X,Y) and (x,y), respectively (though, the algorithm does not depend on the color space used, so any 3-component color space can be used instead). Over the neighborhood the average spatial value (X',Y') and average color vector (R',G',B') are found and they act as the neighborhood center on the next iteration:¥[(X,Y)-(X',Y'), (R,G,B)-(R',G',B').¥]After the iterations over, the color components of the initial pixel (that is, the pixel from where the	6								
imgproc_grabCut	CVAPI(ExceptionStatus) imgproc_grabCut(cv::_InputArray *img, cv::_InputOutputArray *mask, CvRect rect, cv::_InputOutputArray *bgdModel, cv::_InputOutputArray *fgdModel, int iterCount, int mode) { BEGIN_WRAP cv::grabCut(*img, *mask, rect, *bgdModel, *fgdModel, iterCount, mode); END_WRAP }	cv::grabCut()	group__imgproc__segmentation.html	Runs the GrabCut algorithm.	img : Input 8-bit 3-channel image. mask : Input/output 8-bit single-channel mask. The mask is initialized by the function when mode is set to GC_INIT_WITH_RECT. Its elements may have one of the GrabCutClasses. rect : ROI containing a segmented object. The pixels outside of the ROI are marked as "obvious background". The parameter is only used when mode==GC_INIT_WITH_RECT . bgdModel : Temporary array for the background model. Do not modify it while you are processing the same image. fgdModel : Temporary arrays for the foreground model. Do not modify it while you are processing the same image. iterCount : Number of iterations the algorithm should make before returning the result. Note that the result can be refined with further calls with mode==GC_INIT_WITH_MASK or mode==GC_EVAL . mode : Operation mode that could be one of the GrabCutModes	The function implements the GrabCut image segmentation algorithm.Examples: samples/cpp/grabcut.cpp.	7								

imgproc_distanceTransformWithLabels	CVAPI(ExceptionStatus) imgproc_distanceTransformWithLabels(cv::_InputArray *src, cv::_OutputArray *dst, cv::_OutputArray *labels, int distanceType, int maskSize, int labelType) { BEGIN_WRAP cv::distanceTransform(*src, *dst, *labels, distanceType, maskSize, labelType); END_WRAP }	cv::distanceTransform()	group__imgproc__misc.html	Calculates the distance to the closest zero pixel for each pixel of the source image.	src : 8-bit, single-channel (binary) source image. dst : Output image with calculated distances. It is a 8-bit or 32-bit floating-point, single-channel image of the same size as src. labels : Output 2D array of labels (the discrete Voronoi diagram). It has the type CV_32SC1 and the same size as src. distanceType : Type of distance, see DistanceTypes maskSize : Size of the distance transform mask, see DistanceTransformMasks. DIST_MASK_PRECISE is not supported by this variant. In case of the DIST_L1 or DIST_C distance type, the parameter is forced to 3 because a 3×3 mask gives the same result as 5×5 or any larger aperture. labelType : Type of the label array to build, see DistanceTransformLabelTypes.	The function cv::distanceTransform calculates the approximate or precise distance from every binary image pixel to the nearest zero pixel. For zero image pixels, the distance will obviously be zero.When maskSize == DIST_MASK_PRECISE and distanceType == DIST_L2 , the function runs the algorithm described in [72] . This algorithm is parallelized with the TBB library.In other cases, the algorithm [28] is used. This means that for a pixel the function finds the shortest path to the nearest zero pixel consisting of basic shifts: horizontal, vertical, diagonal, or knight's move (the latest is available for a 5×5 mask). The overall distance is calculated as a sum of these basic distances. Since the distance function should be symmetric, all of the horizontal and vertical shifts must have the same cost (denoted as a), all the diagonal shifts must have the same cost (denoted as b), and all knight's moves must have the same cost (denoted as c). For the DIST_C and DIST_L1 types, the	6								
imgproc_distanceTransform	CVAPI(ExceptionStatus) imgproc_distanceTransformWithLabels(cv::_InputArray *src, cv::_OutputArray *dst, cv::_OutputArray *labels, int distanceType, int maskSize, int labelType) { BEGIN_WRAP cv::distanceTransform(*src, *dst, *labels, distanceType, maskSize, labelType); END_WRAP }	cv::distanceTransform()	group__imgproc__misc.html	Calculates the distance to the closest zero pixel for each pixel of the source image.	src : 8-bit, single-channel (binary) source image. dst : Output image with calculated distances. It is a 8-bit or 32-bit floating-point, single-channel image of the same size as src. labels : Output 2D array of labels (the discrete Voronoi diagram). It has the type CV_32SC1 and the same size as src. distanceType : Type of distance, see DistanceTypes maskSize : Size of the distance transform mask, see DistanceTransformMasks. DIST_MASK_PRECISE is not supported by this variant. In case of the DIST_L1 or DIST_C distance type, the parameter is forced to 3 because a 3×3 mask gives the same result as 5×5 or any larger aperture. labelType : Type of the label array to build, see DistanceTransformLabelTypes.	The function cv::distanceTransform calculates the approximate or precise distance from every binary image pixel to the nearest zero pixel. For zero image pixels, the distance will obviously be zero.When maskSize == DIST_MASK_PRECISE and distanceType == DIST_L2 , the function runs the algorithm described in [72] . This algorithm is parallelized with the TBB library.In other cases, the algorithm [28] is used. This means that for a pixel the function finds the shortest path to the nearest zero pixel consisting of basic shifts: horizontal, vertical, diagonal, or knight's move (the latest is available for a 5×5 mask). The overall distance is calculated as a sum of these basic distances. Since the distance function should be symmetric, all of the horizontal and vertical shifts must have the same cost (denoted as a), all the diagonal shifts must have the same cost (denoted as b), and all knight's moves must have the same cost (denoted as c). For the DIST_C and DIST_L1 types, the	6								
imgproc_floodFill1	CVAPI(ExceptionStatus) imgproc_floodFill1(cv::_InputOutputArray *image, MyCvPoint seedPoint, MyCvScalar newVal, MyCvRect *rect, MyCvScalar loDiff, MyCvScalar upDiff, int flags, int *returnValue) { BEGIN_WRAP cv::Rect rect0; *returnValue = cv::floodFill(*image, cpp(seedPoint), cpp(newVal), &rect0, cpp(loDiff), cpp(upDiff), flags); *rect = c(rect0); END_WRAP }	cv::floodFill()	group__imgproc__misc.html	Fills a connected component with the given color.	image : Input/output 1- or 3-channel, 8-bit, or floating-point image. It is modified by the function unless the FLOODFILL_MASK_ONLY flag is set in the second variant of the function. See the details below. mask : Operation mask that should be a single-channel 8-bit image, 2 pixels wider and 2 pixels taller than image. Since this is both an input and output parameter, you must take responsibility of initializing it. Flood-filling cannot go across non-zero pixels in the input mask. For example, an edge detector output can be used as a mask to stop filling at edges. On output, pixels in the mask corresponding to filled pixels in the image are set to 1 or to the a value specified in flags as described below. Additionally, the function fills the border of the mask with ones to simplify internal processing. It is therefore possible to use the same mask in multiple calls to the function to make sure the filled areas do not overlap. seedPoint : Starting point. newVal : New value of the repainted domain pixels. loDiff : Maximal lower brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component. upDiff : Maximal upper brightness/color difference between the currently observed pixel and one of its	The function cv::floodFill fills a connected component starting from the seed point with the specified color. The connectivity is determined by the color/brightness closeness of the neighbor pixels. The pixel at (x,y) is considered to belong to the repainted domain if:in case of a grayscale image and floating range $$\texttt{src} (x',y')- \texttt{loDiff} \leq \texttt{src} (x,y) \leq \texttt{src} (x',y')+ \texttt{upDiff}$$ in case of a grayscale image and fixed range $$\texttt{src} (\texttt{seedPoint} .x, \texttt{seedPoint} .y)- \texttt{loDiff} \leq \texttt{src} (x,y) \leq \texttt{src} (\texttt{seedPoint} .x, \texttt{seedPoint} .y)+ \texttt{upDiff}$$ in case of a color image and floating range $$\texttt{src} (x',y')_r- \texttt{loDiff} _r \leq \texttt{src} (x,y)_r \leq \texttt{src} (x	7								
imgproc_floodFill2	CVAPI(ExceptionStatus) imgproc_floodFill2(cv::_InputOutputArray *image, cv::_InputOutputArray *mask, MyCvPoint seedPoint, MyCvScalar newVal, MyCvRect *rect, MyCvScalar loDiff, MyCvScalar upDiff, int flags, int* returnValue) { BEGIN_WRAP cv::Rect rect0; *returnValue = cv::floodFill(*image, *mask, cpp(seedPoint), cpp(newVal), &rect0, cpp(loDiff), cpp(upDiff), flags); *rect = c(rect0); END_WRAP }	cv::floodFill()	group__imgproc__misc.html	Fills a connected component with the given color.	image : Input/output 1- or 3-channel, 8-bit, or floating-point image. It is modified by the function unless the FLOODFILL_MASK_ONLY flag is set in the second variant of the function. See the details below. mask : Operation mask that should be a single-channel 8-bit image, 2 pixels wider and 2 pixels taller than image. Since this is both an input and output parameter, you must take responsibility of initializing it. Flood-filling cannot go across non-zero pixels in the input mask. For example, an edge detector output can be used as a mask to stop filling at edges. On output, pixels in the mask corresponding to filled pixels in the image are set to 1 or to the a value specified in flags as described below. Additionally, the function fills the border of the mask with ones to simplify internal processing. It is therefore possible to use the same mask in multiple calls to the function to make sure the filled areas do not overlap. seedPoint : Starting point. newVal : New value of the repainted domain pixels. loDiff : Maximal lower brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component. upDiff : Maximal upper brightness/color difference between the currently observed pixel and one of its	The function cv::floodFill fills a connected component starting from the seed point with the specified color. The connectivity is determined by the color/brightness closeness of the neighbor pixels. The pixel at (x,y) is considered to belong to the repainted domain if:in case of a grayscale image and floating range $$\texttt{src} (x',y')- \texttt{loDiff} \leq \texttt{src} (x,y) \leq \texttt{src} (x',y')+ \texttt{upDiff}$$ in case of a grayscale image and fixed range $$\texttt{src} (\texttt{seedPoint} .x, \texttt{seedPoint} .y)- \texttt{loDiff} \leq \texttt{src} (x,y) \leq \texttt{src} (\texttt{seedPoint} .x, \texttt{seedPoint} .y)+ \texttt{upDiff}$$ in case of a color image and floating range $$\texttt{src} (x',y')_r- \texttt{loDiff} _r \leq \texttt{src} (x,y)_r \leq \texttt{src} (x	8								
imgproc_blendLinear	CVAPI(ExceptionStatus) imgproc_blendLinear(cv::_InputArray* src1, cv::_InputArray* src2, cv::_InputArray* weights1, cv::_InputArray* weights2, cv::_OutputArray* dst) { BEGIN_WRAP cv::blendLinear(*src1, *src2, *weights1, *weights2, *dst); END_WRAP }	cv::blendLinear()	group__imgproc__misc.html	Performs linear blending of two images:	src1 : It has a type of CV_8UC(n) or CV_32FC(n), where n is a positive integer. src2 : It has the same type and size as src1. weights1 : It has a type of CV_32FC1 and the same size with src1. weights2 : It has a type of CV_32FC1 and the same size with src1. dst : It is created if it does not have the same size and type with src1.	$$\texttt{dst}(i,j) = \texttt{weights1}(i,j)*\texttt{src1}(i,j) + \texttt{weights2}(i,j)*\texttt{src2}(i,j)$$	5								
imgproc_cvtColor	CVAPI(ExceptionStatus) imgproc_cvtColor(cv::_InputArray *src, cv::_OutputArray *dst, int code, int dstCn) { BEGIN_WRAP cv::cvtColor(*src, *dst, code, dstCn); END_WRAP }	cv::cvtColor()	group__imgproc__color__conversions.html	Converts an image from one color space to another.	src : input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point. dst : output image of the same size and depth as src. code : color space conversion code (see ColorConversionCodes). dstCn : number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code.	The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.The conventional ranges for R, G, and B channel values are:0 to 255 for CV_8U images 0 to 65535 for CV_16U images 0 to 1 for CV_32F imagesIn case of linear transformations, the range does not matter. But in case of a non-linear transformation, an input RGB image should be normalized to the proper value range to get the correct results, for example, for RGB $\rightarrow$ L*u*v* transformation. For example, if you have a 32-bit floating-point image directly	4								
imgproc_cvtColorTwoPlane	CVAPI(ExceptionStatus) imgproc_cvtColorTwoPlane(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst, int code) { BEGIN_WRAP cv::cvtColorTwoPlane(*src1, *src2, *dst, code); END_WRAP }	cv::cvtColorTwoPlane()	group__imgproc__color__conversions.html	Converts an image from one color space to another where the source image is stored in two planes.	src1 : 8-bit image (CV_8U) of the Y plane. src2 : image containing interleaved U/V plane. dst : output image. code : Specifies the type of conversion. It can take any of the following values: COLOR_YUV2BGR_NV12 COLOR_YUV2RGB_NV12 COLOR_YUV2BGRA_NV12 COLOR_YUV2RGBA_NV12 COLOR_YUV2BGR_NV21 COLOR_YUV2RGB_NV21 COLOR_YUV2BGRA_NV21 COLOR_YUV2RGBA_NV21	This function only supports YUV420 to RGB conversion as of now.	4								
imgproc_demosaicing	CVAPI(ExceptionStatus) imgproc_demosaicing(cv::_InputArray *src, cv::_OutputArray *dst, int code, int dstCn) { BEGIN_WRAP cv::demosaicing(*src, *dst, code, dstCn); END_WRAP }	cv::demosaicing()	group__imgproc__color__conversions.html	main function for all demosaicing processes	src : input image: 8-bit unsigned or 16-bit unsigned. dst : output image of the same size and depth as src. code : Color space conversion code (see the description below). dstCn : number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code.	The function can do the following transformations:Demosaicing using bilinear interpolation COLOR_BayerBG2BGR , COLOR_BayerGB2BGR , COLOR_BayerRG2BGR , COLOR_BayerGR2BGR COLOR_BayerBG2GRAY , COLOR_BayerGB2GRAY , COLOR_BayerRG2GRAY , COLOR_BayerGR2GRAY Demosaicing using Variable Number of Gradients. COLOR_BayerBG2BGR_VNG , COLOR_BayerGB2BGR_VNG , COLOR_BayerRG2BGR_VNG , COLOR_BayerGR2BGR_VNG Edge-Aware Demosaicing. COLOR_BayerBG2BGR_EA , COLOR_BayerGB2BGR_EA , COLOR_BayerRG2BGR_EA , COLOR_BayerGR2BGR_EA Demosaicing with alpha channel COLOR_BayerBG2BGRA	4								
imgproc_moments	CVAPI(ExceptionStatus) imgproc_moments(cv::_InputArray *arr, int binaryImage, MyCvMoments *returnValue) { BEGIN_WRAP const auto m = cv::moments(*arr, binaryImage != 0); *returnValue = c(m); END_WRAP }	cv::moments()	group__imgproc__shape.html	Calculates all of the moments up to the third order of a polygon or rasterized shape.	array : Raster image (single-channel, 8-bit or floating-point 2D array) or an array ($1 \times N$ or $N \times 1$) of 2D points (Point or Point2f). binaryImage : If it is true, all non-zero image pixels are treated as 1's. The parameter is used for images only.	The function computes moments, up to the 3rd order, of a vector shape or a rasterized shape. The results are returned in the structure cv::Moments.NoteOnly applicable to contour moments calculations from Python bindings: Note that the numpy type for the input array should be either np.int32 or np.float32.See alsocontourArea, arcLength	2								

imgproc_matchTemplate	CVAPI(ExceptionStatus) imgproc_matchTemplate(cv::_InputArray *image, cv::_InputArray *templ, cv::_OutputArray *result, int method, cv::_InputArray *mask) { BEGIN_WRAP cv::matchTemplate(*image, *templ, *result, method, entity(mask)); END_WRAP }	cv::matchTemplate()	group__imgproc__object.html	Compares a template against overlapped image regions.	image : Image where the search is running. It must be 8-bit or 32-bit floating-point. templ : Searched template. It must be not greater than the source image and have the same data type. result : Map of comparison results. It must be single-channel 32-bit floating-point. If image is ¥(W ¥times H¥) and templ is ¥(w ¥times h¥) , then result is ¥((W-w+1) ¥times (H-h+1)¥) . method : Parameter specifying the comparison method, see TemplateMatchModes mask : Optional mask. It must have the same size as templ. It must either have the same number of channels as template or only one channel, which is then used for all template and image channels. If the data type is CV_8U, the mask is interpreted as a binary mask, meaning only elements where mask is nonzero are used and are kept unchanged independent of the actual mask value (weight equals 1). For data tpye CV_32F, the mask values are used as weights. The exact formulas are documented in TemplateMatchModes.	The function slides through image , compares the overlapped patches of size ¥(w ¥times h¥) against templ using the specified method and stores the comparison results in result . TemplateMatchModes describes the formulae for the available comparison methods (¥(I¥) denotes image, ¥(T¥) template, ¥(R¥) result, ¥(M¥) the optional mask). The summation is done over template and/or the image patch: ¥(x' = 0...w-1, y' = 0...h-1¥)After the function finishes the comparison, the best matches can be found as global minimums (when TM_SQDIFF was used) or maximums (when TM_CCORR or TM_CCOEFF was used) using the minMaxLoc function. In case of a color image, template summation in the numerator and each sum in the denominator is done over all of the channels and separate mean values are used for each channel. That is, the function can take a color template and a color image. The result will still be a single-channel image, which is easier to analyze.Examples:	5								
imgproc_connectedComponentsWithAlgorithm	CVAPI(ExceptionStatus) imgproc_connectedComponentsWithAlgorithm(cv::_InputArray *image, cv::_OutputArray *labels, int connectivity, int ltype, int cctype, int* returnValue) { BEGIN_WRAP *returnValue = cv::connectedComponents(entity(image), entity(labels), connectivity, ltype, cctype); END_WRAP }	cv::connectedComponents()	group__imgproc__shape.html	computes the connected components labeled image of boolean image	image : the 8-bit single-channel image to be labeled labels : destination labeled image connectivity : 8 or 4 for 8-way or 4-way connectivity respectively ltype : output image label type. Currently CV_32S and CV_16U are supported. cctype : connected components algorithm type (see the ConnectedComponentsAlgorithmsTypes).	image with 4 or 8 way connectivity - returns N, the total number of labels [0, N-1] where 0 represents the background label. ltype specifies the output label image type, an important consideration based on the total number of labels or alternatively the total number of pixels in the source image. cctype specifies the connected components labeling algorithm to use, currently Grana (BBDT) and Wu's (SAUF) [276] algorithms are supported, see the ConnectedComponentsAlgorithmsTypes for details. Note that SAUF algorithm forces a row major ordering of labels while BBDT does not. This function uses parallel version of both Grana and Wu's algorithms if at least one allowed parallel framework is enabled and if the rows of the image are at least twice the number returned by getNumberOfCPUs.Examples: samples/cpp/connectedcomponents.cpp.	5								
imgproc_connectedComponents	CVAPI(ExceptionStatus) imgproc_connectedComponentsWithAlgorithm(cv::_InputArray *image, cv::_OutputArray *labels, int connectivity, int ltype, int cctype, int* returnValue) { BEGIN_WRAP *returnValue = cv::connectedComponents(entity(image), entity(labels), connectivity, ltype, cctype); END_WRAP }	cv::connectedComponents()	group__imgproc__shape.html	computes the connected components labeled image of boolean image	image : the 8-bit single-channel image to be labeled labels : destination labeled image connectivity : 8 or 4 for 8-way or 4-way connectivity respectively ltype : output image label type. Currently CV_32S and CV_16U are supported. cctype : connected components algorithm type (see the ConnectedComponentsAlgorithmsTypes).	image with 4 or 8 way connectivity - returns N, the total number of labels [0, N-1] where 0 represents the background label. ltype specifies the output label image type, an important consideration based on the total number of labels or alternatively the total number of pixels in the source image. cctype specifies the connected components labeling algorithm to use, currently Grana (BBDT) and Wu's (SAUF) [276] algorithms are supported, see the ConnectedComponentsAlgorithmsTypes for details. Note that SAUF algorithm forces a row major ordering of labels while BBDT does not. This function uses parallel version of both Grana and Wu's algorithms if at least one allowed parallel framework is enabled and if the rows of the image are at least twice the number returned by getNumberOfCPUs.Examples: samples/cpp/connected_components.cpp.	5								
imgproc_connectedComponentsWithStatsWithA	CVAPI(ExceptionStatus) imgproc_connectedComponentsWithStatsWithAlgorithm(cv::_InputArray *image, cv::_OutputArray *labels, cv::_OutputArray *stats, cv::_OutputArray *centroids, int connectivity, int ltype, int cctype, int* returnValue) { BEGIN_WRAP *returnValue = cv::connectedComponentsWithStats(entity(image), entity(labels), entity(stats), entity(centroids), connectivity, ltype, cctype); END_WRAP }	cv::connectedComponentsWithStats()	group__imgproc__shape.html	computes the connected components labeled image of boolean image and also produces a statistics output for each label	image : the 8-bit single-channel image to be labeled labels : destination labeled image stats : statistics output for each label, including the background label. Statistics are accessed via stats(label, COLUMN) where COLUMN is one of ConnectedComponentsTypes, selecting the statistic. The data type is CV_32S. centroids : centroid output for each label, including the background label. Centroids are accessed via centroids(label, 0) for x and centroids(label, 1) for y. The data type CV_64F. connectivity : 8 or 4 for 8-way or 4-way connectivity respectively ltype : output image label type. Currently CV_32S and CV_16U are supported. cctype : connected components algorithm type (see ConnectedComponentsAlgorithmsTypes).	image with 4 or 8 way connectivity - returns N, the total number of labels [0, N-1] where 0 represents the background label. ltype specifies the output label image type, an important consideration based on the total number of labels or alternatively the total number of pixels in the source image. cctype specifies the connected components labeling algorithm to use, currently Grana's (BBDT) and Wu's (SAUF) [276] algorithms are supported, see the ConnectedComponentsAlgorithmsTypes for details. Note that SAUF algorithm forces a row major ordering of labels while BBDT does not. This function uses parallel version of both Grana and Wu's algorithms (statistics included) if at least one allowed parallel framework is enabled and if the rows of the image are at least twice the number returned by	7								
imgproc_connectedComponentsWithStats	CVAPI(ExceptionStatus) imgproc_connectedComponentsWithStatsWithAlgorithm(cv::_InputArray *image, cv::_OutputArray *labels, cv::_OutputArray *stats, cv::_OutputArray *centroids, int connectivity, int ltype, int cctype, int* returnValue) { BEGIN_WRAP *returnValue = cv::connectedComponentsWithStats(entity(image), entity(labels), entity(stats), entity(centroids), connectivity, ltype, cctype); END_WRAP }	cv::connectedComponentsWithStats()	group__imgproc__shape.html	computes the connected components labeled image of boolean image and also produces a statistics output for each label	image : the 8-bit single-channel image to be labeled labels : destination labeled image stats : statistics output for each label, including the background label. Statistics are accessed via stats(label, COLUMN) where COLUMN is one of ConnectedComponentsTypes, selecting the statistic. The data type is CV_32S. centroids : centroid output for each label, including the background label. Centroids are accessed via centroids(label, 0) for x and centroids(label, 1) for y. The data type CV_64F. connectivity : 8 or 4 for 8-way or 4-way connectivity respectively ltype : output image label type. Currently CV_32S and CV_16U are supported. cctype : connected components algorithm type (see ConnectedComponentsAlgorithmsTypes).	image with 4 or 8 way connectivity - returns N, the total number of labels [0, N-1] where 0 represents the background label. ltype specifies the output label image type, an important consideration based on the total number of labels or alternatively the total number of pixels in the source image. cctype specifies the connected components labeling algorithm to use, currently Grana's (BBDT) and Wu's (SAUF) [276] algorithms are supported, see the ConnectedComponentsAlgorithmsTypes for details. Note that SAUF algorithm forces a row major ordering of labels while BBDT does not. This function uses parallel version of both Grana and Wu's algorithms (statistics included) if at least one allowed parallel framework is enabled and if the rows of the image are at least twice the number returned by	7								
imgproc_findContours1_vector	CVAPI(ExceptionStatus) imgproc_findContours1_vector(cv::_InputArray *image, std::vector<std::vector<cv::Point> > *contours, std::vector<cv::Vec4i> *hierarchy, int mode, int method, MyCvPoint offset) { BEGIN_WRAP cv::findContours(*image, *contours, *hierarchy, mode, method, cpp(offset)); END_WRAP }	cv::findContours()	group__imgproc__shape.html	Finds contours in a binary image.	image : Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare, inRange, threshold , adaptiveThreshold, Canny, and others to create a binary image out of a grayscale or color one. If mode equals to RETR_CCOMP or RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1). contours : Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >). hierarchy : Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. mode : Contour retrieval mode, see RetrievalModes method : Contour approximation method, see ContourApproximationModes offset : Optional offset by which every contour point is	The function retrieves contours from the binary image using the algorithm [233] . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.NoteSince opencv 3.2 source image is not modified by this function.Examples: modules/shape/samples/shape_example.cpp, samples/cpp/contours2.cpp, samples/cpp/fitellipse.cpp, samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/watershed.cpp, and samples/tapi/squares.cpp.	6								
imgproc_findContours1_OutputArray	CVAPI(ExceptionStatus) imgproc_findContours1_OutputArray(cv::_InputArray *image, std::vector<cv::Mat> *contours, cv::_OutputArray *hierarchy, int mode, int method, MyCvPoint offset) { BEGIN_WRAP cv::findContours(*image, *contours, *hierarchy, mode, method, cpp(offset)); END_WRAP }	cv::findContours()	group__imgproc__shape.html	Finds contours in a binary image.	image : Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare, inRange, threshold , adaptiveThreshold, Canny, and others to create a binary image out of a grayscale or color one. If mode equals to RETR_CCOMP or RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1). contours : Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >). hierarchy : Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. mode : Contour retrieval mode, see RetrievalModes method : Contour approximation method, see ContourApproximationModes offset : Optional offset by which every contour point is	The function retrieves contours from the binary image using the algorithm [233] . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.NoteSince opencv 3.2 source image is not modified by this function.Examples: modules/shape/samples/shape_example.cpp, samples/cpp/contours2.cpp, samples/cpp/fitellipse.cpp, samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/watershed.cpp, and samples/tapi/squares.cpp.	6								
imgproc_findContours2_vector	CVAPI(ExceptionStatus) imgproc_findContours2_vector(cv::_InputArray *image, std::vector<std::vector<cv::Point> > *contours, int mode, int method, MyCvPoint offset) { BEGIN_WRAP cv::findContours(*image, *contours, mode, method, cpp(offset)); END_WRAP }	cv::findContours()	group__imgproc__shape.html	Finds contours in a binary image.	image : Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare, inRange, threshold , adaptiveThreshold, Canny, and others to create a binary image out of a grayscale or color one. If mode equals to RETR_CCOMP or RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1). contours : Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >). hierarchy : Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. mode : Contour retrieval mode, see RetrievalModes method : Contour approximation method, see ContourApproximationModes offset : Optional offset by which every contour point is	The function retrieves contours from the binary image using the algorithm [233] . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.NoteSince opencv 3.2 source image is not modified by this function.Examples: modules/shape/samples/shape_example.cpp, samples/cpp/contours2.cpp, samples/cpp/fitellipse.cpp, samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/watershed.cpp, and samples/tapi/squares.cpp.	5								

imgproc_findContours2_OutputArray	CVAPI(ExceptionStatus) imgproc_findContours2_OutputArray(cv::_InputArray *image, std::vector<cv::Mat> *contours, int mode, int method, MyCvPoint offset) { BEGIN_WRAP cv::findContours(*image, *contours, mode, method, cpp(offset)); END_WRAP }	cv::findContours()	group__imgproc__shape.html	Finds contours in a binary image.	image : Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare, inRange, threshold , adaptiveThreshold, Canny, and others to create a binary image out of a grayscale or color one. If mode equals to RETR_CCOMP or RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1). contours : Detected contours. Each contour is stored as a vector of points (e.g. std::vector<std::vector<cv::Point> >). hierarchy : Optional output vector (e.g. std::vector<cv::Vec4i>), containing information about the image topology. It has as many elements as the number of contours. For each i-th contour contours[i], the elements hierarchy[i][0] , hierarchy[i][1] , hierarchy[i][2] , and hierarchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative. mode : Contour retrieval mode, see RetrievalModes method : Contour approximation method, see ContourApproximationModes offset : Optional offset by which every contour point is	The function retrieves contours from the binary image using the algorithm [233] . The contours are a useful tool for shape analysis and object detection and recognition. See squares.cpp in the OpenCV sample directory.NoteSince opencv 3.2 source image is not modified by this function.Examples: modules/shape/samples/shape_example.cpp, samples/cpp/contours2.cpp, samples/cpp/ttellipse.cpp, samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, samples/cpp/watershed.cpp, and samples/tapi/squares.cpp.	5								
imgproc_approxPolyDP_InputArray	CVAPI(ExceptionStatus) imgproc_approxPolyDP_InputArray(cv::_InputArray *curve, cv::_OutputArray *approxCurve, double epsilon, int closed) { BEGIN_WRAP cv::approxPolyDP(*curve, *approxCurve, epsilon, closed != 0); END_WRAP }	cv::approxPolyDP()	group__imgproc__shape.html	Approximates a polygonal curve(s) with the specified precision.	curve : Input vector of a 2D point stored in std::vector or Mat approxCurve : Result of the approximation. The type should match the type of the input curve. epsilon : Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation. closed : If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.	The function cv::approxPolyDP approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the Douglas-Peucker algorithm http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithmExamples: samples/cpp/contours2.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	4								
imgproc_approxPolyDP_Point	CVAPI(ExceptionStatus) imgproc_approxPolyDP_Point(cv::Point *curve, int curveLength, std::vector<cv::Point> *approxCurve, double epsilon, int closed) { BEGIN_WRAP const cv::Mat_<cv::Point> curveMat(curveLength, 1, curve); cv::approxPolyDP(curveMat, *approxCurve, epsilon, closed != 0); END_WRAP }	cv::approxPolyDP()	group__imgproc__shape.html	Approximates a polygonal curve(s) with the specified precision.	curve : Input vector of a 2D point stored in std::vector or Mat approxCurve : Result of the approximation. The type should match the type of the input curve. epsilon : Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation. closed : If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.	The function cv::approxPolyDP approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the Douglas-Peucker algorithm http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithmExamples: samples/cpp/contours2.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	4								
imgproc_approxPolyDP_Point2f	CVAPI(ExceptionStatus) imgproc_approxPolyDP_Point2f(cv::Point2f *curve, int curveLength, std::vector<cv::Point2f> *approxCurve, double epsilon, int closed) { BEGIN_WRAP const cv::Mat_<cv::Point2f> curveMat(curveLength, 1, curve); cv::approxPolyDP(curveMat, *approxCurve, epsilon, closed != 0); END_WRAP }	cv::approxPolyDP()	group__imgproc__shape.html	Approximates a polygonal curve(s) with the specified precision.	curve : Input vector of a 2D point stored in std::vector or Mat approxCurve : Result of the approximation. The type should match the type of the input curve. epsilon : Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation. closed : If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.	The function cv::approxPolyDP approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. It uses the Douglas-Peucker algorithm http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithmExamples: samples/cpp/contours2.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	4								
imgproc_arcLength_InputArray	CVAPI(ExceptionStatus) imgproc_arcLength_InputArray(cv::_InputArray *curve, int closed, double *returnValue) { BEGIN_WRAP *returnValue = cv::arcLength(*curve, closed != 0); END_WRAP }	cv::arcLength()	group__imgproc__shape.html	Calculates a contour perimeter or a curve length.	curve : Input vector of 2D points, stored in std::vector or Mat. closed : Flag indicating whether the curve is closed or not.	The function computes a curve length or a closed contour perimeter.Examples: samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	2								
imgproc_arcLength_Point	CVAPI(ExceptionStatus) imgproc_arcLength_Point(cv::Point *curve, int curveLength, int closed, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> curveMat(curveLength, 1, curve); *returnValue = cv::arcLength(curveMat, closed != 0); END_WRAP }	cv::arcLength()	group__imgproc__shape.html	Calculates a contour perimeter or a curve length.	curve : Input vector of 2D points, stored in std::vector or Mat. closed : Flag indicating whether the curve is closed or not.	The function computes a curve length or a closed contour perimeter.Examples: samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	2								
imgproc_arcLength_Point2f	CVAPI(ExceptionStatus) imgproc_arcLength_Point2f(cv::Point2f *curve, int curveLength, int closed, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> curveMat(curveLength, 1, curve); *returnValue = cv::arcLength(curveMat, closed != 0); END_WRAP }	cv::arcLength()	group__imgproc__shape.html	Calculates a contour perimeter or a curve length.	curve : Input vector of 2D points, stored in std::vector or Mat. closed : Flag indicating whether the curve is closed or not.	The function computes a curve length or a closed contour perimeter.Examples: samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	2								
imgproc_boundingRect_InputArray	CVAPI(ExceptionStatus) imgproc_boundingRect_InputArray(cv::_InputArray *curve, MyCvRect* returnValue) { BEGIN_WRAP *returnValue = c(cv::boundingRect(*curve)); END_WRAP }	cv::boundingRect()	group__imgproc__shape.html	Calculates the up-right bounding rectangle of a point set or non-zero pixels of gray-scale image.	array : Input gray-scale image or 2D point set, stored in std::vector or Mat.	The function calculates and returns the minimal up-right bounding rectangle for the specified point set or non-zero pixels of gray-scale image.	1								
imgproc_boundingRect_Point	CVAPI(ExceptionStatus) imgproc_boundingRect_Point(cv::Point *curve, int curveLength, MyCvRect* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> curveMat(curveLength, 1, curve); *returnValue = c(cv::boundingRect(curveMat)); END_WRAP }	cv::boundingRect()	group__imgproc__shape.html	Calculates the up-right bounding rectangle of a point set or non-zero pixels of gray-scale image.	array : Input gray-scale image or 2D point set, stored in std::vector or Mat.	The function calculates and returns the minimal up-right bounding rectangle for the specified point set or non-zero pixels of gray-scale image.	1								
imgproc_boundingRect_Point2f	CVAPI(ExceptionStatus) imgproc_boundingRect_Point2f(cv::Point2f *curve, int curveLength, MyCvRect* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> curveMat(curveLength, 1, curve); *returnValue = c(cv::boundingRect(curveMat)); END_WRAP }	cv::boundingRect()	group__imgproc__shape.html	Calculates the up-right bounding rectangle of a point set or non-zero pixels of gray-scale image.	array : Input gray-scale image or 2D point set, stored in std::vector or Mat.	The function calculates and returns the minimal up-right bounding rectangle for the specified point set or non-zero pixels of gray-scale image.	1								
imgproc_contourArea_InputArray	CVAPI(ExceptionStatus) imgproc_contourArea_InputArray(cv::_InputArray *contour, int oriented, double* returnValue) { BEGIN_WRAP *returnValue = cv::contourArea(*contour, oriented != 0); END_WRAP }	cv::contourArea()	group__imgproc__shape.html	Calculates a contour area.	contour : Input vector of 2D points (contour vertices), stored in std::vector or Mat. oriented : Oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.	The function computes a contour area. Similarly to moments , the area is computed using the Green formula. Thus, the returned area and the number of non-zero pixels, if you draw the contour using drawContours or fillPoly , can be different. Also, the function will most certainly give a wrong results for contours with self-intersections.Example: vector<Point> contour;contour.push_back(Point2f(0, 0));contour.push_back(Point2f(10, 0));contour.push_back(Point2f(10, 10));contour.push_back(Point2f(5, 4));double area0 = contourArea(contour);vector<Point> approx;approxPolyDP(contour, approx, 5, true);double area1 = contourArea(approx);cout << "area0 =" << area0 << endl << "area1 =" << area1 << endl << "approx poly vertices" << approx.size() << endl;fragmentExamples: samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, and	2								

imgproc_contourArea_Point	CVAPI(ExceptionStatus) imgproc_contourArea_Point(cv::Point *contour, int contourLength, int oriented, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> contourMat(contourLength, 1, contour); *returnValue = cv::contourArea(contourMat, oriented != 0); END_WRAP }	cv::contourArea()	group__imgproc__shape.html	Calculates a contour area.	contour : Input vector of 2D points (contour vertices), stored in std::vector or Mat. oriented : Oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.	The function computes a contour area. Similarly to moments , the area is computed using the Green formula. Thus, the returned area and the number of non-zero pixels, if you draw the contour using drawContours or fillPoly , can be different. Also, the function will most certainly give a wrong results for contours with self-intersections.Example:vector<Point> contour;contour.push_back(Point2f(0, 0));contour.push_back(Point2f(10, 0));contour.push_back(Point2f(10, 10));contour.push_back(Point2f(5, 4));double area0 = contourArea(contour);vector<Point> approx;approxPolyDP(contour, approx, 5, true);double area1 = contourArea(approx);cout << "area0 =" << area0 << endl << "area1 =" << area1 << endl << "approx poly vertices" << approx.size() << endl;fragmentExamples: samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, and	2								
imgproc_contourArea_Point2f	CVAPI(ExceptionStatus) imgproc_contourArea_Point2f(cv::Point2f *contour, int contourLength, int oriented, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> contourMat(contourLength, 1, contour); *returnValue = cv::contourArea(contourMat, oriented != 0); END_WRAP }	cv::contourArea()	group__imgproc__shape.html	Calculates a contour area.	contour : Input vector of 2D points (contour vertices), stored in std::vector or Mat. oriented : Oriented area flag. If it is true, the function returns a signed area value, depending on the contour orientation (clockwise or counter-clockwise). Using this feature you can determine orientation of a contour by taking the sign of an area. By default, the parameter is false, which means that the absolute value is returned.	The function computes a contour area. Similarly to moments , the area is computed using the Green formula. Thus, the returned area and the number of non-zero pixels, if you draw the contour using drawContours or fillPoly , can be different. Also, the function will most certainly give a wrong results for contours with self-intersections.Example:vector<Point> contour;contour.push_back(Point2f(0, 0));contour.push_back(Point2f(10, 0));contour.push_back(Point2f(10, 10));contour.push_back(Point2f(5, 4));double area0 = contourArea(contour);vector<Point> approx;approxPolyDP(contour, approx, 5, true);double area1 = contourArea(approx);cout << "area0 =" << area0 << endl << "area1 =" << area1 << endl << "approx poly vertices" << approx.size() << endl;fragmentExamples: samples/cpp/segment_objects.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ml/introduction_to_pca/introduction_to_pca.cpp, and	2								
imgproc_minAreaRect_InputArray	CVAPI(ExceptionStatus) imgproc_minAreaRect_InputArray(cv::_InputArray *points, MyCvBox2D* returnValue) { BEGIN_WRAP *returnValue = c(cv::minAreaRect(*points)); END_WRAP }	cv::minAreaRect()	group__imgproc__shape.html	Finds a rotated rectangle of the minimum area enclosing the input 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat	The function calculates and returns the minimum-area bounding rectangle (possibly rotated) for a specified point set. Developer should keep in mind that the returned RotatedRect can contain negative indices when data is close to the containing Mat element boundary.Examples: samples/cpp/minarea.cpp.	1								
imgproc_minAreaRect_Point	CVAPI(ExceptionStatus) imgproc_minAreaRect_Point(cv::Point *points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsMat(pointsLength, 1, points); *returnValue = c(cv::minAreaRect(pointsMat)); END_WRAP }	cv::minAreaRect()	group__imgproc__shape.html	Finds a rotated rectangle of the minimum area enclosing the input 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat	The function calculates and returns the minimum-area bounding rectangle (possibly rotated) for a specified point set. Developer should keep in mind that the returned RotatedRect can contain negative indices when data is close to the containing Mat element boundary.Examples: samples/cpp/minarea.cpp.	1								
imgproc_minAreaRect_Point2f	CVAPI(ExceptionStatus) imgproc_minAreaRect_Point2f(cv::Point2f *points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsMat(pointsLength, 1, points); *returnValue = c(cv::minAreaRect(pointsMat)); END_WRAP }	cv::minAreaRect()	group__imgproc__shape.html	Finds a rotated rectangle of the minimum area enclosing the input 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat	The function calculates and returns the minimum-area bounding rectangle (possibly rotated) for a specified point set. Developer should keep in mind that the returned RotatedRect can contain negative indices when data is close to the containing Mat element boundary.Examples: samples/cpp/minarea.cpp.	1								
imgproc_boxPoints_OutputArray	CVAPI(ExceptionStatus) imgproc_boxPoints_OutputArray(MyCvBox2D box, cv::_OutputArray* points) { BEGIN_WRAP cv::boxPoints(cpp(box), *points); END_WRAP }	cv::boxPoints()	group__imgproc__shape.html	Finds the four vertices of a rotated rect. Useful to draw the rotated rectangle.	box : The input rotated rectangle. It may be the output of points : The output array of four vertices of rectangles.	The function finds the four vertices of a rotated rectangle. This function is useful to draw the rectangle. In C++, instead of using this function, you can directly use RotatedRect::points method. Please visit the tutorial on Creating Bounding rotated boxes and ellipses for contours for more information.	2								
imgproc_boxPoints_Point2f	CVAPI(ExceptionStatus) imgproc_boxPoints_Point2f(MyCvBox2D box, cv::Point2f points[4]) { BEGIN_WRAP cpp(box).points(points); END_WRAP }	cv::boxPoints()	group__imgproc__shape.html	Finds the four vertices of a rotated rect. Useful to draw the rotated rectangle.	box : The input rotated rectangle. It may be the output of points : The output array of four vertices of rectangles.	The function finds the four vertices of a rotated rectangle. This function is useful to draw the rectangle. In C++, instead of using this function, you can directly use RotatedRect::points method. Please visit the tutorial on Creating Bounding rotated boxes and ellipses for contours for more information.	-1								
imgproc_minEnclosingCircle_InputArray	CVAPI(ExceptionStatus) imgproc_minEnclosingCircle_InputArray(cv::_InputArray *points, MyCvPoint2D32f *center, float *radius) { BEGIN_WRAP cv::Point2f center0; float radius0; cv::minEnclosingCircle(*points, center0, radius0); *center = c(center0); *radius = radius0; END_WRAP }	cv::minEnclosingCircle()	group__imgproc__shape.html	Finds a circle of the minimum area enclosing a 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat center : Output center of the circle. radius : Output radius of the circle.	The function finds the minimal enclosing circle of a 2D point set using an iterative algorithm.Examples: samples/cpp/minarea.cpp.	3								
imgproc_minEnclosingCircle_Point	CVAPI(ExceptionStatus) imgproc_minEnclosingCircle_Point(cv::Point *points, int pointsLength, MyCvPoint2D32f*center, float *radius) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsMat(pointsLength, 1, points); cv::Point2f center0; float radius0; cv::minEnclosingCircle(pointsMat, center0, radius0); *center = c(center0); *radius = radius0; END_WRAP }	cv::minEnclosingCircle()	group__imgproc__shape.html	Finds a circle of the minimum area enclosing a 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat center : Output center of the circle. radius : Output radius of the circle.	The function finds the minimal enclosing circle of a 2D point set using an iterative algorithm.Examples: samples/cpp/minarea.cpp.	3								
imgproc_minEnclosingCircle_Point2f	CVAPI(ExceptionStatus) imgproc_minEnclosingCircle_Point2f(cv::Point2f *points, int pointsLength, MyCvPoint2D32f*center, float *radius) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsMat(pointsLength, 1, points); cv::Point2f center0; float radius0; cv::minEnclosingCircle(pointsMat, center0, radius0); *center = c(center0); *radius = radius0; END_WRAP }	cv::minEnclosingCircle()	group__imgproc__shape.html	Finds a circle of the minimum area enclosing a 2D point set.	points : Input vector of 2D points, stored in std::vector<> or Mat center : Output center of the circle. radius : Output radius of the circle.	The function finds the minimal enclosing circle of a 2D point set using an iterative algorithm.Examples: samples/cpp/minarea.cpp.	3								
imgproc_minEnclosingTriangle_InputOutputArray	CVAPI(ExceptionStatus) imgproc_minEnclosingTriangle_InputOutputArray(cv::_InputArray *points, cv::_OutputArray *triangle, double *returnValue) { BEGIN_WRAP *returnValue = cv::minEnclosingTriangle(*points, *triangle); END_WRAP }	cv::minEnclosingTriangle()	group__imgproc__shape.html	Finds a triangle of minimum area enclosing a 2D point set and returns its area.	points : Input vector of 2D points with depth CV_32S or CV_32F, stored in std::vector<> or Mat triangle : Output vector of three 2D points defining the vertices of the triangle. The depth of the OutputArray must be CV_32F.	The function finds a triangle of minimum area enclosing the given set of 2D points and returns its area. The output for a given 2D point set is shown in the image below. 2D points are depicted in red* and the enclosing triangle in yellow.Sample output of the minimum enclosing triangle functionThe implementation of the algorithm is based on O'Rourke's [188] and Klee and Laskowski's [129] papers. O'Rourke provides a ¥(¥theta(n)¥) algorithm for finding the minimal enclosing triangle of a 2D convex polygon with n vertices. Since the minEnclosingTriangle function takes a 2D point set as input an additional preprocessing step of computing the convex hull of the 2D point set is required. The complexity of the convexHull function is ¥(O(n log(n))¥) which is higher than ¥(¥theta(n)¥). Thus the overall complexity of the function is ¥(O(n log(n))¥).Examples: samples/cpp/minarea.cpp.	2								

imgproc_minEnclosingTriangle_Point	CVAPI(ExceptionStatus) imgproc_minEnclosingTriangle_Point(cv::Point* points, int pointsLength, std::vector<cv::Point2f>* triangle, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsMat(pointsLength, 1, points); *returnValue = cv::minEnclosingTriangle(pointsMat, *triangle); END_WRAP }	cv::minEnclosingTriangle()	group__imgproc__shape.html	Finds a triangle of minimum area enclosing a 2D point set and returns its area.	points : Input vector of 2D points with depth CV_32S or CV_32F, stored in std::vector<> or Mat triangle : Output vector of three 2D points defining the vertices of the triangle. The depth of the OutputArray must be CV_32F.	The function finds a triangle of minimum area enclosing the given set of 2D points and returns its area. The output for a given 2D point set is shown in the image below. 2D points are depicted in red* and the enclosing triangle in yellow.Sample output of the minimum enclosing triangle functionThe implementation of the algorithm is based on O'Rourke's [188] and Klee and Laskowski's [129] papers. O'Rourke provides a ¥(¥theta(n)¥) algorithm for finding the minimal enclosing triangle of a 2D convex polygon with n vertices. Since the minEnclosingTriangle function takes a 2D point set as input an additional preprocessing step of computing the convex hull of the 2D point set is required. The complexity of the convexHull function is ¥(O(n log(n))¥) which is higher than ¥(¥theta(n)¥). Thus the overall complexity of the function is ¥(O(n log(n))¥).Examples: samples/cpp/minarea.cpp.	2								
imgproc_minEnclosingTriangle_Point2f	CVAPI(ExceptionStatus) imgproc_minEnclosingTriangle_Point2f(cv::Point2f* points, int pointsLength, std::vector<cv::Point2f>* triangle, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsMat(pointsLength, 1, points); *returnValue = cv::minEnclosingTriangle(pointsMat, *triangle); END_WRAP }	cv::minEnclosingTriangle()	group__imgproc__shape.html	Finds a triangle of minimum area enclosing a 2D point set and returns its area.	points : Input vector of 2D points with depth CV_32S or CV_32F, stored in std::vector<> or Mat triangle : Output vector of three 2D points defining the vertices of the triangle. The depth of the OutputArray must be CV_32F.	The function finds a triangle of minimum area enclosing the given set of 2D points and returns its area. The output for a given 2D point set is shown in the image below. 2D points are depicted in red* and the enclosing triangle in yellow.Sample output of the minimum enclosing triangle functionThe implementation of the algorithm is based on O'Rourke's [188] and Klee and Laskowski's [129] papers. O'Rourke provides a ¥(¥theta(n)¥) algorithm for finding the minimal enclosing triangle of a 2D convex polygon with n vertices. Since the minEnclosingTriangle function takes a 2D point set as input an additional preprocessing step of computing the convex hull of the 2D point set is required. The complexity of the convexHull function is ¥(O(n log(n))¥) which is higher than ¥(¥theta(n)¥). Thus the overall complexity of the function is ¥(O(n log(n))¥).Examples: samples/cpp/minarea.cpp.	2								
imgproc_matchShapes_InputArray	CVAPI(ExceptionStatus) imgproc_matchShapes_InputArray(cv::_InputArray *contour1, cv::_InputArray *contour2, int method, double parameter, double* returnValue) { BEGIN_WRAP *returnValue = cv::matchShapes(*contour1, *contour2, method, parameter); END_WRAP }	cv::matchShapes()	group__imgproc__shape.html	Compares two shapes.	contour1 : First contour or grayscale image. contour2 : Second contour or grayscale image. method : Comparison method, see ShapeMatchModes parameter : Method-specific parameter (not supported now).	The function compares two shapes. All three implemented methods use the Hu invariants (see HuMoments)	4								
imgproc_matchShapes_Point	CVAPI(ExceptionStatus) imgproc_matchShapes_Point(cv::Point *contour1, int contour1Length, cv::Point *contour2, int contour2Length, int method, double parameter, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> contour1Mat(contour1Length, 1, contour1); const cv::Mat_<cv::Point> contour2Mat(contour2Length, 1, contour2); *returnValue = cv::matchShapes(contour1Mat, contour2Mat, method, parameter); END_WRAP }	cv::matchShapes()	group__imgproc__shape.html	Compares two shapes.	contour1 : First contour or grayscale image. contour2 : Second contour or grayscale image. method : Comparison method, see ShapeMatchModes parameter : Method-specific parameter (not supported now).	The function compares two shapes. All three implemented methods use the Hu invariants (see HuMoments)	4								
imgproc_convexHull_InputArray	CVAPI(ExceptionStatus) imgproc_convexHull_InputArray(cv::_InputArray *points, cv::_OutputArray *hull, int clockwise, int returnPoints) { BEGIN_WRAP cv::convexHull(*points, *hull, clockwise != 0, returnPoints != 0); END_WRAP }	cv::convexHull()	group__imgproc__shape.html	Finds the convex hull of a point set.	points : Input 2D point set, stored in std::vector or Mat. hull : Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves. clockwise : Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. returnPoints : Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.	The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [224] that has O(N logN) complexity in the current implementation.Notepoints and hull should be different arrays, inplace processing isn't supported.Check the corresponding tutorial for more details.useful links:https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/Examples: samples/cpp/convexhull.cpp.	4								
imgproc_convexHull_Point_ReturnsPoints	CVAPI(ExceptionStatus) imgproc_convexHull_Point_ReturnsPoints(cv::Point *points, int pointsLength, std::vector<cv::Point> *hull, int clockwise) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsMat(pointsLength, 1, points); cv::convexHull(pointsMat, *hull, clockwise != 0, true); END_WRAP }	cv::convexHull()	group__imgproc__shape.html	Finds the convex hull of a point set.	points : Input 2D point set, stored in std::vector or Mat. hull : Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves. clockwise : Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. returnPoints : Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.	The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [224] that has O(N logN) complexity in the current implementation.Notepoints and hull should be different arrays, inplace processing isn't supported.Check the corresponding tutorial for more details.useful links:https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/Examples: samples/cpp/convexhull.cpp.	4								
imgproc_convexHull_Point2f_ReturnsPoints	CVAPI(ExceptionStatus) imgproc_convexHull_Point2f_ReturnsPoints(cv::Point2f *points, int pointsLength, std::vector<cv::Point2f> *hull, int clockwise) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsMat(pointsLength, 1, points); cv::convexHull(pointsMat, *hull, clockwise != 0, true); END_WRAP }	cv::convexHull()	group__imgproc__shape.html	Finds the convex hull of a point set.	points : Input 2D point set, stored in std::vector or Mat. hull : Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves. clockwise : Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. returnPoints : Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.	The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [224] that has O(N logN) complexity in the current implementation.Notepoints and hull should be different arrays, inplace processing isn't supported.Check the corresponding tutorial for more details.useful links:https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/Examples: samples/cpp/convexhull.cpp.	4								
imgproc_convexHull_Point_ReturnsIndices	CVAPI(ExceptionStatus) imgproc_convexHull_Point_ReturnsIndices(cv::Point *points, int pointsLength, std::vector<int> *hull, int clockwise) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsMat(pointsLength, 1, points); cv::convexHull(pointsMat, *hull, clockwise != 0, false); END_WRAP }	cv::convexHull()	group__imgproc__shape.html	Finds the convex hull of a point set.	points : Input 2D point set, stored in std::vector or Mat. hull : Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves. clockwise : Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. returnPoints : Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.	The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [224] that has O(N logN) complexity in the current implementation.Notepoints and hull should be different arrays, inplace processing isn't supported.Check the corresponding tutorial for more details.useful links:https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/Examples: samples/cpp/convexhull.cpp.	4								
imgproc_convexHull_Point2f_ReturnsIndices	CVAPI(ExceptionStatus) imgproc_convexHull_Point2f_ReturnsIndices(cv::Point2f *points, int pointsLength, std::vector<int> *hull, int clockwise) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsMat(pointsLength, 1, points); cv::convexHull(pointsMat, *hull, clockwise != 0, false); END_WRAP }	cv::convexHull()	group__imgproc__shape.html	Finds the convex hull of a point set.	points : Input 2D point set, stored in std::vector or Mat. hull : Output convex hull. It is either an integer vector of indices or vector of points. In the first case, the hull elements are 0-based indices of the convex hull points in the original array (since the set of convex hull points is a subset of the original point set). In the second case, hull elements are the convex hull points themselves. clockwise : Orientation flag. If it is true, the output convex hull is oriented clockwise. Otherwise, it is oriented counter-clockwise. The assumed coordinate system has its X axis pointing to the right, and its Y axis pointing upwards. returnPoints : Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. When the output array is std::vector, the flag is ignored, and the output depends on the type of the vector: std::vector<int> implies returnPoints=false, std::vector<Point> implies returnPoints=true.	The function cv::convexHull finds the convex hull of a 2D point set using the Sklansky's algorithm [224] that has O(N logN) complexity in the current implementation.Notepoints and hull should be different arrays, inplace processing isn't supported.Check the corresponding tutorial for more details.useful links:https://www.learnopencv.com/convex-hull-using-opencv-in-python-and-c/Examples: samples/cpp/convexhull.cpp.	4								

imgproc_convexityDefects_InputArray	CVAPI(ExceptionStatus) imgproc_convexityDefects_InputArray(cv::_InputArray *contour, cv::_InputArray *convexHull, cv::_OutputArray *convexityDefects) { BEGIN_WRAP cv::convexityDefects(*contour, *convexHull, *convexityDefects); END_WRAP }	cv::convexityDefects()	group__imgproc__shape.html	Finds the convexity defects of a contour.	contour : Input contour. convexhull : Convex hull obtained using convexHull that should contain indices of the contour points that make the hull. convexityDefects : The output vector of convexity defects. In C++ and the new Python/Java interface each convexity defect is represented as 4-element integer vector (a.k.a. Vec4i): (start_index, end_index, farthest_pt_index, fixpt_depth), where indices are 0-based indices in the original contour of the convexity defect beginning, end and the farthest point, and fixpt_depth is fixed-point approximation (with 8 fractional bits) of the distance between the farthest contour point and the hull. That is, to get the floating-point value of the depth will be fixpt_depth/256.0.	The figure below displays convexity defects of a hand contour:image	3								
imgproc_convexityDefects_Point	CVAPI(ExceptionStatus) imgproc_convexityDefects_Point(cv::Point *contour, int contourLength, int *convexHull, int convexHullLength, std::vector<cv::Vec4i> *convexityDefects) { BEGIN_WRAP const cv::Mat_<cv::Point> contourMat(contourLength, 1, contour); const cv::Mat_<int> convexHullMat(convexHullLength, 1, convexHull); cv::convexityDefects(contourMat, convexHullMat, *convexityDefects); END_WRAP }	cv::convexityDefects()	group__imgproc__shape.html	Finds the convexity defects of a contour.	contour : Input contour. convexhull : Convex hull obtained using convexHull that should contain indices of the contour points that make the hull. convexityDefects : The output vector of convexity defects. In C++ and the new Python/Java interface each convexity defect is represented as 4-element integer vector (a.k.a. Vec4i): (start_index, end_index, farthest_pt_index, fixpt_depth), where indices are 0-based indices in the original contour of the convexity defect beginning, end and the farthest point, and fixpt_depth is fixed-point approximation (with 8 fractional bits) of the distance between the farthest contour point and the hull. That is, to get the floating-point value of the depth will be fixpt_depth/256.0.	The figure below displays convexity defects of a hand contour:image	3								
imgproc_convexityDefects_Point2f	CVAPI(ExceptionStatus) imgproc_convexityDefects_Point2f(cv::Point2f *contour, int contourLength, int *convexHull, int convexHullLength, std::vector<cv::Vec4i> *convexityDefects) { BEGIN_WRAP const cv::Mat_<cv::Point2f> contourMat(contourLength, 1, contour); const cv::Mat_<int> convexHullMat(convexHullLength, 1, convexHull); cv::convexityDefects(contourMat, convexHullMat, *convexityDefects); END_WRAP }	cv::convexityDefects()	group__imgproc__shape.html	Finds the convexity defects of a contour.	contour : Input contour. convexhull : Convex hull obtained using convexHull that should contain indices of the contour points that make the hull. convexityDefects : The output vector of convexity defects. In C++ and the new Python/Java interface each convexity defect is represented as 4-element integer vector (a.k.a. Vec4i): (start_index, end_index, farthest_pt_index, fixpt_depth), where indices are 0-based indices in the original contour of the convexity defect beginning, end and the farthest point, and fixpt_depth is fixed-point approximation (with 8 fractional bits) of the distance between the farthest contour point and the hull. That is, to get the floating-point value of the depth will be fixpt_depth/256.0.	The figure below displays convexity defects of a hand contour:image	3								
imgproc_isContourConvex_InputArray	CVAPI(ExceptionStatus) imgproc_isContourConvex_InputArray(cv::_InputArray *contour, int* returnValue) { BEGIN_WRAP *returnValue = cv::isContourConvex(*contour) ? 1 : 0; END_WRAP }	cv::isContourConvex()	group__imgproc__shape.html	Tests a contour convexity.	contour : Input vector of 2D points, stored in std::vector<> or Mat	The function tests whether the input contour is convex or not. The contour must be simple, that is, without self-intersections. Otherwise, the function output is undefined.Examples: samples/cpp/intersectExample.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	1								
imgproc_isContourConvex_Point	CVAPI(ExceptionStatus) imgproc_isContourConvex_Point(cv::Point *contour, int contourLength, int* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> contourMat(contourLength, 1, contour); *returnValue = cv::isContourConvex(contourMat) ? 1 : 0; END_WRAP }	cv::isContourConvex()	group__imgproc__shape.html	Tests a contour convexity.	contour : Input vector of 2D points, stored in std::vector<> or Mat	The function tests whether the input contour is convex or not. The contour must be simple, that is, without self-intersections. Otherwise, the function output is undefined.Examples: samples/cpp/intersectExample.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	1								
imgproc_isContourConvex_Point2f	CVAPI(ExceptionStatus) imgproc_isContourConvex_Point2f(cv::Point2f *contour, int contourLength, int* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> contourMat(contourLength, 1, contour); *returnValue = cv::isContourConvex(contourMat) ? 1 : 0; END_WRAP }	cv::isContourConvex()	group__imgproc__shape.html	Tests a contour convexity.	contour : Input vector of 2D points, stored in std::vector<> or Mat	The function tests whether the input contour is convex or not. The contour must be simple, that is, without self-intersections. Otherwise, the function output is undefined.Examples: samples/cpp/intersectExample.cpp, samples/cpp/squares.cpp, and samples/tapi/squares.cpp.	1								
imgproc_intersectConvexConvex_InputArray	CVAPI(ExceptionStatus) imgproc_intersectConvexConvex_InputArray(cv::_InputArray *p1, cv::_InputArray *p2, cv::_OutputArray *p12, int handleNested, float* returnValue) { BEGIN_WRAP *returnValue = cv::intersectConvexConvex(*p1, *p2, *p12, handleNested != 0); END_WRAP }	cv::intersectConvexConvex()	group__imgproc__shape.html	Finds intersection of two convex polygons.	p1 : First polygon p2 : Second polygon p12 : Output polygon describing the intersecting area handleNested : When true, an intersection is found if one of the polygons is fully enclosed in the other. When false, no intersection is found. If the polygons share a side or the vertex of one polygon lies on an edge of the other, they are not considered nested and an intersection will be found regardless of the value of handleNested.	NoteintersectConvexConvex doesn't confirm that both polygons are convex and will return invalid results if they aren't.Examples: samples/cpp/intersectExample.cpp.	4								
imgproc_intersectConvexConvex_Point	CVAPI(ExceptionStatus) imgproc_intersectConvexConvex_Point(cv::Point *p1, int p1Length, cv::Point *p2, int p2Length, std::vector<cv::Point> *p12, int handleNested, float* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> p1Vec(p1Length, 1, p1); const cv::Mat_<cv::Point> p2Vec(p2Length, 1, p2); *returnValue = cv::intersectConvexConvex(p1Vec, p2Vec, *p12, handleNested != 0); END_WRAP }	cv::intersectConvexConvex()	group__imgproc__shape.html	Finds intersection of two convex polygons.	p1 : First polygon p2 : Second polygon p12 : Output polygon describing the intersecting area handleNested : When true, an intersection is found if one of the polygons is fully enclosed in the other. When false, no intersection is found. If the polygons share a side or the vertex of one polygon lies on an edge of the other, they are not considered nested and an intersection will be found regardless of the value of handleNested.	NoteintersectConvexConvex doesn't confirm that both polygons are convex and will return invalid results if they aren't.Examples: samples/cpp/intersectExample.cpp.	4								
imgproc_intersectConvexConvex_Point2f	CVAPI(ExceptionStatus) imgproc_intersectConvexConvex_Point2f(cv::Point2f *p1, int p1Length, cv::Point2f *p2, int p2Length, std::vector<cv::Point2f> *p12, int handleNested, float *returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> p1Vec(p1Length, 1, p1); const cv::Mat_<cv::Point2f> p2Vec(p2Length, 1, p2); *returnValue = cv::intersectConvexConvex(p1Vec, p2Vec, *p12, handleNested != 0); END_WRAP }	cv::intersectConvexConvex()	group__imgproc__shape.html	Finds intersection of two convex polygons.	p1 : First polygon p2 : Second polygon p12 : Output polygon describing the intersecting area handleNested : When true, an intersection is found if one of the polygons is fully enclosed in the other. When false, no intersection is found. If the polygons share a side or the vertex of one polygon lies on an edge of the other, they are not considered nested and an intersection will be found regardless of the value of handleNested.	NoteintersectConvexConvex doesn't confirm that both polygons are convex and will return invalid results if they aren't.Examples: samples/cpp/intersectExample.cpp.	4								
imgproc_fitEllipse_InputArray	CVAPI(ExceptionStatus) imgproc_fitEllipse_InputArray(cv::_InputArray *points, MyCvBox2D* returnValue) { BEGIN_WRAP *returnValue = c(cv::fitEllipse(*points)); END_WRAP }	cv::fitEllipse()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits (in a least-squares sense) a set of 2D points best of all. It returns the rotated rectangle in which the ellipse is inscribed. The first algorithm described by [78] is used. Developer should keep in mind that it is possible that the returned ellipse/rotatedRect data contains negative indices, due to the data points being close to the border of the containing Mat element.Examples:	1								
imgproc_fitEllipse_Point	CVAPI(ExceptionStatus) imgproc_fitEllipse_Point(cv::Point *points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipse(pointsVec)); END_WRAP }	cv::fitEllipse()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits (in a least-squares sense) a set of 2D points best of all. It returns the rotated rectangle in which the ellipse is inscribed. The first algorithm described by [78] is used. Developer should keep in mind that it is possible that the returned ellipse/rotatedRect data contains negative indices, due to the data points being close to the border of the containing Mat element.Examples: samples/cpp/fitellipse.cpp.	1								

imgproc_fitEllipse_Point2f	CVAPI(ExceptionStatus) imgproc_fitEllipse_Point2f(cv::Point2f *points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipse(pointsVec)); END_WRAP }	cv::fitEllipse()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits (in a least-squares sense) a set of 2D points best of all. It returns the rotated rectangle in which the ellipse is inscribed. The first algorithm described by [78] is used. Developer should keep in mind that it is possible that the returned ellipse/rotatedRect data contains negative indices, due to the data points being close to the border of the containing Mat element.Examples: samples/cpp/fitellipse.cpp.	1								
imgproc_fitEllipseAMS_InputArray																
imgproc_fitEllipseAMS_Point	CVAPI(ExceptionStatus) imgproc_fitEllipseAMS_Point(cv::Point* points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipseAMS(pointsVec)); END_WRAP }	cv::fitEllipseAMS()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits a set of 2D points. It returns the rotated rectangle in which the ellipse is inscribed. The Approximate Mean Square (AMS) proposed by [240] is used.For an ellipse, this basis set is ¥(¥chi= ¥left(x^2, x y, y^2, x, y, 1¥right) ¥), which is a set of six free coefficients ¥(A^T=¥left¥{A_{¥text{xx}},A_{¥text{xy}},A_{¥text{yy}},A_x,A_y,A_0¥right¥} ¥). However, to specify an ellipse, all that is needed is five numbers; the major and minor axes lengths ¥((a,b) ¥), the position ¥((x_0,y_0) ¥), and the orientation ¥(¥theta ¥). This is because the basis set includes lines, quadratics, parabolic and hyperbolic functions as well as elliptical functions as possible fits. If the fit is found to be a parabolic or hyperbolic function then the standard fitEllipse method is used. The AMS method restricts the fit to parabolic, hyperbolic and elliptical curves by imposing the condition that ¥(A^T (D_x^T D_x + D_y^T D_y) A = 1 ¥) where the matrices ¥(Dx ¥) and ¥(Dy ¥)	1								
imgproc_fitEllipseAMS_Point2f	CVAPI(ExceptionStatus) imgproc_fitEllipseAMS_Point2f(cv::Point2f* points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipseAMS(pointsVec)); END_WRAP }	cv::fitEllipseAMS()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits a set of 2D points. It returns the rotated rectangle in which the ellipse is inscribed. The Approximate Mean Square (AMS) proposed by [240] is used.For an ellipse, this basis set is ¥(¥chi= ¥left(x^2, x y, y^2, x, y, 1¥right) ¥), which is a set of six free coefficients ¥(A^T=¥left¥{A_{¥text{xx}},A_{¥text{xy}},A_{¥text{yy}},A_x,A_y,A_0¥right¥} ¥). However, to specify an ellipse, all that is needed is five numbers; the major and minor axes lengths ¥((a,b) ¥), the position ¥((x_0,y_0) ¥), and the orientation ¥(¥theta ¥). This is because the basis set includes lines, quadratics, parabolic and hyperbolic functions as well as elliptical functions as possible fits. If the fit is found to be a parabolic or hyperbolic function then the standard fitEllipse method is used. The AMS method restricts the fit to parabolic, hyperbolic and elliptical curves by imposing the condition that ¥(A^T (D_x^T D_x + D_y^T D_y) A = 1 ¥) where the matrices ¥(Dx ¥) and ¥(Dy ¥)	1								
imgproc_fitEllipseDirect_InputArray																
imgproc_fitEllipseDirect_Point	CVAPI(ExceptionStatus) imgproc_fitEllipseDirect_Point(cv::Point* points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipseDirect(pointsVec)); END_WRAP }	cv::fitEllipseDirect()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits a set of 2D points. It returns the rotated rectangle in which the ellipse is inscribed. The Direct least square (Direct) method by [79] is used.For an ellipse, this basis set is ¥(¥chi= ¥left(x^2, x y, y^2, x, y, 1¥right) ¥), which is a set of six free coefficients ¥(A^T=¥left¥{A_{¥text{xx}},A_{¥text{xy}},A_{¥text{yy}},A_x,A_y,A_0¥right¥} ¥). However, to specify an ellipse, all that is needed is five numbers; the major and minor axes lengths ¥((a,b) ¥), the position ¥((x_0,y_0) ¥), and the orientation ¥(¥theta ¥). This is because the basis set includes lines, quadratics, parabolic and hyperbolic functions as well as elliptical functions as possible fits. The Direct method confines the fit to ellipses by ensuring that ¥(4 A_{xx} A_{yy}- A_{xy}^2 > 0 ¥). The condition imposed is that ¥(4 A_{xx} A_{yy}- A_{xy}^2=1 ¥) which satisfies the inequality and as the coefficients can be arbitrarily scaled is not overly restrictive.¥begin{equation*} ¥epsilon ^2= A^T D^T D A ¥quad	1								
imgproc_fitEllipseDirect_Point2f	CVAPI(ExceptionStatus) imgproc_fitEllipseDirect_Point2f(cv::Point2f* points, int pointsLength, MyCvBox2D* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsVec(pointsLength, 1, points); *returnValue = c(cv::fitEllipseDirect(pointsVec)); END_WRAP }	cv::fitEllipseDirect()	group__imgproc__shape.html	Fits an ellipse around a set of 2D points.	points : Input 2D point set, stored in std::vector<> or Mat	The function calculates the ellipse that fits a set of 2D points. It returns the rotated rectangle in which the ellipse is inscribed. The Direct least square (Direct) method by [79] is used.For an ellipse, this basis set is ¥(¥chi= ¥left(x^2, x y, y^2, x, y, 1¥right) ¥), which is a set of six free coefficients ¥(A^T=¥left¥{A_{¥text{xx}},A_{¥text{xy}},A_{¥text{yy}},A_x,A_y,A_0¥right¥} ¥). However, to specify an ellipse, all that is needed is five numbers; the major and minor axes lengths ¥((a,b) ¥), the position ¥((x_0,y_0) ¥), and the orientation ¥(¥theta ¥). This is because the basis set includes lines, quadratics, parabolic and hyperbolic functions as well as elliptical functions as possible fits. The Direct method confines the fit to ellipses by ensuring that ¥(4 A_{xx} A_{yy}- A_{xy}^2 > 0 ¥). The condition imposed is that ¥(4 A_{xx} A_{yy}- A_{xy}^2=1 ¥) which satisfies the inequality and as the coefficients can be arbitrarily scaled is not overly restrictive.¥begin{equation*} ¥epsilon ^2= A^T D^T D A ¥quad	1								
imgproc_fitLine_InputArray	CVAPI(ExceptionStatus) imgproc_fitLine_InputArray(cv::_InputArray *points, cv::_OutputArray *line, int distType, double param, double reps, double aeps) { BEGIN_WRAP cv::fitLine(*points, *line, distType, param, reps, aeps); END_WRAP }	cv::fitLine()	group__imgproc__shape.html	Fits a line to a 2D or 3D point set.	points : Input vector of 2D or 3D points, stored in std::vector<> or Mat. line : Output line parameters. In case of 2D fitting, it should be a vector of 4 elements (like Vec4f) - (vx, vy, x0, y0), where (vx, vy) is a normalized vector collinear to the line and (x0, y0) is a point on the line. In case of 3D fitting, it should be a vector of 6 elements (like Vec6f) - (vx, vy, vz, x0, y0, z0), where (vx, vy, vz) is a normalized vector collinear to the line and (x0, y0, z0) is a point on the line. distType : Distance used by the M-estimator, see DistanceTypes param : Numerical parameter (C) for some types of distances. If it is 0, an optimal value is chosen. reps : Sufficient accuracy for the radius (distance between the coordinate origin and the line). aeps : Sufficient accuracy for the angle. 0.01 would be a good default value for reps and aeps.	The function fitLine fits a line to a 2D or 3D point set by minimizing ¥(¥sum_i ¥rho(r_i)¥) where ¥(r_i¥) is a distance between the ¥(i^{th}¥) point, the line and ¥(¥rho(r)¥) is a distance function, one of the following:DIST_L2 ¥(¥rho (r) = r^2/2 ¥quad ¥text{(the simplest and the fastest least-squares method)}¥] DIST_L1 ¥(¥rho (r) = r¥] DIST_L12 ¥(¥rho (r) = 2 ¥cdot (¥sqrt{1 + ¥frac{r^2}{2}} - 1)¥] DIST_FAIR ¥(¥rho ¥left (r ¥right) = C^2 ¥cdot ¥left (¥frac{r}{C} - ¥log{¥left(1 + ¥frac{r}{C}¥right)} ¥right) ¥quad ¥text{where} ¥quad C=1.3998¥]	6								
imgproc_fitLine_Point	CVAPI(ExceptionStatus) imgproc_fitLine_Point(cv::Point *points, int pointsLength, float *line, int distType, double param, double reps, double aeps) { BEGIN_WRAP const cv::Mat_<cv::Point> pointsVec(pointsLength, 1, points); cv::Mat_<float> lineVec(4, 1, line); cv::fitLine(pointsVec, lineVec, distType, param, reps, aeps); END_WRAP }	cv::fitLine()	group__imgproc__shape.html	Fits a line to a 2D or 3D point set.	points : Input vector of 2D or 3D points, stored in std::vector<> or Mat. line : Output line parameters. In case of 2D fitting, it should be a vector of 4 elements (like Vec4f) - (vx, vy, x0, y0), where (vx, vy) is a normalized vector collinear to the line and (x0, y0) is a point on the line. In case of 3D fitting, it should be a vector of 6 elements (like Vec6f) - (vx, vy, vz, x0, y0, z0), where (vx, vy, vz) is a normalized vector collinear to the line and (x0, y0, z0) is a point on the line. distType : Distance used by the M-estimator, see DistanceTypes param : Numerical parameter (C) for some types of distances. If it is 0, an optimal value is chosen. reps : Sufficient accuracy for the radius (distance between the coordinate origin and the line). aeps : Sufficient accuracy for the angle. 0.01 would be a good default value for reps and aeps.	The function fitLine fits a line to a 2D or 3D point set by minimizing ¥(¥sum_i ¥rho(r_i)¥) where ¥(r_i¥) is a distance between the ¥(i^{th}¥) point, the line and ¥(¥rho(r)¥) is a distance function, one of the following:DIST_L2 ¥(¥rho (r) = r^2/2 ¥quad ¥text{(the simplest and the fastest least-squares method)}¥] DIST_L1 ¥(¥rho (r) = r¥] DIST_L12 ¥(¥rho (r) = 2 ¥cdot (¥sqrt{1 + ¥frac{r^2}{2}} - 1)¥] DIST_FAIR ¥(¥rho ¥left (r ¥right) = C^2 ¥cdot ¥left (¥frac{r}{C} - ¥log{¥left(1 + ¥frac{r}{C}¥right)} ¥right) ¥quad ¥text{where} ¥quad C=1.3998¥]	6								
imgproc_fitLine_Point2f	CVAPI(ExceptionStatus) imgproc_fitLine_Point2f(cv::Point2f *points, int pointsLength, float *line, int distType, double param, double reps, double aeps) { BEGIN_WRAP const cv::Mat_<cv::Point2f> pointsVec(pointsLength, 1, points); cv::Mat_<float> lineVec(4, 1, line); cv::fitLine(pointsVec, lineVec, distType, param, reps, aeps); END_WRAP }	cv::fitLine()	group__imgproc__shape.html	Fits a line to a 2D or 3D point set.	points : Input vector of 2D or 3D points, stored in std::vector<> or Mat. line : Output line parameters. In case of 2D fitting, it should be a vector of 4 elements (like Vec4f) - (vx, vy, x0, y0), where (vx, vy) is a normalized vector collinear to the line and (x0, y0) is a point on the line. In case of 3D fitting, it should be a vector of 6 elements (like Vec6f) - (vx, vy, vz, x0, y0, z0), where (vx, vy, vz) is a normalized vector collinear to the line and (x0, y0, z0) is a point on the line. distType : Distance used by the M-estimator, see DistanceTypes param : Numerical parameter (C) for some types of distances. If it is 0, an optimal value is chosen. reps : Sufficient accuracy for the radius (distance between the coordinate origin and the line). aeps : Sufficient accuracy for the angle. 0.01 would be a good default value for reps and aeps.	The function fitLine fits a line to a 2D or 3D point set by minimizing ¥(¥sum_i ¥rho(r_i)¥) where ¥(r_i¥) is a distance between the ¥(i^{th}¥) point, the line and ¥(¥rho(r)¥) is a distance function, one of the following:DIST_L2 ¥(¥rho (r) = r^2/2 ¥quad ¥text{(the simplest and the fastest least-squares method)}¥] DIST_L1 ¥(¥rho (r) = r¥] DIST_L12 ¥(¥rho (r) = 2 ¥cdot (¥sqrt{1 + ¥frac{r^2}{2}} - 1)¥] DIST_FAIR ¥(¥rho ¥left (r ¥right) = C^2 ¥cdot ¥left (¥frac{r}{C} - ¥log{¥left(1 + ¥frac{r}{C}¥right)} ¥right) ¥quad ¥text{where} ¥quad C=1.3998¥]	6								

imgproc_fitLine_Point3i	CVAPI(ExceptionStatus) imgproc_fitLine_Point3i(cv::Point3i *points, int pointsLength, float *line, int distType, double param, double reps, double aeps) { BEGIN_WRAP const cv::Mat_<cv::Point3i> pointsVec(pointsLength, 1, points); cv::Mat_<float> lineVec(6, 1, line); cv::fitLine(pointsVec, lineVec, distType, param, reps, aeps); END_WRAP }	cv::fitLine()	group__imgproc__shape.html	Fits a line to a 2D or 3D point set.	points : Input vector of 2D or 3D points, stored in std::vector<> or Mat. line : Output line parameters. In case of 2D fitting, it should be a vector of 4 elements (like Vec4f) - (vx, vy, x0, y0), where (vx, vy) is a normalized vector collinear to the line and (x0, y0) is a point on the line. In case of 3D fitting, it should be a vector of 6 elements (like Vec6f) - (vx, vy, vz, x0, y0, z0), where (vx, vy, vz) is a normalized vector collinear to the line and (x0, y0, z0) is a point on the line. distType : Distance used by the M-estimator, see DistanceTypes param : Numerical parameter (C) for some types of distances. If it is 0, an optimal value is chosen. reps : Sufficient accuracy for the radius (distance between the coordinate origin and the line). aeps : Sufficient accuracy for the angle. 0.01 would be a good default value for reps and aeps.	The function fitLine fits a line to a 2D or 3D point set by minimizing ¥(¥sum_i ¥rho(r_i)¥) where ¥(r_i¥) is a distance between the ¥(i^{th}¥) point, the line and ¥(¥rho(r)¥) is a distance function, one of the following:DIST_L2 ¥[¥rho (r) = r^2/2 ¥quad ¥text{(the simplest and the fastest least-squares method)}¥] DIST_L1 ¥[¥rho (r) = r¥] DIST_L12 ¥[¥rho (r) = 2 ¥cdot (¥sqrt{1 + ¥frac{r^2}{2}} - 1)¥] DIST_FAIR ¥[¥rho ¥left (r ¥right) = C^2 ¥cdot ¥left (¥frac{r}{C} - ¥log(¥left(1 + ¥frac{r}{C}¥right)¥right) ¥quad ¥text{where} ¥quad C=1.3998¥]	6								
imgproc_fitLine_Point3f	CVAPI(ExceptionStatus) imgproc_fitLine_Point3f(cv::Point3f *points, int pointsLength, float *line, int distType, double param, double reps, double aeps) { BEGIN_WRAP const cv::Mat_<cv::Point3f> pointsVec(pointsLength, 1, points); cv::Mat_<float> lineVec(6, 1, line); cv::fitLine(pointsVec, lineVec, distType, param, reps, aeps); END_WRAP }	cv::fitLine()	group__imgproc__shape.html	Fits a line to a 2D or 3D point set.	points : Input vector of 2D or 3D points, stored in std::vector<> or Mat. line : Output line parameters. In case of 2D fitting, it should be a vector of 4 elements (like Vec4f) - (vx, vy, x0, y0), where (vx, vy) is a normalized vector collinear to the line and (x0, y0) is a point on the line. In case of 3D fitting, it should be a vector of 6 elements (like Vec6f) - (vx, vy, vz, x0, y0, z0), where (vx, vy, vz) is a normalized vector collinear to the line and (x0, y0, z0) is a point on the line. distType : Distance used by the M-estimator, see DistanceTypes param : Numerical parameter (C) for some types of distances. If it is 0, an optimal value is chosen. reps : Sufficient accuracy for the radius (distance between the coordinate origin and the line). aeps : Sufficient accuracy for the angle. 0.01 would be a good default value for reps and aeps.	The function fitLine fits a line to a 2D or 3D point set by minimizing ¥(¥sum_i ¥rho(r_i)¥) where ¥(r_i¥) is a distance between the ¥(i^{th}¥) point, the line and ¥(¥rho(r)¥) is a distance function, one of the following:DIST_L2 ¥[¥rho (r) = r^2/2 ¥quad ¥text{(the simplest and the fastest least-squares method)}¥] DIST_L1 ¥[¥rho (r) = r¥] DIST_L12 ¥[¥rho (r) = 2 ¥cdot (¥sqrt{1 + ¥frac{r^2}{2}} - 1)¥] DIST_FAIR ¥[¥rho ¥left (r ¥right) = C^2 ¥cdot ¥left (¥frac{r}{C} - ¥log(¥left(1 + ¥frac{r}{C}¥right)¥right) ¥quad ¥text{where} ¥quad C=1.3998¥]	6								
imgproc_pointPolygonTest_InputArray	CVAPI(ExceptionStatus) imgproc_pointPolygonTest_InputArray(cv::_InputArray* contour, MyCvPoint2D32f pt, int measureDist, double *returnValue) { BEGIN_WRAP *returnValue = cv::pointPolygonTest(*contour, cpp(pt), measureDist != 0); END_WRAP }	cv::pointPolygonTest()	group__imgproc__shape.html	Performs a point-in-contour test.	contour : Input contour. pt : Point tested against the contour. measureDist : If true, the function estimates the signed distance from the point to the nearest contour edge. Otherwise, the function only checks if the point is inside a contour or not.	The function determines whether the point is inside a contour, outside, or lies on an edge (or coincides with a vertex). It returns positive (inside), negative (outside), or zero (on an edge) value, correspondingly. When measureDist=false , the return value is +1, -1, and 0, respectively. Otherwise, the return value is a signed distance between the point and the nearest contour edge.See below a sample output of the function where each image pixel is tested against the contour:sample output	3								
imgproc_pointPolygonTest_Point	CVAPI(ExceptionStatus) imgproc_pointPolygonTest_Point(cv::Point *contour, int contourLength, MyCvPoint2D32f pt, int measureDist, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point> contourVec(contourLength, 1, contour); *returnValue = cv::pointPolygonTest(contourVec, cpp(pt), measureDist != 0); END_WRAP }	cv::pointPolygonTest()	group__imgproc__shape.html	Performs a point-in-contour test.	contour : Input contour. pt : Point tested against the contour. measureDist : If true, the function estimates the signed distance from the point to the nearest contour edge. Otherwise, the function only checks if the point is inside a contour or not.	The function determines whether the point is inside a contour, outside, or lies on an edge (or coincides with a vertex). It returns positive (inside), negative (outside), or zero (on an edge) value, correspondingly. When measureDist=false , the return value is +1, -1, and 0, respectively. Otherwise, the return value is a signed distance between the point and the nearest contour edge.See below a sample output of the function where each image pixel is tested against the contour:sample output	3								
imgproc_pointPolygonTest_Point2f	CVAPI(ExceptionStatus) imgproc_pointPolygonTest_Point2f(cv::Point2f *contour, int contourLength, MyCvPoint2D32f pt, int measureDist, double* returnValue) { BEGIN_WRAP const cv::Mat_<cv::Point2f> contourVec(contourLength, 1, contour); *returnValue = cv::pointPolygonTest(contourVec, cpp(pt), measureDist != 0); END_WRAP }	cv::pointPolygonTest()	group__imgproc__shape.html	Performs a point-in-contour test.	contour : Input contour. pt : Point tested against the contour. measureDist : If true, the function estimates the signed distance from the point to the nearest contour edge. Otherwise, the function only checks if the point is inside a contour or not.	The function determines whether the point is inside a contour, outside, or lies on an edge (or coincides with a vertex). It returns positive (inside), negative (outside), or zero (on an edge) value, correspondingly. When measureDist=false , the return value is +1, -1, and 0, respectively. Otherwise, the return value is a signed distance between the point and the nearest contour edge.See below a sample output of the function where each image pixel is tested against the contour:sample output	3								
imgproc_rotatedRectangleIntersection_Output	CVAPI(ExceptionStatus) imgproc_rotatedRectangleIntersection_OutputArray(MyCvBox2D rect1, MyCvBox2D rect2, cv::_OutputArray *intersectingRegion, int* returnValue) { BEGIN_WRAP *returnValue = cv::rotatedRectangleIntersection(cpp(rect1), cpp(rect2), *intersectingRegion); END_WRAP }	cv::rotatedRectangleIntersection()	group__imgproc__shape.html	Finds out if there is any intersection between two rotated rectangles.	rect1 : First rectangle rect2 : Second rectangle intersectingRegion : The output array of the vertices of the intersecting region. It returns at most 8 vertices. Stored as std::vector<cv::Point2f> or cv::Mat as Mx1 of type CV_32FC2.	If there is then the vertices of the intersecting region are returned as well.Below are some examples of intersection configurations. The hatched pattern indicates the intersecting region and the red vertices are returned by the function.intersection examples	3								
imgproc_rotatedRectangleIntersection_vector	CVAPI(ExceptionStatus) imgproc_rotatedRectangleIntersection_vector(MyCvBox2D rect1, MyCvBox2D rect2, std::vector<cv::Point2f> *intersectingRegion, int* returnValue) { BEGIN_WRAP *returnValue = cv::rotatedRectangleIntersection(cpp(rect1), cpp(rect2), *intersectingRegion); END_WRAP }	cv::rotatedRectangleIntersection()	group__imgproc__shape.html	Finds out if there is any intersection between two rotated rectangles.	rect1 : First rectangle rect2 : Second rectangle intersectingRegion : The output array of the vertices of the intersecting region. It returns at most 8 vertices. Stored as std::vector<cv::Point2f> or cv::Mat as Mx1 of type CV_32FC2.	If there is then the vertices of the intersecting region are returned as well.Below are some examples of intersection configurations. The hatched pattern indicates the intersecting region and the red vertices are returned by the function.intersection examples	3								
imgproc_applyColorMap1	CVAPI(ExceptionStatus) imgproc_applyColorMap1(cv::_InputArray *src, cv::_OutputArray *dst, int colorMap) { BEGIN_WRAP cv::applyColorMap(*src, *dst, colorMap); END_WRAP }	cv::applyColorMap()	group__imgproc__colormap.html	Applies a GNU Octave/MATLAB equivalent colormap on a given image.	src : The source image, grayscale or colored of type CV_8UC1 or CV_8UC3. dst : The result is the colormapped source image. Note: Mat::create is called on dst. colormap : The colormap to apply, see ColormapTypes	Examples: samples/cpp/falsecolor.cpp.	3								
imgproc_applyColorMap2	CVAPI(ExceptionStatus) imgproc_applyColorMap2(cv::_InputArray *src, cv::_OutputArray *dst, cv::_InputArray *userColor) { BEGIN_WRAP cv::applyColorMap(*src, *dst, *userColor); END_WRAP }	cv::applyColorMap()	group__imgproc__colormap.html	Applies a GNU Octave/MATLAB equivalent colormap on a given image.	src : The source image, grayscale or colored of type CV_8UC1 or CV_8UC3. dst : The result is the colormapped source image. Note: Mat::create is called on dst. colormap : The colormap to apply, see ColormapTypes	Examples: samples/cpp/falsecolor.cpp.	3								
imgproc_line	CVAPI(ExceptionStatus) imgproc_linearPolar(cv::_InputArray *src, cv::_OutputArray *dst, MyCvPoint2D32f center, double maxRadius, int flags) { BEGIN_WRAP cv::linearPolar(*src, *dst, cpp(center), maxRadius, flags); END_WRAP }	cv::linearPolar()	group__imgproc__transform.html	Remaps an image to polar coordinates space.		Deprecated:This function produces same result as cv::warpPolar(src, dst, src.size(), center, maxRadius, flags)Examples: samples/cpp/polar_transforms.cpp.	5								
imgproc_arrowedLine	CVAPI(ExceptionStatus) imgproc_arrowedLine(cv::_InputOutputArray *img, MyCvPoint pt1, MyCvPoint pt2, MyCvScalar color, int thickness, int line_type, int shift, double tipLength) { BEGIN_WRAP cv::arrowedLine(*img, cpp(pt1), cpp(pt2), cpp(color), thickness, line_type, shift, tipLength); END_WRAP }	cv::arrowedLine()	group__imgproc__draw.html	Draws a arrow segment pointing from the first point to the second one.	img : Image. pt1 : The point the arrow starts from. pt2 : The point the arrow points to. color : Line color. thickness : Line thickness. line_type : Type of the line. See LineTypes shift : Number of fractional bits in the point coordinates. tipLength : The length of the arrow tip in relation to the arrow length	The function cv::arrowedLine draws an arrow between pt1 and pt2 points in the image. See also line.	8								

imgproc_rectangle_InputOutputArray_Point	CVAPI(ExceptionStatus) imgproc_rectangle_InputOutputArray_Point(cv::_InputOutputArray *img, MyCvPoint pt1, MyCvPoint pt2, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::rectangle(*img, cpp(pt1), cpp(pt2), cpp(color), thickness, lineType, shift); END_WRAP }	cv::rectangle()	group__imgproc__draw.html	Draws a simple, thick, or filled up-right rectangle.	img : Image. pt1 : Vertex of the rectangle. pt2 : Vertex of the rectangle opposite to pt1 . color : Rectangle color or brightness (grayscale image). thickness : Thickness of lines that make up the rectangle. Negative values, like FILLED, mean that the function has to draw a filled rectangle. lineType : Type of the line. See LineTypes shift : Number of fractional bits in the point coordinates.	The function cv::rectangle draws a rectangle outline or a filled rectangle whose two opposite corners are pt1 and pt2.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/demhist.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/grabcut.cpp, samples/cpp/intersectExample.cpp, samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/dnn/object_detection.cpp, and	7								
imgproc_rectangle_InputOutputArray_Rect	CVAPI(ExceptionStatus) imgproc_rectangle_InputOutputArray_Rect(cv::_InputOutputArray* img, MyCvRect rect, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::rectangle(*img, cpp(rect), cpp(color), thickness, lineType, shift); END_WRAP }	cv::rectangle()	group__imgproc__draw.html	Draws a simple, thick, or filled up-right rectangle.	img : Image. pt1 : Vertex of the rectangle. pt2 : Vertex of the rectangle opposite to pt1 . color : Rectangle color or brightness (grayscale image). thickness : Thickness of lines that make up the rectangle. Negative values, like FILLED, mean that the function has to draw a filled rectangle. lineType : Type of the line. See LineTypes shift : Number of fractional bits in the point coordinates.	The function cv::rectangle draws a rectangle outline or a filled rectangle whose two opposite corners are pt1 and pt2.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/demhist.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/grabcut.cpp, samples/cpp/intersectExample.cpp, samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/dnn/object_detection.cpp, and	6								
imgproc_rectangle_Mat_Point	CVAPI(ExceptionStatus) imgproc_rectangle_Mat_Point(cv::Mat* img, MyCvPoint pt1, MyCvPoint pt2, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::rectangle(*img, cpp(pt1), cpp(pt2), cpp(color), thickness, lineType, shift); END_WRAP }	cv::rectangle()	group__imgproc__draw.html	Draws a simple, thick, or filled up-right rectangle.	img : Image. pt1 : Vertex of the rectangle. pt2 : Vertex of the rectangle opposite to pt1 . color : Rectangle color or brightness (grayscale image). thickness : Thickness of lines that make up the rectangle. Negative values, like FILLED, mean that the function has to draw a filled rectangle. lineType : Type of the line. See LineTypes shift : Number of fractional bits in the point coordinates.	The function cv::rectangle draws a rectangle outline or a filled rectangle whose two opposite corners are pt1 and pt2.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/demhist.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/grabcut.cpp, samples/cpp/intersectExample.cpp, samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/dnn/object_detection.cpp, and	7								
imgproc_rectangle_Mat_Rect	CVAPI(ExceptionStatus) imgproc_rectangle_Mat_Rect(cv::Mat *img, MyCvRect rect, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::rectangle(*img, cpp(rect), cpp(color), thickness, lineType, shift); END_WRAP }	cv::rectangle()	group__imgproc__draw.html	Draws a simple, thick, or filled up-right rectangle.	img : Image. pt1 : Vertex of the rectangle. pt2 : Vertex of the rectangle opposite to pt1 . color : Rectangle color or brightness (grayscale image). thickness : Thickness of lines that make up the rectangle. Negative values, like FILLED, mean that the function has to draw a filled rectangle. lineType : Type of the line. See LineTypes shift : Number of fractional bits in the point coordinates.	The function cv::rectangle draws a rectangle outline or a filled rectangle whose two opposite corners are pt1 and pt2.Examples: samples/cpp/camshiftdemo.cpp, samples/cpp/demhist.cpp, samples/cpp/facedetect.cpp, samples/cpp/falsecolor.cpp, samples/cpp/grabcut.cpp, samples/cpp/intersectExample.cpp, samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, samples/cpp/tutorial_code/Histograms_Matching/MatchTemplate_Demo.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/dnn/object_detection.cpp, and	6								
imgproc_circle	CVAPI(ExceptionStatus) imgproc_circle(cv::_InputOutputArray *img, MyCvPoint center, int radius, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::circle(*img, cpp(center), radius, cpp(color), thickness, lineType, shift); END_WRAP }	cv::circle()	group__imgproc__draw.html	Draws a circle.	img : Image where the circle is drawn. center : Center of the circle. radius : Radius of the circle. color : Circle color. thickness : Thickness of the circle outline, if positive. Negative values, like FILLED, mean that a filled circle is to be drawn. lineType : Type of the circle boundary. See LineTypes shift : Number of fractional bits in the coordinates of the center and in the radius value.	The function cv::circle draws a simple or filled circle with a given center and radius.Examples: samples/cpp/convexhull.cpp, samples/cpp/falsecolor.cpp, samples/cpp/kmeans.cpp, samples/cpp/lkdemo.cpp, samples/cpp/minarea.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/cpp/tutorial_code/ImgTrans/houghcircles.cpp, and samples/dnn/openpose.cpp.	7								
imgproc_ellipse1	CVAPI(ExceptionStatus) imgproc_ellipse1(cv::_InputOutputArray *img, MyCvPoint center, MyCvSize axes, double angle, double startAngle, double endAngle, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::ellipse(*img, cpp(center), cpp(axes), angle, startAngle, endAngle, cpp(color), thickness, lineType, shift); END_WRAP }	cv::ellipse()	group__imgproc__draw.html	Draws a simple or thick elliptic arc or fills an ellipse sector.	img : Image. center : Center of the ellipse. axes : Half of the size of the ellipse main axes. angle : Ellipse rotation angle in degrees. startAngle : Starting angle of the elliptic arc in degrees. endAngle : Ending angle of the elliptic arc in degrees. color : Ellipse color. thickness : Thickness of the ellipse arc outline, if positive. Otherwise, this indicates that a filled ellipse sector is to be drawn. lineType : Type of the ellipse boundary. See LineTypes shift : Number of fractional bits in the coordinates of the center and values of axes.	The function cv::ellipse with more parameters draws an ellipse outline, a filled ellipse, an elliptic arc, or a filled ellipse sector. The drawing code uses general parametric form. A piecewise-linear curve is used to approximate the elliptic arc boundary. If you need more control of the ellipse rendering, you can retrieve the curve using ellipse2Poly and then render it with polylines or fill it with fillPoly. If you use the first variant of the function and want to draw the whole ellipse, not an arc, pass startAngle=0 and endAngle=360. If startAngle is greater than endAngle, they are swapped. The figure below explains the meaning of the parameters to draw the blue arc.ellipse.svg Parameters of Elliptic ArcExamples: fld_lines.cpp, samples/cpp/camshiftdemo.cpp, samples/cpp/contours2.cpp, samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, and	10								
imgproc_ellipse2	CVAPI(ExceptionStatus) imgproc_ellipse2(cv::_InputOutputArray *img, MyCvBox2D box, MyCvScalar color, int thickness, int lineType) { BEGIN_WRAP cv::ellipse(*img, cpp(box), cpp(color), thickness, lineType); END_WRAP }	cv::ellipse()	group__imgproc__draw.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	img : Image. box : Alternative ellipse representation via RotatedRect. This means that the function draws an ellipse inscribed in the rotated rectangle. color : Ellipse color. thickness : Thickness of the ellipse arc outline, if positive. Otherwise, this indicates that a filled ellipse sector is to be drawn. lineType : Type of the ellipse boundary. See LineTypes		5								
imgproc_drawMarker	CVAPI(ExceptionStatus) imgproc_drawMarker(cv::_InputOutputArray *img, MyCvPoint position, MyCvScalar color, int markerType, int markerSize, int thickness, int lineType) { BEGIN_WRAP cv::drawMarker(*img, cpp(position), cpp(color), markerType, markerSize, thickness, lineType); END_WRAP }	cv::drawMarker()	group__imgproc__draw.html	Draws a marker on a predefined position in an image.	img : Image. position : The point where the crosshair is positioned. color : Line color. markerType : The specific type of marker you want to use, see MarkerTypes thickness : Line thickness. line_type : Type of the line, See LineTypes markerSize : The length of the marker axis [default = 20 pixels]	The function cv::drawMarker draws a marker on a given position in the image. For the moment several marker types are supported, see MarkerTypes for more information.Examples: samples/cpp/polar_transforms.cpp.	7								
imgproc_fillConvexPoly_Mat	CVAPI(ExceptionStatus) imgproc_fillConvexPoly_Mat(cv::Mat *img, cv::Point *pts, int npts, MyCvScalar color, int lineType, int shift) { BEGIN_WRAP cv::fillConvexPoly(*img, pts, npts, cpp(color), lineType, shift); END_WRAP }	cv::fillConvexPoly()	group__imgproc__draw.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.			6								
imgproc_fillConvexPoly_InputOutputArray	CVAPI(ExceptionStatus) imgproc_fillConvexPoly_InputOutputArray(cv::_InputOutputArray *img, cv::_InputArray *points, MyCvScalar color, int lineType, int shift) { BEGIN_WRAP cv::fillConvexPoly(*img, *points, cpp(color), lineType, shift); END_WRAP }	cv::fillConvexPoly()	group__imgproc__draw.html	Fills a convex polygon.	img : Image. points : Polygon vertices. color : Polygon color. lineType : Type of the polygon boundaries. See LineTypes shift : Number of fractional bits in the vertex coordinates.	The function cv::fillConvexPoly draws a filled convex polygon. This function is much faster than the function fillPoly . It can fill not only convex polygons but any monotonic polygon without self-intersections, that is, a polygon whose contour intersects every horizontal line (scan line) twice at the most (though, its top-most and/or the bottom edge could be horizontal).	5								
imgproc_fillPoly_Mat	CVAPI(ExceptionStatus) imgproc_fillPoly_Mat(cv::Mat *img, const cv::Point **pts, const int *npts, int ncontours, MyCvScalar color, int lineType, int shift, MyCvPoint offset) { BEGIN_WRAP cv::fillPoly(*img, pts, npts, ncontours, cpp(color), lineType, shift, cpp(offset)); END_WRAP }	cv::fillPoly()	group__imgproc__draw.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.		Examples: samples/cpp/create_mask.cpp, samples/cpp/intersectExample.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_1.cpp, and samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp.	8								
imgproc_fillPoly_InputOutputArray	CVAPI(ExceptionStatus) imgproc_fillPoly_InputOutputArray(cv::_InputOutputArray *img, cv::_InputArray *pts, MyCvScalar color, int lineType, int shift, MyCvPoint offset) { BEGIN_WRAP cv::fillPoly(*img, *pts, cpp(color), lineType, shift, cpp(offset)); END_WRAP }	cv::fillPoly()	group__imgproc__draw.html	Fills the area bounded by one or more polygons.	img : Image. pts : Array of polygons where each polygon is represented as an array of points. color : Polygon color. lineType : Type of the polygon boundaries. See LineTypes shift : Number of fractional bits in the vertex coordinates. offset : Optional offset of all points of the contours.	The function cv::fillPoly fills an area bounded by several polygonal contours. The function can fill complex areas, for example, areas with holes, contours with self-intersections (some of their parts), and so forth.	6								

imgproc_polylines_Mat	CVAPI(ExceptionStatus) imgproc_polylines_Mat(cv::Mat *img, const cv::Point **pts, const int *npts, int ncontours, int isClosed, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::polylines(*img, pts, npts, ncontours, isClosed != 0, cpp(color), thickness, lineType, shift); END_WRAP }	cv::polylines()	group__imgproc__draw.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.		Examples: fld_lines.cpp, samples/cpp/create_mask.cpp, samples/cpp/intersectExample.cpp, samples/cpp/squares.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/dnn/text_detection.cpp, and samples/tapi/squares.cpp.	9								
imgproc_polylines_InputOutputArray	CVAPI(ExceptionStatus) imgproc_polylines_InputOutputArray(cv::_InputOutputArray *img, cv::_InputArray *pts, int isClosed, MyCvScalar color, int thickness, int lineType, int shift) { BEGIN_WRAP cv::polylines(*img, *pts, isClosed != 0, cpp(color), thickness, lineType, shift); END_WRAP }	cv::polylines()	group__imgproc__draw.html	Draws several polygonal curves.	img : Image. pts : Array of polygonal curves. isClosed : Flag indicating whether the drawn polylines are closed or not. If they are closed, the function draws a line from the last vertex of each curve to its first vertex. color : Polyline color. thickness : Thickness of the polyline edges. lineType : Type of the line segments. See LineTypes shift : Number of fractional bits in the vertex coordinates.	The function cv::polylines draws one or more polygonal curves.	7								
imgproc_drawContours_vector	CVAPI(ExceptionStatus) imgproc_drawContours_vector (cv::_InputOutputArray *image, cv::Point **contours, int contoursSize1, int *contoursSize2, int contourIdx, MyCvScalar color, int thickness, int lineType, cv::Vec4i *hierarchy, int hierarchyLength, int maxLevel, MyCvPoint offset) { BEGIN_WRAP std::vector<std::vector<cv::Point> > contoursVec; for (auto i = 0; i < contoursSize1; i++) { std::vector<cv::Point> c1(contours[i], contours[i] + contoursSize2[i]);	cv::drawContours()	group__imgproc__draw.html	Draws contours outlines or filled contours.	image : Destination image. contours : All the input contours. Each contour is stored as a point vector. contourIdx : Parameter indicating a contour to draw. If it is negative, all the contours are drawn. color : Color of the contours. thickness : Thickness of lines the contours are drawn with. If it is negative (for example, thickness=FILLED), the contour interiors are drawn. lineType : Line connectivity. See LineTypes hierarchy : Optional information about hierarchy. It is only needed if you want to draw only some of the contours (see maxLevel). maxLevel : Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. This parameter is only taken into account when there is hierarchy available. offset : Optional contour shift parameter. Shift all the drawn contours by the specified ¥(¥texttt{offset}=(dx,dy)¥) .	The function draws contour outlines in the image if ¥(¥texttt{thickness} ¥ge 0¥) or fills the area bounded by the contours if ¥(¥texttt{thickness}<0¥) . The example below shows how to retrieve connected components from the binary image and label them: :#include "opencv2/imgproc.hpp"#include "opencv2/highgui.hpp"using namespace cv;using namespace std;int main(int argc, char** argv){ Mat src; // the first command-line parameter must be a filename of the binary // (black-n-white) image if(argc != 2 || !(src=imread(argv[1], 0)).data) return -1; Mat dst = Mat::zeros(src.rows, src.cols, CV_8UC3); src = src > 1; namedWindow("Source", 1); imshow("Source", src); vector<vector<Point> > contours; vector<Vec4i> hierarchy; findContours(src, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE); // iterate through all the top-level contours, // draw each connected component with its own random color int idx = 0; for(; idx >= 0; idx =	9								
imgproc_drawContours_InputArray	CVAPI(ExceptionStatus) imgproc_drawContours_InputArray(cv::_InputOutputArray *image, cv::Mat **contours, int contoursLength, int contourIdx, MyCvScalar color, int thickness, int lineType, cv::_InputArray *hierarchy, int maxLevel, MyCvPoint offset) { BEGIN_WRAP std::vector<std::vector<cv::Point> > contoursVec(contoursLength); for (auto i = 0; i < contoursLength; i++) contoursVec[i] = *contours[i]; cv::drawContours(*image, contoursVec	cv::drawContours()	group__imgproc__draw.html	Draws contours outlines or filled contours.	image : Destination image. contours : All the input contours. Each contour is stored as a point vector. contourIdx : Parameter indicating a contour to draw. If it is negative, all the contours are drawn. color : Color of the contours. thickness : Thickness of lines the contours are drawn with. If it is negative (for example, thickness=FILLED), the contour interiors are drawn. lineType : Line connectivity. See LineTypes hierarchy : Optional information about hierarchy. It is only needed if you want to draw only some of the contours (see maxLevel). maxLevel : Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. This parameter is only taken into account when there is hierarchy available. offset : Optional contour shift parameter. Shift all the drawn contours by the specified ¥(¥texttt{offset}=(dx,dy)¥) .	The function draws contour outlines in the image if ¥(¥texttt{thickness} ¥ge 0¥) or fills the area bounded by the contours if ¥(¥texttt{thickness}<0¥) . The example below shows how to retrieve connected components from the binary image and label them: :#include "opencv2/imgproc.hpp"#include "opencv2/highgui.hpp"using namespace cv;using namespace std;int main(int argc, char** argv){ Mat src; // the first command-line parameter must be a filename of the binary // (black-n-white) image if(argc != 2 || !(src=imread(argv[1], 0)).data) return -1; Mat dst = Mat::zeros(src.rows, src.cols, CV_8UC3); src = src > 1; namedWindow("Source", 1); imshow("Source", src); vector<vector<Point> > contours; vector<Vec4i> hierarchy; findContours(src, contours, hierarchy, RETR_CCOMP, CHAIN_APPROX_SIMPLE); // iterate through all the top-level contours, // draw each connected component with its own random color int idx = 0; for(; idx >= 0; idx =	9								
imgproc_clipLine1	CVAPI(ExceptionStatus) imgproc_clipLine1(MyCvSize imgSize, MyCvPoint *pt1, MyCvPoint *pt2, int* returnValue) { BEGIN_WRAP auto pt1c = cpp(*pt1), pt2c = cpp(*pt2); const auto result = cv::clipLine(cpp(imgSize), pt1c, pt2c); *pt1 = c(pt1c); *pt2 = c(pt2c); *returnValue = result ? 1 : 0; END_WRAP }	cv::clipLine()	group__imgproc__draw.html	Clips the line against the image rectangle.	imgSize : Image size. The image rectangle is Rect(0, 0, imgSize.width, imgSize.height) . pt1 : First line point. pt2 : Second line point.	The function cv::clipLine calculates a part of the line segment that is entirely within the specified rectangle. it returns false if the line segment is completely outside the rectangle. Otherwise, it returns true .	3								
imgproc_clipLine2	CVAPI(ExceptionStatus) imgproc_clipLine2(MyCvRect imgRect, MyCvPoint *pt1, MyCvPoint *pt2, int* returnValue) { BEGIN_WRAP auto pt1c = cpp(*pt1), pt2c = cpp(*pt2); const auto result = cv::clipLine(cpp(imgRect), pt1c, pt2c); *pt1 = c(pt1c); *pt2 = c(pt2c); *returnValue = result ? 1 : 0; END_WRAP }	cv::clipLine()	group__imgproc__draw.html	Clips the line against the image rectangle.	imgSize : Image size. The image rectangle is Rect(0, 0, imgSize.width, imgSize.height) . pt1 : First line point. pt2 : Second line point.	The function cv::clipLine calculates a part of the line segment that is entirely within the specified rectangle. it returns false if the line segment is completely outside the rectangle. Otherwise, it returns true .	3								
imgproc_ellipse2Poly_int	CVAPI(ExceptionStatus) imgproc_ellipse2Poly_int(MyCvPoint center, MyCvSize axes, int angle, int arcStart, int arcEnd, int delta, std::vector<cv::Point> * pts) { BEGIN_WRAP cv::ellipse2Poly(cpp(center), cpp(axes), angle, arcStart, arcEnd, delta, *pts); END_WRAP }	cv::ellipse2Poly()	group__imgproc__draw.html	Approximates an elliptic arc with a polyline.	center : Center of the arc. axes : Half of the size of the ellipse main axes. See ellipse for details. angle : Rotation angle of the ellipse in degrees. See ellipse for details. arcStart : Starting angle of the elliptic arc in degrees. arcEnd : Ending angle of the elliptic arc in degrees. delta : Angle between the subsequent polyline vertices. It defines the approximation accuracy. pts : Output vector of polyline vertices.	The function ellipse2Poly computes the vertices of a polyline that approximates the specified elliptic arc. It is used by ellipse. If arcStart is greater than arcEnd, they are swapped.	7								
imgproc_ellipse2Poly_double	CVAPI(ExceptionStatus) imgproc_ellipse2Poly_double(MyCvPoint2D64f center, MyCvSize2D64f axes, int angle, int arcStart, int arcEnd, int delta, std::vector<cv::Point2d>* pts) { BEGIN_WRAP cv::ellipse2Poly(cpp(center), cpp(axes), angle, arcStart, arcEnd, delta, *pts); END_WRAP }	cv::ellipse2Poly()	group__imgproc__draw.html	Approximates an elliptic arc with a polyline.	center : Center of the arc. axes : Half of the size of the ellipse main axes. See ellipse for details. angle : Rotation angle of the ellipse in degrees. See ellipse for details. arcStart : Starting angle of the elliptic arc in degrees. arcEnd : Ending angle of the elliptic arc in degrees. delta : Angle between the subsequent polyline vertices. It defines the approximation accuracy. pts : Output vector of polyline vertices.	The function ellipse2Poly computes the vertices of a polyline that approximates the specified elliptic arc. It is used by ellipse. If arcStart is greater than arcEnd, they are swapped.	7								
imgproc_putText	CVAPI(ExceptionStatus) imgproc_putText(cv::_InputOutputArray *img, const char *text, MyCvPoint org, int fontFace, double fontScale, MyCvScalar color, int thickness, int lineType, int bottomLeftOrigin) { BEGIN_WRAP cv::putText(*img, text, cpp(org), fontFace, fontScale, cpp(color), thickness, lineType, bottomLeftOrigin != 0); END_WRAP }	cv::putText()	group__imgproc__draw.html	Draws a text string.	img : Image. text : Text string to be drawn. org : Bottom-left corner of the text string in the image. fontFace : Font type, see HersheyFonts. fontScale : Font scale factor that is multiplied by the font-specific base size. color : Text color. thickness : Thickness of the lines used to draw a text. lineType : Line type. See LineTypes bottomLeftOrigin : When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.	The function cv::putText renders the specified text string in the image. Symbols that cannot be rendered using the specified font are replaced by question marks. See getTextSize for a text rendering code example.Examples: samples/cpp/falsecolor.cpp, samples/cpp/fitellipse.cpp, samples/cpp/intersectExample.cpp, samples/cpp/peopledetect.cpp, samples/cpp/tutorial_code/ImgProc/basic_drawing/Drawing_2.cpp, samples/cpp/tutorial_code/ImgProc/Smoothing/Smoothing.cpp, samples/cpp/warpPerspective_demo.cpp, samples/dnn/classification.cpp, samples/dnn/object_detection.cpp, samples/dnn/segmentation.cpp, samples/dnn/text_detection.cpp, and samples/tapi/hog.cpp.	9								
imgproc_getTextSize	CVAPI(ExceptionStatus) imgproc_getTextSize(const char *text, int fontFace, double fontScale, int thickness, int *baseLine, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(cv::getTextSize(text, fontFace, fontScale, thickness, baseLine)); END_WRAP }	cv::getTextSize()	group__imgproc__draw.html	Calculates the width and height of a text string.	text : Input text string. fontFace : Font to use, see HersheyFonts. fontScale : Font scale factor that is multiplied by the font-specific base size. thickness : Thickness of lines used to render the text. See putText for details. baseLine : [out]	The function cv::getTextSize calculates and returns the size of a box that contains the specified text. That is, the following code renders some text, the tight box surrounding it, and the baseline: :String text = "Funny text inside the box";int fontFace = FONT_HERSHEY_SCRIPT_SIMPLEX;double fontScale = 2;int thickness = 3;Mat img(600, 800, CV_8UC3, Scalar::all(0));int baseline=0;Size textSize = getTextSize(text, fontFace, fontScale, thickness, &baseline);baseline += thickness;// center the textPoint textOrg((img.cols - textSize.width)/2, (img.rows + textSize.height)/2);// draw the boxrectangle(img, textOrg + Point(0, baseline), textOrg + Point(textSize.width, -textSize.height), Scalar(0,0,255));// ... and the baseline firstline(img, textOrg + Point(0, thickness), textOrg + Point(textSize.width, thickness), Scalar(0, 0, 255));// then put the text itselfputText(img, text, textOrg, fontFace, fontScale, Scalar::all(255),	5								

imgproc_getFontScaleFromHeight	CVAPI(ExceptionStatus) imgproc_getFontScaleFromHeight(int fontFace, int pixelHeight, int thickness, double* returnValue) { BEGIN_WRAP *returnValue = cv::getFontScaleFromHeight(fontFace, pixelHeight, thickness); END_WRAP }	cv::getFontScaleFromHeight()	group__imgproc__draw.html	Calculates the font-specific size to use to achieve a given height in pixels.	fontFace : Font to use, see cv::HersheyFonts. pixelHeight : Pixel height to compute the fontScale for thickness : Thickness of lines used to render the text.See putText for details.	See alsocv::putText	3									
imgproc_createCLAHE	CVAPI(ExceptionStatus) imgproc_createCLAHE(double clipLimit, MyCvSize tileGridSize, cv::Ptr<cv::CLAHE> **returnValue) { BEGIN_WRAP const auto ret = cv::createCLAHE(clipLimit, cpp(tileGridSize)); *returnValue = clone(ret); END_WRAP }	cv::createCLAHE()	group__imgproc__hist.html	Creates a smart pointer to a cv::CLAHE class and initializes it.	clipLimit : Threshold for contrast limiting. tileGridSize : Size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles. tileGridSize defines the number of tiles in row and column.		2									
imgproc_Ptr_CLAHE_delete	CVAPI(ExceptionStatus) imgproc_Ptr_CLAHE_delete(cv::Ptr<cv::CLAHE> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::CLAHE	classcv_1_1CLAHE.html	Base class for Contrast Limited Adaptive Histogram Equalization.								•				
imgproc_Ptr_CLAHE_get	CVAPI(ExceptionStatus) imgproc_Ptr_CLAHE_get(cv::Ptr<cv::CLAHE> *obj, cv::CLAHE **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::CLAHE	classcv_1_1CLAHE.html	Base class for Contrast Limited Adaptive Histogram Equalization.										•		
imgproc_CLAHE_apply	CVAPI(ExceptionStatus) imgproc_CLAHE_apply(cv::CLAHE *obj, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP obj->apply(*src, *dst); END_WRAP }	cv::CLAHE::apply()	classcv_1_1CLAHE.html	Equalizes the histogram of a grayscale image using Contrast Limited Adaptive Histogram Equalization.	src : Source image of type CV_8UC1 or CV_16UC1. dst : Destination image.		2									
imgproc_CLAHE_setClipLimit	CVAPI(ExceptionStatus) imgproc_CLAHE_setClipLimit(cv::CLAHE *obj, double clipLimit) { BEGIN_WRAP obj->setClipLimit(clipLimit); END_WRAP }	cv::CLAHE::setClipLimit()	classcv_1_1CLAHE.html	Sets threshold for contrast limiting.	clipLimit : threshold value.		1									
imgproc_CLAHE_getClipLimit	CVAPI(ExceptionStatus) imgproc_CLAHE_getClipLimit(cv::CLAHE *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getClipLimit(); END_WRAP }	cv::CLAHE::getClipLimit()	classcv_1_1CLAHE.html	Returns threshold value for contrast limiting.			1									
imgproc_CLAHE_setTilesGridSize	CVAPI(ExceptionStatus) imgproc_CLAHE_setTilesGridSize(cv::CLAHE *obj, MyCvSize tileGridSize) { BEGIN_WRAP obj->setTilesGridSize(cpp(tileGridSize)); END_WRAP }	cv::CLAHE::setTilesGridSize()	classcv_1_1CLAHE.html	Sets size of grid for histogram equalization. Input image will be divided into equally sized rectangular tiles.	tileGridSize : defines the number of tiles in row and column.		1									
imgproc_CLAHE_getTilesGridSize	CVAPI(ExceptionStatus) imgproc_CLAHE_getTilesGridSize(cv::CLAHE *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->getTilesGridSize()); END_WRAP }	cv::CLAHE::getTilesGridSize()	classcv_1_1CLAHE.html	Returns Size defines the number of tiles in row and column.			1									
imgproc_CLAHE_collectGarbage	CVAPI(ExceptionStatus) imgproc_CLAHE_collectGarbage(cv::CLAHE *obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::CLAHE::collectGarbage()	classcv_1_1CLAHE.html				1									
imgproc_GeneralizedHough_setTemplate1	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setTemplate1(cv::GeneralizedHough *obj, cv::_InputArray *templ, MyCvPoint templCenter) { BEGIN_WRAP obj->setTemplate(*templ, cpp(templCenter)); END_WRAP }	cv::GeneralizedHough::setTemplate()	classcv_1_1GeneralizedHough.html	set template to search			2									
imgproc_GeneralizedHough_setTemplate2	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setTemplate2(cv::GeneralizedHough *obj, cv::_InputArray *edges, cv::_InputArray *dx, cv::_InputArray *dy, MyCvPoint templCenter) { BEGIN_WRAP obj->setTemplate(*edges, *dx, *dy, cpp(templCenter)); END_WRAP }	cv::GeneralizedHough::setTemplate()	classcv_1_1GeneralizedHough.html	set template to search			4									
imgproc_GeneralizedHough_detect1	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_detect1(cv::GeneralizedHough *obj, cv::_InputArray *image, cv::_OutputArray *positions, cv::_OutputArray *votes) { BEGIN_WRAP obj->detect(*image, *positions, entity(votes)); END_WRAP }	cv::GeneralizedHough::detect()	classcv_1_1GeneralizedHough.html	find template on image			3									
imgproc_GeneralizedHough_detect2	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_detect2(cv::GeneralizedHough *obj, cv::_InputArray *edges, cv::_InputArray *dx, cv::_InputArray *dy, cv::_OutputArray *positions, cv::_OutputArray *votes) { BEGIN_WRAP obj->detect(*edges, *dx, *dy, *positions, entity(votes)); END_WRAP }	cv::GeneralizedHough::detect()	classcv_1_1GeneralizedHough.html	find template on image			5									
imgproc_GeneralizedHough_setCannyLowThres	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setCannyLowThresh(cv::GeneralizedHough *obj, int val) { BEGIN_WRAP obj->setCannyLowThresh(val); END_WRAP }	cv::GeneralizedHough::setCannyLowThresh()	classcv_1_1GeneralizedHough.html	Canny low threshold.			1									
imgproc_GeneralizedHough_getCannyLowThres	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_getCannyLowThresh(cv::GeneralizedHough *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getCannyLowThresh(); END_WRAP }	cv::GeneralizedHough::getCannyLowThresh()	classcv_1_1GeneralizedHough.html				1									
imgproc_GeneralizedHough_setCannyHighThre	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setCannyHighThresh(cv::GeneralizedHough *obj, int val) { BEGIN_WRAP obj->setCannyHighThresh(val); END_WRAP }	cv::GeneralizedHough::setCannyHighThresh()	classcv_1_1GeneralizedHough.html	Canny high threshold.			1									

imgproc_GeneralizedHough_getCannyHighThre	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_getCannyHighThresh(cv::GeneralizedHough *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getCannyHighThresh(); END_WRAP }	cv::GeneralizedHough::getCannyHighThresh()	classcv_1_1GeneralizedHough.html					1									
imgproc_GeneralizedHough_setMinDist	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setMinDist(cv::GeneralizedHough *obj, double val) { BEGIN_WRAP obj->setMinDist(val); END_WRAP }	cv::GeneralizedHough::setMinDist()	classcv_1_1GeneralizedHough.html	Minimum distance between the centers of the detected objects.				1									
imgproc_GeneralizedHough_getMinDist	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_getMinDist(cv::GeneralizedHough *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMinDist(); END_WRAP }	cv::GeneralizedHough::getMinDist()	classcv_1_1GeneralizedHough.html					1									
imgproc_GeneralizedHough_setDp	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setDp(cv::GeneralizedHough *obj, double val) { BEGIN_WRAP obj->setDp(val); END_WRAP }	cv::GeneralizedHough::setDp()	classcv_1_1GeneralizedHough.html	Inverse ratio of the accumulator resolution to the image resolution.				1									
imgproc_GeneralizedHough_getDp	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_getDp(cv::GeneralizedHough *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getDp(); END_WRAP }	cv::GeneralizedHough::getDp()	classcv_1_1GeneralizedHough.html					1									
imgproc_GeneralizedHough_setMaxBufferSize	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_setMaxBufferSize(cv::GeneralizedHough *obj, int val) { BEGIN_WRAP obj->setMaxBufferSize(val); END_WRAP }	cv::GeneralizedHough::setMaxBufferSize()	classcv_1_1GeneralizedHough.html	Maximal size of inner buffers.				1									
imgproc_GeneralizedHough_getMaxBufferSize	CVAPI(ExceptionStatus) imgproc_GeneralizedHough_getMaxBufferSize(cv::GeneralizedHough *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxBufferSize(); END_WRAP }	cv::GeneralizedHough::getMaxBufferSize()	classcv_1_1GeneralizedHough.html					1									
imgproc_createGeneralizedHoughBallard	CVAPI(ExceptionStatus) imgproc_createGeneralizedHoughBallard(cv::Ptr<cv::GeneralizedHoughBallard> **returnValue) { BEGIN_WRAP const auto ptr = cv::createGeneralizedHoughBallard(); *returnValue = new cv::Ptr<cv::GeneralizedHoughBallard>(ptr); END_WRAP }	cv::createGeneralizedHoughBallard()	group__imgproc__shape.html	Creates a smart pointer to a cv::GeneralizedHoughBallard class and initializes it.				1									
imgproc_Ptr_GeneralizedHoughBallard_get	CVAPI(ExceptionStatus) imgproc_Ptr_GeneralizedHoughBallard_get(cv::Ptr<cv::GeneralizedHoughBallard> *obj, cv::GeneralizedHoughBallard **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::GeneralizedHoughBallard	classcv_1_1GeneralizedHoughBallard.html	finds arbitrary template in the grayscale image using Generalized Hough Transform Detects position only without translation and rotation [14] .									•		
imgproc_Ptr_GeneralizedHoughBallard_delete	CVAPI(ExceptionStatus) imgproc_Ptr_GeneralizedHoughBallard_delete(cv::Ptr<cv::GeneralizedHoughBallard> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::GeneralizedHoughBallard	classcv_1_1GeneralizedHoughBallard.html	finds arbitrary template in the grayscale image using Generalized Hough Transform Detects position only without translation and rotation [14] .									•			
imgproc_GeneralizedHoughBallard_setLevels	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughBallard_setLevels(cv::GeneralizedHoughBallard *obj, int val) { BEGIN_WRAP obj->setLevels(val); END_WRAP }	cv::GeneralizedHoughBallard::setLevels()	classcv_1_1GeneralizedHoughBallard.html	R-Table levels.				1									
imgproc_GeneralizedHoughBallard_getLevels	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughBallard_getLevels(cv::GeneralizedHoughBallard *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getLevels(); END_WRAP }	cv::GeneralizedHoughBallard::getLevels()	classcv_1_1GeneralizedHoughBallard.html					1									
imgproc_GeneralizedHoughBallard_setVotesTh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughBallard_setVotesThreshold(cv::GeneralizedHoughBallard *obj, int val) { BEGIN_WRAP obj->setVotesThreshold(val); END_WRAP }	cv::GeneralizedHoughBallard::setVotesThreshold()	classcv_1_1GeneralizedHoughBallard.html	The accumulator threshold for the template centers at the detection stage. The smaller it is, the more false positions may be detected.				1									
imgproc_GeneralizedHoughBallard_getVotesTh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughBallard_getVotesThreshold(cv::GeneralizedHoughBallard *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getVotesThreshold(); END_WRAP }	cv::GeneralizedHoughBallard::getVotesThreshold()	classcv_1_1GeneralizedHoughBallard.html					1									
imgproc_createGeneralizedHoughGuil	CVAPI(ExceptionStatus) imgproc_createGeneralizedHoughGuil(cv::Ptr<cv::GeneralizedHoughGuil> **returnValue) { BEGIN_WRAP const auto ptr = cv::createGeneralizedHoughGuil(); *returnValue = new cv::Ptr<cv::GeneralizedHoughGuil>(ptr); END_WRAP }	cv::createGeneralizedHoughGuil()	group__imgproc__shape.html	Creates a smart pointer to a cv::GeneralizedHoughGuil class and initializes it.				1									
imgproc_Ptr_GeneralizedHoughGuil_get	CVAPI(ExceptionStatus) imgproc_Ptr_GeneralizedHoughGuil_get(cv::Ptr<cv::GeneralizedHoughGuil> *obj, cv::GeneralizedHoughGuil **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::GeneralizedHoughGuil	classcv_1_1GeneralizedHoughGuil.html	finds arbitrary template in the grayscale image using Generalized Hough Transform Detects position, translation and rotation [100] .									•		
imgproc_Ptr_GeneralizedHoughGuil_delete	CVAPI(ExceptionStatus) imgproc_Ptr_GeneralizedHoughGuil_delete(cv::Ptr<cv::GeneralizedHoughGuil> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::GeneralizedHoughGuil	classcv_1_1GeneralizedHoughGuil.html	finds arbitrary template in the grayscale image using Generalized Hough Transform Detects position, translation and rotation [100] .									•			

imgproc_GeneralizedHoughGuil_setXi	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setXi(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setXi(val); END_WRAP }	cv::GeneralizedHoughGuil::setXi()	classcv_1_1GeneralizedHoughGuil.html	Angle difference in degrees between two points in feature.			1										
imgproc_GeneralizedHoughGuil_getXi	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getXi(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getXi(); END_WRAP }	cv::GeneralizedHoughGuil::getXi()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setLevels	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setLevels(cv::GeneralizedHoughGuil *obj, int val) { BEGIN_WRAP obj->setLevels(val); END_WRAP }	cv::GeneralizedHoughGuil::setLevels()	classcv_1_1GeneralizedHoughGuil.html	Feature table levels.			1										
imgproc_GeneralizedHoughGuil_getLevels	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getLevels(cv::GeneralizedHoughGuil *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getLevels(); END_WRAP }	cv::GeneralizedHoughGuil::getLevels()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setAngleEpsilo	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setAngleEpsilon(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setAngleEpsilon(val); END_WRAP }	cv::GeneralizedHoughGuil::setAngleEpsilon()	classcv_1_1GeneralizedHoughGuil.html	Maximal difference between angles that treated as equal.			1										
imgproc_GeneralizedHoughGuil_getAngleEpsilo	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getAngleEpsilon(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getAngleEpsilon(); END_WRAP }	cv::GeneralizedHoughGuil::getAngleEpsilon()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setMinAngle	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setMinAngle(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setMinAngle(val); END_WRAP }	cv::GeneralizedHoughGuil::setMinAngle()	classcv_1_1GeneralizedHoughGuil.html	Minimal rotation angle to detect in degrees.			1										
imgproc_GeneralizedHoughGuil_getMinAngle	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getMinAngle(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMinAngle(); END_WRAP }	cv::GeneralizedHoughGuil::getMinAngle()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setMaxAngle	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setMaxAngle(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setMaxAngle(val); END_WRAP }	cv::GeneralizedHoughGuil::setMaxAngle()	classcv_1_1GeneralizedHoughGuil.html	Maximal rotation angle to detect in degrees.			1										
imgproc_GeneralizedHoughGuil_getMaxAngle	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getMaxAngle(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxAngle(); END_WRAP }	cv::GeneralizedHoughGuil::getMaxAngle()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setAngleStep	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setAngleStep(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setAngleStep(val); END_WRAP }	cv::GeneralizedHoughGuil::setAngleStep()	classcv_1_1GeneralizedHoughGuil.html	Angle step in degrees.			1										
imgproc_GeneralizedHoughGuil_getAngleStep	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getAngleStep(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getAngleStep(); END_WRAP }	cv::GeneralizedHoughGuil::getAngleStep()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setAngleThresl	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setAngleThresh(cv::GeneralizedHoughGuil *obj, int val) { BEGIN_WRAP obj->setAngleThresh(val); END_WRAP }	cv::GeneralizedHoughGuil::setAngleThresh()	classcv_1_1GeneralizedHoughGuil.html	Angle votes threshold.			1										
imgproc_GeneralizedHoughGuil_getAngleThresl	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getAngleThresh(cv::GeneralizedHoughGuil *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getAngleThresh(); END_WRAP }	cv::GeneralizedHoughGuil::getAngleThresh()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setMinScale	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setMinScale(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setMinScale(val); END_WRAP }	cv::GeneralizedHoughGuil::setMinScale()	classcv_1_1GeneralizedHoughGuil.html	Minimal scale to detect.			1										
imgproc_GeneralizedHoughGuil_getMinScale	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getMinScale(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMinScale(); END_WRAP }	cv::GeneralizedHoughGuil::getMinScale()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setMaxScale	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setMaxScale(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setMaxScale(val); END_WRAP }	cv::GeneralizedHoughGuil::setMaxScale()	classcv_1_1GeneralizedHoughGuil.html	Maximal scale to detect.			1										
imgproc_GeneralizedHoughGuil_getMaxScale	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getMaxScale(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxScale(); END_WRAP }	cv::GeneralizedHoughGuil::getMaxScale()	classcv_1_1GeneralizedHoughGuil.html				1										
imgproc_GeneralizedHoughGuil_setScaleStep	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setScaleStep(cv::GeneralizedHoughGuil *obj, double val) { BEGIN_WRAP obj->setScaleStep(val); END_WRAP }	cv::GeneralizedHoughGuil::setScaleStep()	classcv_1_1GeneralizedHoughGuil.html	Scale step.			1										

imgproc_GeneralizedHoughGuil_getScaleStep	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getScaleStep(cv::GeneralizedHoughGuil *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getScaleStep(); END_WRAP }	cv::GeneralizedHoughGuil::getScaleStep()	classcv_1_1GeneralizedHoughGuil.html					1										
imgproc_GeneralizedHoughGuil_setScaleThresh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setScaleThresh(cv::GeneralizedHoughGuil *obj, int val) { BEGIN_WRAP obj->setScaleThresh(val); END_WRAP }	cv::GeneralizedHoughGuil::setScaleThresh()	classcv_1_1GeneralizedHoughGuil.html	Scale votes threshold.				1										
imgproc_GeneralizedHoughGuil_getScaleThresh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getScaleThresh(cv::GeneralizedHoughGuil *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getScaleThresh(); END_WRAP }	cv::GeneralizedHoughGuil::getScaleThresh()	classcv_1_1GeneralizedHoughGuil.html					1										
imgproc_GeneralizedHoughGuil_setPosThresh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_setPosThresh(cv::GeneralizedHoughGuil *obj, int val) { BEGIN_WRAP obj->setPosThresh(val); END_WRAP }	cv::GeneralizedHoughGuil::setPosThresh()	classcv_1_1GeneralizedHoughGuil.html	Position votes threshold.				1										
imgproc_GeneralizedHoughGuil_getPosThresh	CVAPI(ExceptionStatus) imgproc_GeneralizedHoughGuil_getPosThresh(cv::GeneralizedHoughGuil *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getPosThresh(); END_WRAP }	cv::GeneralizedHoughGuil::getPosThresh()	classcv_1_1GeneralizedHoughGuil.html					1										
imgproc_LineIterator_new	CVAPI(ExceptionStatus) imgproc_LineIterator_new(cv::Mat *img, MyCvPoint pt1, MyCvPoint pt2, int connectivity, int leftToRight, cv::LineIterator** returnValue) { BEGIN_WRAP *returnValue = new cv::LineIterator(*img, cpp(pt1), cpp(pt2), connectivity, leftToRight != 0); END_WRAP }	cv::LineIterator	classcv_1_1LineIterator.html	Line iterator. The class is used to iterate over all the pixels on the raster line segment connecting two specified points. The class LineIterator is used to get each pixel of a raster line. It can be treated as versatile implementation of the Bresenham algorithm where you can stop at each pixel and do some extra processing, for example, grab pixel values along the line or draw a line with an effect (for example, with XOR operation). The number of pixels along the line is stored in LineIterator::count. The method LineIterator::pos returns the current position in the image: // grabs pixels along the line (pt1, pt2)// from 8-bit 3-channel image to the bufferLineIterator it(img, pt1, pt2, 8);LineIterator it2 = it;vector<Vec3b> buf(it.count);for(int i = 0; i < it.count; i++, ++it) buf[i] = *(const Vec3b*)*it;// alternative way of iterating through the linefor(int i = 0; i < it2.count; i++, ++it2){ Vec3b val = img.at<Vec3b>(it2.pos()); CV_Assert(buf[i] == val);}								•			
imgproc_LineIterator_delete	CVAPI(ExceptionStatus) imgproc_LineIterator_delete(cv::LineIterator *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::LineIterator	classcv_1_1LineIterator.html	Line iterator. The class is used to iterate over all the pixels on the raster line segment connecting two specified points. The class LineIterator is used to get each pixel of a raster line. It can be treated as versatile implementation of the Bresenham algorithm where you can stop at each pixel and do some extra processing, for example, grab pixel values along the line or draw a line with an effect (for example, with XOR operation). The number of pixels along the line is stored in LineIterator::count. The method LineIterator::pos returns the current position in the image: // grabs pixels along the line (pt1, pt2)// from 8-bit 3-channel image to the bufferLineIterator it(img, pt1, pt2, 8);LineIterator it2 = it;vector<Vec3b> buf(it.count);for(int i = 0; i < it.count; i++, ++it) buf[i] = *(const Vec3b*)*it;// alternative way of iterating through the linefor(int i = 0; i < it2.count; i++, ++it2){ Vec3b val = img.at<Vec3b>(it2.pos()); CV_Assert(buf[i] == val);}									•		
imgproc_LineIterator_getValuePosAndShiftToN	CVAPI(ExceptionStatus) imgproc_LineIterator_getValuePosAndShiftToNext(cv::LineIterator* obj, uchar** returnValue, MyCvPoint *returnPos) { BEGIN_WRAP *returnValue = **obj; *returnPos = c(obj->pos()); (*obj)++; END_WRAP }	cv::LineIterator	classcv_1_1LineIterator.html	Line iterator. The class is used to iterate over all the pixels on the raster line segment connecting two specified points. The class LineIterator is used to get each pixel of a raster line. It can be treated as versatile implementation of the Bresenham algorithm where you can stop at each pixel and do some extra processing, for example, grab pixel values along the line or draw a line with an effect (for example, with XOR operation). The number of pixels along the line is stored in LineIterator::count. The method LineIterator::pos returns the current position in the image: // grabs pixels along the line (pt1, pt2)// from 8-bit 3-channel image to the bufferLineIterator it(img, pt1, pt2, 8);LineIterator it2 = it;vector<Vec3b> buf(it.count);for(int i = 0; i < it.count; i++, ++it) buf[i] = *(const Vec3b*)*it;// alternative way of iterating through the linefor(int i = 0; i < it2.count; i++, ++it2){ Vec3b val = img.at<Vec3b>(it2.pos()); CV_Assert(buf[i] == val);}						•						
imgproc_LineIterator_ptr_get	CVAPI(ExceptionStatus) imgproc_LineIterator_ptr_get(cv::LineIterator *obj, uchar **returnValue) { BEGIN_WRAP *returnValue = obj->ptr; END_WRAP }	cv::LineIterator	classcv_1_1LineIterator.html	Line iterator. The class is used to iterate over all the pixels on the raster line segment connecting two specified points. The class LineIterator is used to get each pixel of a raster line. It can be treated as versatile implementation of the Bresenham algorithm where you can stop at each pixel and do some extra processing, for example, grab pixel values along the line or draw a line with an effect (for example, with XOR operation). The number of pixels along the line is stored in LineIterator::count. The method LineIterator::pos returns the current position in the image: // grabs pixels along the line (pt1, pt2)// from 8-bit 3-channel image to the bufferLineIterator it(img, pt1, pt2, 8);LineIterator it2 = it;vector<Vec3b> buf(it.count);for(int i = 0; i < it.count; i++, ++it) buf[i] = *(const Vec3b*)*it;// alternative way of iterating through the linefor(int i = 0; i < it2.count; i++, ++it2){ Vec3b val = img.at<Vec3b>(it2.pos()); CV_Assert(buf[i] == val);}										•		
imgproc_LineIterator_ptr0_get	CVAPI(ExceptionStatus) imgproc_LineIterator_ptr0_get(cv::LineIterator *obj, const uchar** returnValue) { BEGIN_WRAP *returnValue = obj->ptr0; END_WRAP }	cv::LineIterator::ptr0	classcv_1_1LineIterator.html								•							
imgproc_LineIterator_step_get	CVAPI(ExceptionStatus) imgproc_LineIterator_step_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->step; END_WRAP }	cv::LineIterator::step	classcv_1_1LineIterator.html								•							
imgproc_LineIterator_elemSize_get	CVAPI(ExceptionStatus) imgproc_LineIterator_elemSize_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->elemSize; END_WRAP }	cv::LineIterator::elemSize	classcv_1_1LineIterator.html								•							
imgproc_LineIterator_err_get	CVAPI(ExceptionStatus) imgproc_LineIterator_err_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->err; END_WRAP }	cv::LineIterator::err	classcv_1_1LineIterator.html								•							
imgproc_LineIterator_count_get	CVAPI(ExceptionStatus) imgproc_LineIterator_count_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->count; END_WRAP }	cv::LineIterator::count	classcv_1_1LineIterator.html								•							
imgproc_LineIterator_minusDelta_get	CVAPI(ExceptionStatus) imgproc_LineIterator_minusDelta_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->minusDelta; END_WRAP }	cv::LineIterator::minusDelta	classcv_1_1LineIterator.html								•							

imgproc_LineIterator_plusDelta_get	CVAPI(ExceptionStatus) imgproc_LineIterator_plusDelta_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->plusDelta; END_WRAP }	cv::LineIterator::plusDelta	classcv_1_1LineIterator.html								•					
imgproc_LineIterator_minusStep_get	CVAPI(ExceptionStatus) imgproc_LineIterator_minusStep_get(cv::LineIterator *obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->minusStep; END_WRAP }	cv::LineIterator::minusStep	classcv_1_1LineIterator.html								•					
imgproc_LineIterator_plusStep_get	CVAPI(ExceptionStatus) imgproc_LineIterator_plusStep_get(cv::LineIterator *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->plusStep; END_WRAP }	cv::LineIterator::plusStep	classcv_1_1LineIterator.html								•					
imgproc_seg_ISMB_new	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_new(cv::segmentation::IntelligentScissorsMB** returnValue) { BEGIN_WRAP *returnValue = new cv::segmentation::IntelligentScissorsMB(); END_WRAP }	cv::segmentation::IntelligentScissorsMB	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Intelligent Scissors image segmentation. This class is used to find the path (contour) between two points which can be used for image segmentation. Usage example: segmentation::IntelligentScissorsMB tool; tool.setEdgeFeatureCannyParameters(16, 100) // using Canny() as edge feature extractor .setGradientMagnitudeMaxLimit(200); // calculate image features tool.applyImage(image); // calculate map for specified source point Point source_point(200, 100); tool.buildMap(source_point); // fast fetching of contours // for specified target point and the pre-calculated map (stored internally) Point target_point(400, 300); std::vector<Point> pts; tool.getContour(target_point, pts); Reference: "Intelligent Scissors for Image Composition" algorithm designed by Eric N. Mortensen and William A. Barrett, Brigham Young University [178]									•			
imgproc_seg_ISMB_delete	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_delete(cv::segmentation::IntelligentScissorsMB *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::segmentation::IntelligentScissorsMB	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Intelligent Scissors image segmentation. This class is used to find the path (contour) between two points which can be used for image segmentation. Usage example: segmentation::IntelligentScissorsMB tool; tool.setEdgeFeatureCannyParameters(16, 100) // using Canny() as edge feature extractor .setGradientMagnitudeMaxLimit(200); // calculate image features tool.applyImage(image); // calculate map for specified source point Point source_point(200, 100); tool.buildMap(source_point); // fast fetching of contours // for specified target point and the pre-calculated map (stored internally) Point target_point(400, 300); std::vector<Point> pts; tool.getContour(target_point, pts); Reference: "Intelligent Scissors for Image Composition" algorithm designed by Eric N. Mortensen and William A. Barrett, Brigham Young University [178]										•	
imgproc_seg_ISMB_setWeights	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_setWeights(cv::segmentation::IntelligentScissorsMB *obj, float weight_non_edge, float weight_gradient_direction, float weight_gradient_magnitude) { BEGIN_WRAP obj->setWeights(weight_non_edge, weight_gradient_direction, weight_gradient_magnitude); END_WRAP }	cv::segmentation::IntelligentScissorsMB::setWeights()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Specify weights of feature functions.	weight_non_edge : Specify cost of non-edge pixels (default: 0.43f) weight_gradient_direction : Specify cost of gradient direction function (default: 0.43f) weight_gradient_magnitude : Specify cost of gradient magnitude function (default: 0.14f)	Consider keeping weights normalized (sum of weights equals to 1.0) Discrete dynamic programming (DP) goal is minimization of costs between pixels.	3								
imgproc_seg_ISMB_setGradientMagnitudeMaxLimit	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_setGradientMagnitudeMaxLimit(cv::segmentation::IntelligentScissorsMB *obj, float gradient_magnitude_threshold_max) { BEGIN_WRAP obj->setGradientMagnitudeMaxLimit(gradient_magnitude_threshold_max); END_WRAP }	cv::segmentation::IntelligentScissorsMB::setGradientMagnitudeMaxLimit()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Specify gradient magnitude max value threshold.	gradient_magnitude_threshold_max : Specify gradient magnitude max value threshold (default: 0, disabled)	Zero limit value is used to disable gradient magnitude thresholding (default behavior, as described in original article). Otherwize pixels with gradient magnitude >= threshold have zero cost.NoteThresholding should be used for images with irregular regions (to avoid stuck on parameters from high-contract areas, like embedded logos).	1								
imgproc_seg_ISMB_setEdgeFeatureZeroCrossingParameters	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_setEdgeFeatureZeroCrossingParameters(cv::segmentation::IntelligentScissorsMB *obj, float gradient_magnitude_min_value) { BEGIN_WRAP obj->setEdgeFeatureZeroCrossingParameters(gradient_magnitude_min_value); END_WRAP }	cv::segmentation::IntelligentScissorsMB::setEdgeFeatureZeroCrossingParameters()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Switch to "Laplacian Zero-Crossing" edge feature extractor and specify its parameters.	gradient_magnitude_min_value : Minimal gradient magnitude value for edge pixels (default: 0, check is disabled)	This feature extractor is used by default according to article.Implementation has additional filtering for regions with low-amplitude noise. This filtering is enabled through parameter of minimal gradient amplitude (use some small value 4, 8, 16).NoteCurrent implementation of this feature extractor is based on processing of grayscale images (color image is converted to grayscale image first). Canny edge detector is a bit slower, but provides better results (especially on color images): use setEdgeFeatureCannyParameters().	1								
imgproc_seg_ISMB_setEdgeFeatureCannyParameters	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_setEdgeFeatureCannyParameters(cv::segmentation::IntelligentScissorsMB *obj, double threshold1, double threshold2, int apertureSize, int L2gradient) { BEGIN_WRAP obj->setEdgeFeatureCannyParameters(threshold1, threshold2, apertureSize, L2gradient != 0); END_WRAP }	cv::segmentation::IntelligentScissorsMB::setEdgeFeatureCannyParameters()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Switch edge feature extractor to use Canny edge detector.		Note"Laplacian Zero-Crossing" feature extractor is used by default (following to original article)See alsoCanny	4								
imgproc_seg_ISMB_applyImage	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_applyImage(cv::segmentation::IntelligentScissorsMB *obj, cv::_InputArray *image) { BEGIN_WRAP obj->applyImage(*image); END_WRAP }	cv::segmentation::IntelligentScissorsMB::applyImage()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Specify input image and extract image features.	image : input image. Type is CV_8UC1 / CV_8UC3		1								
imgproc_seg_ISMB_applyImageFeatures	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_applyImageFeatures(cv::segmentation::IntelligentScissorsMB *obj, cv::_InputArray *non_edge, cv::_InputArray *gradient_direction, cv::_InputArray *gradient_magnitude, cv::_InputArray *image) { BEGIN_WRAP obj->applyImageFeatures(*non_edge, *gradient_direction, *gradient_magnitude, entity(image)); END_WRAP }	cv::segmentation::IntelligentScissorsMB::applyImageFeatures()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Specify custom features of imput image.	non_edge : Specify cost of non-edge pixels. Type is CV_8UC1. Expected values are {0, 1}. gradient_direction : Specify gradient direction feature. Type is CV_32FC2. Values are expected to be normalized: x^2 + y^2 == 1 gradient_magnitude : Specify cost of gradient magnitude function: Type is CV_32FC1. Values should be in range [0, 1]. image : Optional parameter. Must be specified if subset of features is specified (non-specified features are calculated internally)	Customized advanced variant of applyImage() call.	4								
imgproc_seg_ISMB_buildMap	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_buildMap(cv::segmentation::IntelligentScissorsMB *obj, MyCvPoint sourcePt) { BEGIN_WRAP obj->buildMap(cpp(sourcePt)); END_WRAP }	cv::segmentation::IntelligentScissorsMB::buildMap()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Prepares a map of optimal paths for the given source point on the image.	sourcePt : The source point used to find the paths	Noteapply3mage() / apply3mageFeatures() must be called before this call	1								

imgproc_seg_ISMB_getContour	CVAPI(ExceptionStatus) imgproc_segmentation_IntelligentScissorsMB_getContour(cv::segmentation::IntelligentScissorsMB *obj, MyCvPoint targetPt, cv::_OutputArray *contour, int backward) { BEGIN_WRAP obj- >getContour(cpp(targetPt), *contour, backward != 0); END_WRAP }	cv::segmentation::IntelligentScissorsMB::getContour()	classcv_1_1segmentation_1_1IntelligentScissorsMB.html	Extracts optimal contour for the given target point on the image.	targetPt : The target point contour : [out] backward : Flag to indicate reverse order of retrived pixels (use "true" value to fetch points from the target to the source point)	NotebuildMap() must be called before this call	3									
imgproc_Subdiv2D_new1	CVAPI(ExceptionStatus) imgproc_Subdiv2D_new1(cv::Subdiv2D **returnValue) { BEGIN_WRAP *returnValue = new cv::Subdiv2D; END_WRAP }	cv::Subdiv2D										•				
imgproc_Subdiv2D_new2	CVAPI(ExceptionStatus) imgproc_Subdiv2D_new2(MyCvRect rect, cv::Subdiv2D **returnValue) { BEGIN_WRAP *returnValue = new cv::Subdiv2D(cpp(rect)); END_WRAP }	cv::Subdiv2D										•				
imgproc_Subdiv2D_delete	CVAPI(ExceptionStatus) imgproc_Subdiv2D_delete(cv::Subdiv2D *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::Subdiv2D											•			
imgproc_Subdiv2D_initDelaunay	CVAPI(ExceptionStatus) imgproc_Subdiv2D_initDelaunay(cv::Subdiv2D *obj, CvRect rect) { BEGIN_WRAP obj->initDelaunay(rect); END_WRAP }	cv::Subdiv2D::initDelaunay()	classcv_1_1Subdiv2D.html	Creates a new empty Delaunay subdivision.	rect : Rectangle that includes all of the 2D points that are to be added to the subdivision.		1									
imgproc_Subdiv2D_insert1	CVAPI(ExceptionStatus) imgproc_Subdiv2D_insert1(cv::Subdiv2D *obj, MyCvPoint2D32f pt, int *returnValue) { BEGIN_WRAP *returnValue = obj->insert(cpp(pt)); END_WRAP }	cv::Subdiv2D::insert()	classcv_1_1Subdiv2D.html	Insert a single point into a Delaunay triangulation.	pt : Point to insert.	The function inserts a single point into a subdivision and modifies the subdivision topology appropriately. If a point with the same coordinates exists already, no new point is added.*Note:*If the point is outside of the triangulation specified rect a runtime error is raised.	1									
imgproc_Subdiv2D_insert2	CVAPI(ExceptionStatus) imgproc_Subdiv2D_insert2(cv::Subdiv2D *obj, MyCvPoint2D32f *ptArray, int length) { BEGIN_WRAP std::vector<cv::Point2f> ptVec(length); for (int i = 0; i < length; i++) { ptVec[i] = cpp(ptArray[i]); } obj->insert(ptVec); END_WRAP }	cv::Subdiv2D::insert()	classcv_1_1Subdiv2D.html	Insert a single point into a Delaunay triangulation.	pt : Point to insert.	The function inserts a single point into a subdivision and modifies the subdivision topology appropriately. If a point with the same coordinates exists already, no new point is added.*Note:*If the point is outside of the triangulation specified rect a runtime error is raised.	1									
imgproc_Subdiv2D_locate	CVAPI(ExceptionStatus) imgproc_Subdiv2D_locate(cv::Subdiv2D *obj, MyCvPoint2D32f pt, int *edge, int *vertex, int *returnValue) { BEGIN_WRAP *returnValue = obj->locate(cpp(pt), *edge, *vertex); END_WRAP }	cv::Subdiv2D::locate()	classcv_1_1Subdiv2D.html	Returns the location of a point within a Delaunay triangulation.	pt : Point to locate. edge : Output edge that the point belongs to or is located to the right of it. vertex : Optional output vertex the input point coincides with.	The function locates the input point within the subdivision and gives one of the triangle edges or vertices.	3									
imgproc_Subdiv2D_findNearest	CVAPI(ExceptionStatus) imgproc_Subdiv2D_findNearest(cv::Subdiv2D *obj, MyCvPoint2D32f pt, MyCvPoint2D32f* nearestPt, int *returnValue) { BEGIN_WRAP cv::Point2f nearestPt0; *returnValue = obj->findNearest(cpp(pt), &nearestPt0); *nearestPt = c(nearestPt0); END_WRAP }	cv::Subdiv2D::findNearest()	classcv_1_1Subdiv2D.html	Finds the subdivision vertex closest to the given point.	pt : Input point. nearestPt : Output subdivision vertex point.	The function is another function that locates the input point within the subdivision. It finds the subdivision vertex that is the closest to the input point. It is not necessarily one of vertices of the facet containing the input point, though the facet (located using locate()) is used as a starting point.	2									
imgproc_Subdiv2D_getEdgeList	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getEdgeList(cv::Subdiv2D *obj, std::vector<cv::Vec4f> *edgeList) { BEGIN_WRAP obj->getEdgeList(*edgeList); END_WRAP }	cv::Subdiv2D::getEdgeList()	classcv_1_1Subdiv2D.html	Returns a list of all edges.	edgeList : Output vector.	The function gives each edge as a 4 numbers vector, where each two are one of the edge vertices. i.e. org_x = v[0], org_y = v[1], dst_x = v[2], dst_y = v[3].	1									
imgproc_Subdiv2D_getLeadingEdgeList	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getLeadingEdgeList(cv::Subdiv2D *obj, std::vector<int> *leadingEdgeList) { BEGIN_WRAP obj->getLeadingEdgeList(*leadingEdgeList); END_WRAP }	cv::Subdiv2D::getLeadingEdgeList()	classcv_1_1Subdiv2D.html	Returns a list of the leading edge ID connected to each triangle.	leadingEdgeList : Output vector.	The function gives one edge ID for each triangle.	1									
imgproc_Subdiv2D_getTriangleList	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getTriangleList(cv::Subdiv2D *obj, std::vector<cv::Vec6f> *triangleList) { BEGIN_WRAP obj->getTriangleList(*triangleList); END_WRAP }	cv::Subdiv2D::getTriangleList()	classcv_1_1Subdiv2D.html	Returns a list of all triangles.	triangleList : Output vector.	The function gives each triangle as a 6 numbers vector, where each two are one of the triangle vertices. i.e. p1_x = v[0], p1_y = v[1], p2_x = v[2], p2_y = v[3], p3_x = v[4], p3_y = v[5].	1									
imgproc_Subdiv2D_getVoronoiFacetList	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getVoronoiFacetList(cv::Subdiv2D *obj, int *idx, int idxCount, std::vector<std::vector<cv::Point2f> > *facetList, std::vector<cv::Point2f> *facetCenters) { BEGIN_WRAP std::vector<int> idxVec; if (idx != nullptr) idxVec = std::vector<int>(idx, idx + idxCount); obj->getVoronoiFacetList(idxVec, *facetList, *facetCenters); END_WRAP }	cv::Subdiv2D::getVoronoiFacetList()	classcv_1_1Subdiv2D.html	Returns a list of all Voronoi facets.	idx : Vector of vertices IDs to consider. For all vertices you can pass empty vector. facetList : Output vector of the Voronoi facets. facetCenters : Output vector of the Voronoi facets center points.		3									
imgproc_Subdiv2D_getVertex	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getVertex(cv::Subdiv2D *obj, int vertex, int* firstEdge, MyCvPoint2D32f *returnValue) { BEGIN_WRAP *returnValue = c(obj->getVertex(vertex, firstEdge)); END_WRAP }	cv::Subdiv2D::getVertex()	classcv_1_1Subdiv2D.html	Returns vertex location from vertex ID.	vertex : vertex ID. firstEdge : Optional. The first edge ID which is connected to the vertex.		2									
imgproc_Subdiv2D_getEdge	CVAPI(ExceptionStatus) imgproc_Subdiv2D_getEdgeList(cv::Subdiv2D *obj, std::vector<cv::Vec4f> *edgeList) { BEGIN_WRAP obj->getEdgeList(*edgeList); END_WRAP }	cv::Subdiv2D::getEdgeList()	classcv_1_1Subdiv2D.html	Returns a list of all edges.	edgeList : Output vector.	The function gives each edge as a 4 numbers vector, where each two are one of the edge vertices. i.e. org_x = v[0], org_y = v[1], dst_x = v[2], dst_y = v[3].	1									

imgproc_Subdiv2D_nextEdge	CVAPI(ExceptionStatus) imgproc_Subdiv2D_nextEdge(cv::Subdiv2D *obj, int edge, int *returnValue) { BEGIN_WRAP *returnValue = obj->nextEdge(edge); END_WRAP }	cv::Subdiv2D::nextEdge()	classcv_1_1Subdiv2D.html	Returns next edge around the edge origin.	edge : Subdivision edge ID.		1									
imgproc_Subdiv2D_rotateEdge	CVAPI(ExceptionStatus) imgproc_Subdiv2D_rotateEdge(cv::Subdiv2D *obj, int edge, int rotate, int *returnValue) { BEGIN_WRAP *returnValue = obj->rotateEdge(edge, rotate); END_WRAP }	cv::Subdiv2D::rotateEdge()	classcv_1_1Subdiv2D.html	Returns another edge of the same quad-edge.	edge : Subdivision edge ID. rotate : Parameter specifying which of the edges of the same quad-edge as the input one to return. The following values are possible: 0 - the input edge (e on the picture below if e is the input edge) 1 - the rotated edge (eRot) 2 - the reversed edge (reversed e (in green)) 3 - the reversed rotated edge (reversed eRot (in green))		2								
imgproc_Subdiv2D_symEdge	CVAPI(ExceptionStatus) imgproc_Subdiv2D_symEdge(cv::Subdiv2D *obj, int edge, int *returnValue) { BEGIN_WRAP *returnValue = obj->symEdge(edge); END_WRAP }	cv::Subdiv2D::symEdge()	classcv_1_1Subdiv2D.html				1									
imgproc_Subdiv2D_edgeOrg	CVAPI(ExceptionStatus) imgproc_Subdiv2D_edgeOrg(cv::Subdiv2D *obj, int edge, MyCvPoint2D32f *orgPt, int *returnValue) { BEGIN_WRAP cv::Point2f orgPt0; *returnValue = obj->edgeOrg(edge, &orgPt0); *orgPt = c(orgPt0); END_WRAP }	cv::Subdiv2D::edgeOrg()	classcv_1_1Subdiv2D.html	Returns the edge origin.	edge : Subdivision edge ID. orgpt : Output vertex location.		2									
imgproc_Subdiv2D_edgeDst	CVAPI(ExceptionStatus) imgproc_Subdiv2D_edgeDst(cv::Subdiv2D *obj, int edge, MyCvPoint2D32f *dstPt, int *returnValue) { BEGIN_WRAP cv::Point2f dstPt0; *returnValue = obj->edgeDst(edge, &dstPt0); *dstPt = c(dstPt0); END_WRAP }	cv::Subdiv2D::edgeDst()	classcv_1_1Subdiv2D.html	Returns the edge destination.	edge : Subdivision edge ID. dstpt : Output vertex location.		2									
ml_ANN_MLP_setTrainMethod	CVAPI(ExceptionStatus) ml_ANN_MLP_setTrainMethod(cv::ml::ANN_MLP *obj, int method, double param1, double param2) { BEGIN_WRAP obj->setTrainMethod(method, param1, param2); END_WRAP }	cv::ml::ANN_MLP::setTrainMethod()	classcv_1_1ml_1_1ANN__MLP.html	Sets training method and common parameters.	method : Default value is ANN_MLP::RPROP. See ANN_MLP::TrainingMethods. param1 : passed to setRpropDW0 for ANN_MLP::RPROP and to setBackpropWeightScale for ANN_MLP::BACKPROP and to initialT for ANN_MLP::ANNEAL. param2 : passed to setRpropDWMin for ANN_MLP::RPROP and to setBackpropMomentumScale for ANN_MLP::BACKPROP and to finalT for ANN_MLP::ANNEAL.		3								
ml_ANN_MLP_getTrainMethod	CVAPI(ExceptionStatus) ml_ANN_MLP_getTrainMethod(cv::ml::ANN_MLP *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getTrainMethod(); END_WRAP }	cv::ml::ANN_MLP::getTrainMethod()	classcv_1_1ml_1_1ANN__MLP.html	Returns current training method			1									
ml_ANN_MLP_setActivationFunction	CVAPI(ExceptionStatus) ml_ANN_MLP_setActivationFunction(cv::ml::ANN_MLP *obj, int type, double param1, double param2) { BEGIN_WRAP obj->setActivationFunction(type, param1, param2); END_WRAP }	cv::ml::ANN_MLP::setActivationFunction()	classcv_1_1ml_1_1ANN__MLP.html	Initialize the activation function for each neuron. Currently the default and the only fully supported activation function is ANN_MLP::SIGMOID_SYM.	type : The type of activation function. See ANN_MLP::ActivationFunctions. param1 : The first parameter of the activation function, ¥(¥alpha¥). Default value is 0. param2 : The second parameter of the activation function, ¥(¥beta¥). Default value is 0.		3								
ml_ANN_MLP_setLayerSizes	CVAPI(ExceptionStatus) ml_ANN_MLP_setLayerSizes(cv::ml::ANN_MLP *obj, cv::_InputArray *_layer_sizes) { BEGIN_WRAP obj->setLayerSizes(entity(_layer_sizes)); END_WRAP }	cv::ml::ANN_MLP::setLayerSizes()	classcv_1_1ml_1_1ANN__MLP.html	Integer vector specifying the number of neurons in each layer including the input and output layers. The very first element specifies the number of elements in the input layer. The last element - number of elements in the output layer. Default value is empty Mat.		See alsogetLayerSizes	1							
ml_ANN_MLP_getLayerSizes	CVAPI(ExceptionStatus) ml_ANN_MLP_getLayerSizes(cv::ml::ANN_MLP *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->getLayerSizes()); END_WRAP }	cv::ml::ANN_MLP::getLayerSizes()														
ml_ANN_MLP_getTermCriteria	CVAPI(ExceptionStatus) ml_ANN_MLP_getTermCriteria(cv::ml::ANN_MLP *obj, MyCvTermCriteria *returnValue) { BEGIN_WRAP *returnValue = c(obj->getTermCriteria()); END_WRAP }	cv::ml::ANN_MLP::getTermCriteria()	classcv_1_1ml_1_1ANN__MLP.html	Termination criteria of the training algorithm. You can specify the maximum number of iterations (maxCount) and/or how much the error could change between the iterations to make the algorithm continue (epsilon). Default value is TermCriteria(TermCriteria::MAX_ITER + TermCriteria::EPS, 1000, 0.01).		See alsosetTermCriteria	1							
ml_ANN_MLP_setTermCriteria	CVAPI(ExceptionStatus) ml_ANN_MLP_setTermCriteria(cv::ml::ANN_MLP *obj, MyCvTermCriteria val) { BEGIN_WRAP obj->setTermCriteria(cpp(val)); END_WRAP }	cv::ml::ANN_MLP::setTermCriteria()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetTermCriteria			1									
ml_ANN_MLP_getBackpropWeightScale	CVAPI(ExceptionStatus) ml_ANN_MLP_getBackpropWeightScale(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getBackpropWeightScale(); END_WRAP }	cv::ml::ANN_MLP::getBackpropWeightScale()	classcv_1_1ml_1_1ANN__MLP.html	BPROP: Strength of the weight gradient term. The recommended value is about 0.1. Default value is 0.1.		See alsosetBackpropWeightScale	1							
ml_ANN_MLP_setBackpropWeightScale	CVAPI(ExceptionStatus) ml_ANN_MLP_setBackpropWeightScale(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setBackpropWeightScale(val); END_WRAP }	cv::ml::ANN_MLP::setBackpropWeightScale()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetBackpropWeightScale			1									
ml_ANN_MLP_getBackpropMomentumScale	CVAPI(ExceptionStatus) ml_ANN_MLP_getBackpropMomentumScale(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getBackpropMomentumScale(); END_WRAP }	cv::ml::ANN_MLP::getBackpropMomentumScale()	classcv_1_1ml_1_1ANN__MLP.html	BPROP: Strength of the momentum term (the difference between weights on the 2 previous iterations). This parameter provides some inertia to smooth the random fluctuations of the weights. It can vary from 0 (the feature is disabled) to 1 and beyond. The value 0.1 or so is good enough. Default value is 0.1.		See alsosetBackpropMomentumScale	1							
ml_ANN_MLP_setBackpropMomentumScale	CVAPI(ExceptionStatus) ml_ANN_MLP_setBackpropMomentumScale(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setBackpropMomentumScale(val); END_WRAP }	cv::ml::ANN_MLP::setBackpropMomentumScale()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetBackpropMomentumScale			1									
ml_ANN_MLP_getRpropDW0	CVAPI(ExceptionStatus) ml_ANN_MLP_getRpropDW0(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getRpropDW0(); END_WRAP }	cv::ml::ANN_MLP::getRpropDW0()	classcv_1_1ml_1_1ANN__MLP.html	RPROP: Initial value ¥(¥Delta_0¥) of update-values ¥(¥Delta_{ij}¥). Default value is 0.1.		See alsosetRpropDW0	1							
ml_ANN_MLP_setRpropDW0	CVAPI(ExceptionStatus) ml_ANN_MLP_setRpropDW0(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setRpropDW0(val); END_WRAP }	cv::ml::ANN_MLP::setRpropDW0()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetRpropDW0			1									

ml_ANN_MLP_getRpropDWPlus	CVAPI(ExceptionStatus) ml_ANN_MLP_getRpropDWPlus(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getRpropDWPlus(); END_WRAP }	cv::ml::ANN_MLP::getRpropDWPlus()	classcv_1_1ml_1_1ANN__MLP.html			See alsosetRpropDWPlus	1									
ml_ANN_MLP_setRpropDWPlus	CVAPI(ExceptionStatus) ml_ANN_MLP_setRpropDWPlus(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setRpropDWPlus(val); END_WRAP }	cv::ml::ANN_MLP::setRpropDWPlus()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetRpropDWPlus			1									
ml_ANN_MLP_getRpropDWMinus	CVAPI(ExceptionStatus) ml_ANN_MLP_getRpropDWMinus(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getRpropDWMinus(); END_WRAP }	cv::ml::ANN_MLP::getRpropDWMinus()	classcv_1_1ml_1_1ANN__MLP.html	RPROP: Decrease factor ¥(¥eta^-¥). It must be <1. Default value is 0.5.		See alsosetRpropDWMinus	1									
ml_ANN_MLP_setRpropDWMinus	CVAPI(ExceptionStatus) ml_ANN_MLP_setRpropDWMinus(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setRpropDWMinus(val); END_WRAP }	cv::ml::ANN_MLP::setRpropDWMinus()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetRpropDWMinus			1									
ml_ANN_MLP_getRpropDWMin	CVAPI(ExceptionStatus) ml_ANN_MLP_getRpropDWMinus(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getRpropDWMinus(); END_WRAP }	cv::ml::ANN_MLP::getRpropDWMinus()	classcv_1_1ml_1_1ANN__MLP.html	RPROP: Decrease factor ¥(¥eta^-¥). It must be <1. Default value is 0.5.		See alsosetRpropDWMinus	1									
ml_ANN_MLP_setRpropDWMin	CVAPI(ExceptionStatus) ml_ANN_MLP_setRpropDWMinus(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setRpropDWMinus(val); END_WRAP }	cv::ml::ANN_MLP::setRpropDWMinus()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetRpropDWMinus			1									
ml_ANN_MLP_getRpropDWMax	CVAPI(ExceptionStatus) ml_ANN_MLP_getRpropDWMax(cv::ml::ANN_MLP *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getRpropDWMax(); END_WRAP }	cv::ml::ANN_MLP::getRpropDWMax()	classcv_1_1ml_1_1ANN__MLP.html	RPROP: Update-values upper limit ¥(¥Delta_{max}¥). It must be >1. Default value is 50.		See alsosetRpropDWMax	1									
ml_ANN_MLP_setRpropDWMax	CVAPI(ExceptionStatus) ml_ANN_MLP_setRpropDWMax(cv::ml::ANN_MLP *obj, double val) { BEGIN_WRAP obj->setRpropDWMax(val); END_WRAP }	cv::ml::ANN_MLP::setRpropDWMax()	classcv_1_1ml_1_1ANN__MLP.html	See alsogetRpropDWMax			1									
ml_ANN_MLP_getWeights	CVAPI(ExceptionStatus) ml_ANN_MLP_getWeights(cv::ml::ANN_MLP *obj, int layerIdx, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->getWeights(layerIdx)); END_WRAP }	cv::ml::ANN_MLP::getWeights()	classcv_1_1ml_1_1ANN__MLP.html				1									
ml_ANN_MLP_create	CVAPI(ExceptionStatus) ml_ANN_MLP_create(cv::Ptr<cv::ml::ANN_MLP> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::ANN_MLP::create(); *returnValue = new cv::Ptr<cv::ml::ANN_MLP>(ptr); END_WRAP }	cv::ml::ANN_MLP::create()	classcv_1_1ml_1_1ANN__MLP.html	Creates empty model.		Use StatModel::train to train the model, Algorithm::load<ANN_MLP>(filename) to load the pre-trained model. Note that the train method has optional flags: ANN_MLP::TrainFlags.	1									
ml_Ptr_ANN_MLP_delete	CVAPI(ExceptionStatus) ml_Ptr_ANN_MLP_delete(cv::Ptr<cv::ml::ANN_MLP> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::ANN_MLP	classcv_1_1ml_1_1ANN__MLP.html	Artificial Neural Networks - Multi-Layer Perceptrons. Unlike many other models in ML that are constructed and trained at once, in the MLP model these steps are separated. First, a network with the specified topology is created using the non-default constructor or the method ANN_MLP::create. All the weights are set to zeros. Then, the network is trained using a set of input and output vectors. The training procedure can be repeated more than once, that is, the weights can be adjusted based on the new training data. Additional flags for StatModel::train are available: ANN_MLP::TrainFlags. See alsoNeural Networks			1					•		
ml_Ptr_ANN_MLP_get	CVAPI(ExceptionStatus) ml_Ptr_ANN_MLP_get(cv::Ptr<cv::ml::ANN_MLP> *obj, cv::ml::ANN_MLP **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::ANN_MLP	classcv_1_1ml_1_1ANN__MLP.html	Artificial Neural Networks - Multi-Layer Perceptrons. Unlike many other models in ML that are constructed and trained at once, in the MLP model these steps are separated. First, a network with the specified topology is created using the non-default constructor or the method ANN_MLP::create. All the weights are set to zeros. Then, the network is trained using a set of input and output vectors. The training procedure can be repeated more than once, that is, the weights can be adjusted based on the new training data. Additional flags for StatModel::train are available: ANN_MLP::TrainFlags. See alsoNeural Networks			1						•	
ml_ANN_MLP_load	CVAPI(ExceptionStatus) ml_ANN_MLP_load(const char *filePath, cv::Ptr<cv::ml::ANN_MLP> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::ANN_MLP::load(filePath); *returnValue = new cv::Ptr<cv::ml::ANN_MLP>(ptr); END_WRAP }	cv::ml::ANN_MLP::load()	classcv_1_1ml_1_1ANN__MLP.html	Loads and creates a serialized ANN from a file.	filepath : path to serialized ANN	Use ANN::save to serialize and store an ANN to disk. Load the ANN from this file again, by calling this function with the path to the file.	1									
ml_ANN_MLP_loadFromString	CVAPI(ExceptionStatus) ml_ANN_MLP_loadFromString(const char *strModel, cv::Ptr<cv::ml::ANN_MLP> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::ANN_MLP::create()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromString<cv::ml::ANN_MLP>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::ANN_MLP>(ptr); END_WRAP }	cv::ml::ANN_MLP::create()	classcv_1_1ml_1_1ANN__MLP.html	Creates empty model.		Use StatModel::train to train the model, Algorithm::load<ANN_MLP>(filename) to load the pre-trained model. Note that the train method has optional flags: ANN_MLP::TrainFlags.	2		•							
ml_Boost_getBoostType	CVAPI(ExceptionStatus) ml_Boost_getBoostType(cv::ml::Boost *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getBoostType(); END_WRAP }	cv::ml::Boost::getBoostType()	classcv_1_1ml_1_1Boost.html	Type of the boosting algorithm. See Boost::Types. Default value is Boost::REAL.		See alsosetBoostType	1									
ml_Boost_setBoostType	CVAPI(ExceptionStatus) ml_Boost_setBoostType(cv::ml::Boost *obj, int val) { BEGIN_WRAP obj->setBoostType(val); END_WRAP }	cv::ml::Boost::setBoostType()	classcv_1_1ml_1_1Boost.html	See alsogetBoostType			1									
ml_Boost_getWeakCount	CVAPI(ExceptionStatus) ml_Boost_getWeakCount(cv::ml::Boost *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getWeakCount(); END_WRAP }	cv::ml::Boost::getWeakCount()	classcv_1_1ml_1_1Boost.html	The number of weak classifiers. Default value is 100.		See alsosetWeakCount	1									

ml_Boost_setWeakCount	CVAPI(ExceptionStatus) ml_Boost_setWeakCount(cv::ml::Boost *obj, int val) { BEGIN_WRAP obj->setWeakCount(val); END_WRAP }	cv::ml::Boost::setWeakCount()	classcv_1_1ml_1_1Boost.html	See alsogetWeakCount			1										
ml_Boost_getWeightTrimRate	CVAPI(ExceptionStatus) ml_Boost_getWeightTrimRate(cv::ml::Boost *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getWeightTrimRate(); END_WRAP }	cv::ml::Boost::getWeightTrimRate()	classcv_1_1ml_1_1Boost.html	A threshold between 0 and 1 used to save computational time. Samples with summary weight ¥(¥leq 1 - weight_trim_rate¥) do not participate in the next iteration of training. Set this parameter to 0 to turn off this functionality. Default value is 0.95.		See alsosetWeightTrimRate	1										
ml_Boost_setWeightTrimRate	CVAPI(ExceptionStatus) ml_Boost_setWeightTrimRate(cv::ml::Boost *obj, double val) { BEGIN_WRAP obj->setWeightTrimRate(val); END_WRAP }	cv::ml::Boost::setWeightTrimRate()	classcv_1_1ml_1_1Boost.html	See alsogetWeightTrimRate			1										
ml_Boost_create	CVAPI(ExceptionStatus) ml_Boost_create(cv::Ptr<cv::ml::Boost> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::Boost::create(); *returnValue = new cv::Ptr<cv::ml::Boost>(ptr); END_WRAP }	cv::ml::Boost::create()	classcv_1_1ml_1_1Boost.html	Creates the empty model. Use StatModel::train to train the model, Algorithm::load<Boost>(filename) to load the pre-trained model.			1										
ml_Ptr_Boost_delete	CVAPI(ExceptionStatus) ml_Ptr_Boost_delete(cv::Ptr<cv::ml::Boost> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::Boost	classcv_1_1ml_1_1Boost.html	Boosted tree classifier derived from DTrees. See alsoBoosting									•				
ml_Ptr_Boost_get	CVAPI(ExceptionStatus) ml_Ptr_Boost_get(cv::Ptr<cv::ml::Boost>* obj, cv::ml::Boost **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::Boost	classcv_1_1ml_1_1Boost.html	Boosted tree classifier derived from DTrees. See alsoBoosting										•			
ml_Boost_load	CVAPI(ExceptionStatus) ml_Boost_load(const char *filePath, cv::Ptr<cv::ml::Boost> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml::Boost>(filePath); *returnValue = new cv::Ptr<cv::ml::Boost>(ptr); END_WRAP }	cv::ml::Boost	classcv_1_1ml_1_1Boost.html	Boosted tree classifier derived from DTrees. See alsoBoosting					•								
ml_Boost_loadFromString	CVAPI(ExceptionStatus) ml_Boost_loadFromString(const char *strModel, cv::Ptr<cv::ml::Boost> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::Boost::create()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromString<cv::ml::Boost>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::Boost>(ptr); END_WRAP }	cv::ml::Boost	classcv_1_1ml_1_1Boost.html	Boosted tree classifier derived from DTrees. See alsoBoosting					•								
ml_DTrees_getMaxCategories	CVAPI(ExceptionStatus) ml_DTrees_getMaxCategories(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxCategories(); END_WRAP }	cv::ml::Dtrees::getMaxCategories()																
ml_DTrees_setMaxCategories	CVAPI(ExceptionStatus) ml_DTrees_setMaxCategories(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setMaxCategories(val); END_WRAP }	cv::ml::Dtrees::setMaxCategories()																
ml_DTrees_getMaxDepth	CVAPI(ExceptionStatus) ml_DTrees_getMaxDepth(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxDepth(); END_WRAP }	cv::ml::Dtrees::getMaxDepth()																
ml_DTrees_setMaxDepth	CVAPI(ExceptionStatus) ml_DTrees_setMaxDepth(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setMaxDepth(val); END_WRAP }	cv::ml::Dtrees::setMaxDepth()																
ml_DTrees_getMinSampleCount	CVAPI(ExceptionStatus) ml_DTrees_getMinSampleCount(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMinSampleCount(); END_WRAP }	cv::ml::Dtrees::getMinSampleCount()																
ml_DTrees_setMinSampleCount	CVAPI(ExceptionStatus) ml_DTrees_setMinSampleCount(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setMinSampleCount(val); END_WRAP }	cv::ml::Dtrees::setMinSampleCount()																
ml_DTrees_getCVFolds	CVAPI(ExceptionStatus) ml_DTrees_getCVFolds(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getCVFolds(); END_WRAP }	cv::ml::Dtrees::getCVFolds()																
ml_DTrees_setCVFolds	CVAPI(ExceptionStatus) ml_DTrees_setCVFolds(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setCVFolds(val); END_WRAP }	cv::ml::Dtrees::setCVFolds()																
ml_DTrees_getUseSurrogates	CVAPI(ExceptionStatus) ml_DTrees_getUseSurrogates(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getUseSurrogates() ? 1 : 0; END_WRAP }	cv::ml::Dtrees::getUseSurrogates()																
ml_DTrees_setUseSurrogates	CVAPI(ExceptionStatus) ml_DTrees_setUseSurrogates(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setUseSurrogates(val != 0); END_WRAP }	cv::ml::Dtrees::setUseSurrogates()																
ml_DTrees_getUse1SERule	CVAPI(ExceptionStatus) ml_DTrees_getUse1SERule(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getUse1SERule() ? 1 : 0; END_WRAP }	cv::ml::Dtrees::getUse1SERule()																

ml_DTrees_setUse1SERule	CVAPI(ExceptionStatus) ml_DTrees_setUse1SERule(cv ::ml::DTrees *obj, int val) { BEGIN_WRAP obj->setUse1SERule(val != 0); END_WRAP }	cv::ml::Dtrees::setUse1SERule()															
ml_DTrees_getTruncatePrunedTree	CVAPI(ExceptionStatus) ml_DTrees_getTruncatePrune dTree(cv::ml::DTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getTruncatePrunedTree() ? 1 : 0; END_WRAP }	cv::ml::Dtrees::getTruncatePrunedTree()															
ml_DTrees_setTruncatePrunedTree	CVAPI(ExceptionStatus) ml_DTrees_setTruncatePrune dTree(cv::ml::DTrees *obj, int val) { BEGIN_WRAP obj ->setTruncatePrunedTree(val != 0); END_WRAP }	cv::ml::Dtrees::setTruncatePrunedTree()															
ml_DTrees_getRegressionAccuracy	CVAPI(ExceptionStatus) ml_DTrees_getRegressionAcc uracy(cv::ml::DTrees *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj- >getRegressionAccuracy(); END_WRAP }	cv::ml::Dtrees::getRegressionAccuracy()															
ml_DTrees_setRegressionAccuracy	CVAPI(ExceptionStatus) ml_DTrees_setRegressionAcc uracy(cv::ml::DTrees *obj, float val) { BEGIN_WRAP obj ->setRegressionAccuracy(val); END_WRAP }	cv::ml::Dtrees::setRegressionAccuracy()															
ml_DTrees_getPriors	CVAPI(ExceptionStatus) ml_DTrees_getPriors(cv::ml:: DTrees *obj, cv::Mat **returnValue) { BEGIN_WRAP const auto m = obj- >getPriors(); *returnValue = new cv::Mat(m); END_WRAP }	cv::ml::Dtrees::getPriors()															
ml_DTrees_setPriors	CVAPI(ExceptionStatus) ml_DTrees_setPriors(cv::ml:: DTrees *obj, cv::Mat *val) { BEGIN_WRAP obj->setPriors(*val); END_WRAP }	cv::ml::Dtrees::setPriors()															
ml_DTrees_getRoots	CVAPI(ExceptionStatus) ml_DTrees_getRoots(cv::ml:: DTrees *obj, std::vector<int> *result) { BEGIN_WRAP const auto& org = obj- >getRoots(); result->clear(); for (auto i : org) { result->push_back(i); } END_WRAP }	cv::ml::Dtrees::getRoots()															
ml_DTrees_getNodes	CVAPI(ExceptionStatus) ml_DTrees_getNodes(cv::ml: :DTrees *obj, std::vector<DTrees_Node> *result) { BEGIN_WRAP const auto& org = obj- >getNodes(); result->clear(); for (const auto& i : org) { result->push_back(c(i)); } END_WRAP }	cv::ml::Dtrees::getNodes()															
ml_DTrees_getSplits	CVAPI(ExceptionStatus) ml_DTrees_getSplits(cv::ml:: DTrees *obj, std::vector<DTrees_Split> *result) { BEGIN_WRAP const auto& org = obj- >getSplits(); result->clear(); for (const auto& i : org) { result->push_back(c(i)); } END_WRAP }	cv::ml::Dtrees::getSplits()															
ml_DTrees_getSubsets	CVAPI(ExceptionStatus) ml_DTrees_getSubsets(cv::ml ::DTrees *obj, std::vector<int> *result) { BEGIN_WRAP const auto& org = obj- >getSubsets(); result->clear(); for (auto i : org) { result->push_back(i); } END_WRAP }	cv::ml::Dtrees::getSubsets()															
ml_DTrees_create	CVAPI(ExceptionStatus) ml_DTrees_create(cv::Ptr<cv ::ml::DTrees> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::DTrees::create(); *returnValue = new cv::Ptr<cv::ml::DTrees>(ptr) ; END_WRAP }	cv::ml::Dtrees::create()															
ml_Ptr_DTrees_delete	CVAPI(ExceptionStatus) ml_Ptr_DTrees_delete(cv::Ptr <cv::ml::DTrees> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::Dtrees													•		
ml_Ptr_DTrees_get	CVAPI(ExceptionStatus) ml_Ptr_DTrees_get(cv::Ptr<cc v::ml::DTrees> *obj, cv::ml::DTrees **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::Dtrees														•	
ml_DTrees_load	CVAPI(ExceptionStatus) ml_DTrees_load(const char *filePath, cv::Ptr<cv::ml::DTrees> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml:: DTrees>(filePath); *returnValue = new cv::Ptr<cv::ml::DTrees>(ptr) ; END_WRAP }	cv::ml::Dtrees								•							

ml_DTrees_loadFromString	CVAPI(ExceptionStatus) ml_DTrees_loadFromString(co nst char *strModel, cv::Ptr<cv::ml::DTrees> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::DTrees::create()- >getDefaultName(); const auto ptr = cv::Algorithm::loadFromStrin g<cv::ml::DTrees>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::DTrees>(ptr) ; END_WRAP }	cv::ml::DTrees								•						
ml_EM_getClustersNumber	CVAPI(ExceptionStatus) ml_EM_getClustersNumber(cv ::ml::EM *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getClustersNumber(); END_WRAP }	cv::ml::EM::getClustersNumber()	classcv_1_1ml_1_1EM.html	The number of mixture components in the Gaussian mixture model. Default value of the parameter is EM::DEFAULT_NCLUSTERS=5. Some of EM implementation could determine the optimal number of mixtures within a specified value range, but that is not the case in ML yet.		See alsosetClustersNumber	1									
ml_EM_setClustersNumber	CVAPI(ExceptionStatus) ml_EM_setClustersNumber(cv ::ml::EM *obj, int val) { BEGIN_WRAP obj- >setClustersNumber(val); END_WRAP }	cv::ml::EM::setClustersNumber()	classcv_1_1ml_1_1EM.html	See alsogetClustersNumber			1									
ml_EM_getCovarianceMatrixType	CVAPI(ExceptionStatus) ml_EM_getCovarianceMatrixT ype(cv::ml::EM *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getCovarianceMatrixType(); END_WRAP }	cv::ml::EM::getCovarianceMatrixType()	classcv_1_1ml_1_1EM.html	Constraint on covariance matrices which defines type of matrices. See EM::Types.		See alsosetCovarianceMatrixType	1									
ml_EM_setCovarianceMatrixType	CVAPI(ExceptionStatus) ml_EM_setCovarianceMatrixTy pe(cv::ml::EM *obj, int val) { BEGIN_WRAP obj- >setCovarianceMatrixType(val); END_WRAP }	cv::ml::EM::setCovarianceMatrixType()	classcv_1_1ml_1_1EM.html	See alsogetCovarianceMatrixType			1									
ml_EM_getTermCriteria	CVAPI(ExceptionStatus) ml_EM_getTermCriteria(cv::m l::EM *obj, MyCvTermCriteria *returnValue) { BEGIN_WRAP *returnValue = c(obj- >getTermCriteria()); END_WRAP }	cv::ml::EM::getTermCriteria()	classcv_1_1ml_1_1EM.html	The termination criteria of the EM algorithm. The EM algorithm can be terminated by the number of iterations termCrit.maxCount (number of M-steps) or when relative change of likelihood logarithm is less than termCrit.epsilon. Default maximum number of iterations is EM::DEFAULT_MAX_ITERS=100.		See alsosetTermCriteria	1									
ml_EM_setTermCriteria	CVAPI(ExceptionStatus) ml_EM_setTermCriteria(cv::m l::EM *obj, MyCvTermCriteria val) { BEGIN_WRAP obj- >setTermCriteria(cpp(val)); END_WRAP }	cv::ml::EM::setTermCriteria()	classcv_1_1ml_1_1EM.html	See alsogetTermCriteria			1									
ml_EM_getWeights	CVAPI(ExceptionStatus) ml_EM_getWeights(cv::ml::E M *obj, cv::Mat **returnValue) { BEGIN_WRAP const auto m = obj- >getWeights(); *returnValue = new cv::Mat(m); END_WRAP }	cv::ml::EM::getWeights()	classcv_1_1ml_1_1EM.html	Returns weights of the mixtures.		Returns vector with the number of elements equal to the number of mixtures.	1									
ml_EM_getMeans	CVAPI(ExceptionStatus) ml_EM_getMeans(cv::ml::EM *obj, cv::Mat **returnValue) { BEGIN_WRAP const auto m = obj- >getMeans(); *returnValue = new cv::Mat(m); END_WRAP }	cv::ml::EM::getMeans()	classcv_1_1ml_1_1EM.html	Returns the cluster centers (means of the Gaussian mixture)		Returns matrix with the number of rows equal to the number of mixtures and number of columns equal to the space dimensionality.	1									
ml_EM_getCovs	CVAPI(ExceptionStatus) ml_EM_getCovs(cv::ml::EM *obj, std::vector<cv::Mat* > *covs) { BEGIN_WRAP std::vector<cv::Mat> raw; obj->getCovs(raw); covs->resize(raw.size()); for (size_t i = 0; i < raw.size(); i++) { covs->at(i) = new cv::Mat(raw[i]); } END_WRAP }	cv::ml::EM::getCovs()	classcv_1_1ml_1_1EM.html	Returns covariation matrices.		Returns vector of covariation matrices. Number of matrices is the number of gaussian mixtures, each matrix is a square floating-point matrix NxN, where N is the space dimensionality.	1									
ml_EM_predict2	CVAPI(ExceptionStatus) ml_EM_predict2(cv::ml::EM *obj, cv::_InputArray *sample, cv::_OutputArray *probs, CvVec2d *returnValue) { BEGIN_WRAP auto vec = obj- >predict2(*sample, *probs); CvVec2d ret; ret.val[0] = vec[0]; ret.val[1] = vec[1]; *returnValue = ret; END_WRAP }	cv::ml::EM::predict2()	classcv_1_1ml_1_1EM.html	Returns a likelihood logarithm value and an index of the most probable mixture component for the given sample.	sample : A sample for classification. It should be a one-channel matrix of ¥(1 ¥times dims¥) or ¥(dims ¥times 1¥) size. probs : Optional output matrix that contains posterior probabilities of each component given the sample. It has ¥(1 ¥times nclusters¥) size and CV_64FC1 type.	The method returns a two-element double vector. Zero element is a likelihood logarithm value for the sample. First element is an index of the most probable mixture component for the given sample.	2									
ml_EM_trainEM	CVAPI(ExceptionStatus) ml_EM_trainEM(cv::ml::EM *obj, cv::_InputArray *samples, cv::_OutputArray *logLikelihoods, cv::_OutputArray *labels, cv::_OutputArray *probs, int *returnValue) { BEGIN_WRAP const auto ret = obj- >trainEM(*samples, entity(logLikelihoods), entity(labels), entity(probs)); *returnValue = ret ? 1 : 0; END_WRAP }	cv::ml::EM::trainEM()	classcv_1_1ml_1_1EM.html	Estimate the Gaussian mixture parameters from a samples set.	samples : Samples from which the Gaussian mixture model will be estimated. It should be a one-channel matrix, each row of which is a sample. If the matrix does not have CV_64F type it will be converted to the inner matrix of such type for the further computing. logLikelihoods : The optional output matrix that contains a likelihood logarithm value for each sample. It has ¥(nsamples ¥times 1¥) size and CV_64FC1 type. labels : The optional output "class label" for each sample: ¥(¥texttt{labels}_i=¥texttt{arg max}_k(p_{i,k}), i=1..N¥) (indices of the most probable mixture component for each sample). It has ¥(nsamples ¥times 1¥) size and CV_32SC1 type. probs : The optional output matrix that contains posterior probabilities of each Gaussian mixture component given the each sample. It has ¥(nsamples ¥times nclusters¥) size and CV_64FC1 type.	This variation starts with Expectation step. Initial values of the model parameters will be estimated by the k-means algorithm.Unlike many of the ML models, EM is an unsupervised learning algorithm and it does not take responses (class labels or function values) as input. Instead, it computes the Maximum Likelihood Estimate of the Gaussian mixture parameters from an input sample set, stores all the parameters inside the structure: ¥(p_{i,k}¥) in probs, ¥(a_k¥) in means , ¥(S_k¥) in covs[k], ¥(¥pi_k¥) in weights , and optionally computes the output "class label" for each sample: ¥(¥texttt{labels}_i=¥texttt{arg max}_k(p_{i,k}), i=1..N¥) (indices of the most probable mixture component for each sample).The trained model can be used further for prediction, just like any other classifier. The trained model is similar to the NormalBayesClassifier.	4									
ml_EM_trainE	CVAPI(ExceptionStatus) ml_EM_trainEM(cv::ml::EM *obj, cv::_InputArray *samples, cv::_OutputArray *logLikelihoods, cv::_OutputArray *labels, cv::_OutputArray *probs, int *returnValue) { BEGIN_WRAP const auto ret = obj- >trainEM(*samples, entity(logLikelihoods), entity(labels), entity(probs)); *returnValue = ret ? 1 : 0; END_WRAP }	cv::ml::EM::trainEM()	classcv_1_1ml_1_1EM.html	Estimate the Gaussian mixture parameters from a samples set.	samples : Samples from which the Gaussian mixture model will be estimated. It should be a one-channel matrix, each row of which is a sample. If the matrix does not have CV_64F type it will be converted to the inner matrix of such type for the further computing. logLikelihoods : The optional output matrix that contains a likelihood logarithm value for each sample. It has ¥(nsamples ¥times 1¥) size and CV_64FC1 type. labels : The optional output "class label" for each sample: ¥(¥texttt{labels}_i=¥texttt{arg max}_k(p_{i,k}), i=1..N¥) (indices of the most probable mixture component for each sample). It has ¥(nsamples ¥times 1¥) size and CV_32SC1 type. probs : The optional output matrix that contains posterior probabilities of each Gaussian mixture component given the each sample. It has ¥(nsamples ¥times nclusters¥) size and CV_64FC1 type.	This variation starts with Expectation step. Initial values of the model parameters will be estimated by the k-means algorithm.Unlike many of the ML models, EM is an unsupervised learning algorithm and it does not take responses (class labels or function values) as input. Instead, it computes the Maximum Likelihood Estimate of the Gaussian mixture parameters from an input sample set, stores all the parameters inside the structure: ¥(p_{i,k}¥) in probs, ¥(a_k¥) in means , ¥(S_k¥) in covs[k], ¥(¥pi_k¥) in weights , and optionally computes the output "class label" for each sample: ¥(¥texttt{labels}_i=¥texttt{arg max}_k(p_{i,k}), i=1..N¥) (indices of the most probable mixture component for each sample).The trained model can be used further for prediction, just like any other classifier. The trained model is similar to the NormalBayesClassifier.	4									
ml_EM_trainM	CVAPI(ExceptionStatus) ml_EM_trainM(cv::ml::EM *obj, cv::_InputArray *samples, cv::_InputArray *probs0, cv::_OutputArray *logLikelihoods, cv::_OutputArray *labels, cv::_OutputArray *probs, int *returnValue) { BEGIN_WRAP const auto ret = obj- >trainM(*samples, *probs0, entity(logLikelihoods), entity(labels), entity(probs)); *returnValue = ret ? 1 : 0; END_WRAP }	cv::ml::EM::trainM()	classcv_1_1ml_1_1EM.html	Estimate the Gaussian mixture parameters from a samples set.	samples : Samples from which the Gaussian mixture model will be estimated. It should be a one-channel matrix, each row of which is a sample. If the matrix does not have CV_64F type it will be converted to the inner matrix of such type for the further computing. probs0 : the probabilities logLikelihoods : The optional output matrix that contains a likelihood logarithm value for each sample. It has ¥(nsamples ¥times 1¥) size and CV_64FC1 type. labels : The optional output "class label" for each sample: ¥(¥texttt{labels}_i=¥texttt{arg max}_k(p_{i,k}), i=1..N¥) (indices of the most probable mixture component for each sample). It has ¥(nsamples ¥times 1¥) size and CV_32SC1 type. probs : The optional output matrix that contains posterior probabilities of each Gaussian mixture component given the each sample. It has ¥(nsamples ¥times nclusters¥) size and CV_64FC1 type.	This variation starts with Maximization step. You need to provide initial probabilities ¥(p_{i,k}¥) to use this option.	5									

ml_EM_create	CVAPI(ExceptionStatus) ml_EM_create(cv::Ptr<cv::ml ::EM> **returnValue) { BEGIN_WRAP const auto obj = cv::ml::EM::create(); *returnValue = new cv::Ptr<cv::ml::EM>(obj); END_WRAP }	cv::ml::EM::create()	classcv_1_1ml_1_1EM.html	Creates empty EM model. The model should be trained then using StatModel::train(traindata, flags) method. Alternatively, you can use one of the EM::train* methods or load it from file using Algorithm::load<EM>(filename).			1									
ml_Ptr_EM_get	CVAPI(ExceptionStatus) ml_Ptr_EM_get(cv::Ptr<cv::m l::EM> *obj, cv::ml::EM **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::EM	classcv_1_1ml_1_1EM.html	The class implements the Expectation Maximization algorithm. See alsoExpectation Maximization											•	
ml_Ptr_EM_delete	CVAPI(ExceptionStatus) ml_Ptr_EM_delete(cv::Ptr<cv ::ml::EM> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::EM	classcv_1_1ml_1_1EM.html	The class implements the Expectation Maximization algorithm. See alsoExpectation Maximization										•		
ml_EM_load	CVAPI(ExceptionStatus) ml_EM_load(const char *filePath, cv::Ptr<cv::ml::EM> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml:: EM>(filePath); *returnValue = new cv::Ptr<cv::ml::EM>(ptr); END_WRAP }	cv::ml::EM	classcv_1_1ml_1_1EM.html	The class implements the Expectation Maximization algorithm. See alsoExpectation Maximization						•						
ml_EM_loadFromString	CVAPI(ExceptionStatus) ml_EM_loadFromString(const char *strModel, cv::Ptr<cv::ml::EM> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::EM::create()- >getDefaultName(); const auto ptr = cv::Algorithm::loadFromStrin g<cv::ml::EM>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::EM>(ptr); END_WRAP }	cv::ml::EM	classcv_1_1ml_1_1EM.html	The class implements the Expectation Maximization algorithm. See alsoExpectation Maximization						•						
ml_KNearest_getDefaultK	CVAPI(ExceptionStatus) ml_KNearest_getDefaultK(cv: :ml::KNearest *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getDefaultK(); END_WRAP }	cv::ml::Knearest::getDefaultK()														
ml_KNearest_setDefaultK	CVAPI(ExceptionStatus) ml_KNearest_setDefaultK(cv:: ml::KNearest *obj, int val) { BEGIN_WRAP obj->setDefaultK(val); END_WRAP }	cv::ml::Knearest::setDefaultK()														
ml_KNearest_getIsClassifier	CVAPI(ExceptionStatus) ml_KNearest_getIsClassifier(c v::ml::KNearest *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getIsClassifier() ? 1 : 0; END_WRAP }	cv::ml::Knearest::getIsClassifier()														
ml_KNearest_setIsClassifier	CVAPI(ExceptionStatus) ml_KNearest_setIsClassifier(c v::ml::KNearest *obj, int val) { BEGIN_WRAP obj->setIsClassifier(val != 0); END_WRAP }	cv::ml::Knearest::setIsClassifier()														
ml_KNearest_getEmax	CVAPI(ExceptionStatus) ml_KNearest_getEmax(cv::ml ::KNearest *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getEmax(); END_WRAP }	cv::ml::Knearest::getEmax()														
ml_KNearest_setEmax	CVAPI(ExceptionStatus) ml_KNearest_setEmax(cv::ml ::KNearest *obj, int val) { BEGIN_WRAP obj->setEmax(val); END_WRAP }	cv::ml::Knearest::setEmax()														
ml_KNearest_getAlgorithmType	CVAPI(ExceptionStatus) ml_KNearest_getAlgorithmTy pe(cv::ml::KNearest *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getAlgorithmType(); END_WRAP }	cv::ml::Knearest::getAlgorithmType()														
ml_KNearest_setAlgorithmType	CVAPI(ExceptionStatus) ml_KNearest_setAlgorithmTyp e(cv::ml::KNearest *obj, int val) { BEGIN_WRAP obj- >setAlgorithmType(val); END_WRAP }	cv::ml::Knearest::setAlgorithmType()														
ml_KNearest_findNearest	CVAPI(ExceptionStatus) ml_KNearest_findNearest(cv:: ml::KNearest *obj, cv::_InputArray *samples, int k, cv::_OutputArray *results, cv::_OutputArray *neighborResponses, cv::_OutputArray *dist, float *returnValue) { BEGIN_WRAP *returnValue = obj- >findNearest(entity(samples), k, entity(results), entity(neighborResponses), entity(dist)); END_WRAP }	cv::ml::Knearest::findNearest()														
ml_KNearest_create	CVAPI(ExceptionStatus) ml_KNearest_create(cv::Ptr< cv::ml::KNearest> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::KNearest::create(); *returnValue = new cv::Ptr<cv::ml::KNearest>(pt r); END_WRAP }	cv::ml::Knearest::create()														
ml_Ptr_KNearest_delete	CVAPI(ExceptionStatus) ml_Ptr_KNearest_delete(cv::P tr<cv::ml::KNearest> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::Knearest													•	
ml_Ptr_KNearest_get	CVAPI(ExceptionStatus) ml_Ptr_KNearest_get(cv::Ptr <cv::ml::KNearest> * obj, cv::ml::KNearest **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::Knearest														

ml_KNearest_load	CVAPI(ExceptionStatus) ml_KNearest_load(const char *filePath, cv::Ptr<cv::ml::KNearest> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml::KNearest>(filePath); *returnValue = new cv::Ptr<cv::ml::KNearest>(ptr); END_WRAP }	cv::ml::Knearest								•							
ml_KNearest_loadFromString	CVAPI(ExceptionStatus) ml_KNearest_loadFromString(const char *strModel, cv::Ptr<cv::ml::KNearest> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::KNearest::create()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromString<cv::ml::KNearest>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::KNearest>(ptr); END_WRAP }	cv::ml::Knearest								•							
ml_LogisticRegression_getLearningRate	CVAPI(ExceptionStatus) ml_LogisticRegression_getLearningRate(cv::ml::LogisticRegression *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getLearningRate(); END_WRAP }	cv::ml::LogisticRegression::getLearningRate()	classcv_1_1ml_1_1LogisticRegression.html	Learning rate.		See alsosetLearningRate	1									
ml_LogisticRegression_setLearningRate	CVAPI(ExceptionStatus) ml_LogisticRegression_setLearningRate(cv::ml::LogisticRegression *obj, double val) { BEGIN_WRAP obj->setLearningRate(val); END_WRAP }	cv::ml::LogisticRegression::setLearningRate()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetLearningRate			1									
ml_LogisticRegression_getIterations	CVAPI(ExceptionStatus) ml_LogisticRegression_getIterations(cv::ml::LogisticRegression *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::ml::LogisticRegression::getIterations()	classcv_1_1ml_1_1LogisticRegression.html	Number of iterations.		See alsosetIterations	1									
ml_LogisticRegression_setIterations	CVAPI(ExceptionStatus) ml_LogisticRegression_setIterations(cv::ml::LogisticRegression *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::ml::LogisticRegression::setIterations()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetIterations			1									
ml_LogisticRegression_getRegularization	CVAPI(ExceptionStatus) ml_LogisticRegression_getRegularization(cv::ml::LogisticRegression *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getRegularization(); END_WRAP }	cv::ml::LogisticRegression::getRegularization()	classcv_1_1ml_1_1LogisticRegression.html	Kind of regularization to be applied. See LogisticRegression::RegKinds.		See alsosetRegularization	1									
ml_LogisticRegression_setRegularization	CVAPI(ExceptionStatus) ml_LogisticRegression_setRegularization(cv::ml::LogisticRegression *obj, int val) { BEGIN_WRAP obj->setRegularization(val); END_WRAP }	cv::ml::LogisticRegression::setRegularization()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetRegularization			1									
ml_LogisticRegression_getTrainMethod	CVAPI(ExceptionStatus) ml_LogisticRegression_getTrainMethod(cv::ml::LogisticRegression *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getTrainMethod(); END_WRAP }	cv::ml::LogisticRegression::getTrainMethod()	classcv_1_1ml_1_1LogisticRegression.html	Kind of training method used. See LogisticRegression::Methods.		See alsosetTrainMethod	1									
ml_LogisticRegression_setTrainMethod	CVAPI(ExceptionStatus) ml_LogisticRegression_setTrainMethod(cv::ml::LogisticRegression *obj, int val) { BEGIN_WRAP obj->setTrainMethod(val); END_WRAP }	cv::ml::LogisticRegression::setTrainMethod()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetTrainMethod			1									
ml_LogisticRegression_getMiniBatchSize	CVAPI(ExceptionStatus) ml_LogisticRegression_getMiniBatchSize(cv::ml::LogisticRegression *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMiniBatchSize(); END_WRAP }	cv::ml::LogisticRegression::getMiniBatchSize()	classcv_1_1ml_1_1LogisticRegression.html	Specifies the number of training samples taken in each step of Mini-Batch Gradient Descent. Will only be used if using LogisticRegression::MINI_BATCH training algorithm. It has to take values less than the total number of training samples.		See alsosetMiniBatchSize	1									
ml_LogisticRegression_setMiniBatchSize	CVAPI(ExceptionStatus) ml_LogisticRegression_setMiniBatchSize(cv::ml::LogisticRegression *obj, int val) { BEGIN_WRAP obj->setMiniBatchSize(val); END_WRAP }	cv::ml::LogisticRegression::setMiniBatchSize()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetMiniBatchSize			1									
ml_LogisticRegression_getTermCriteria	CVAPI(ExceptionStatus) ml_LogisticRegression_getTermCriteria(cv::ml::LogisticRegression *obj, MyCvTermCriteria *returnValue) { BEGIN_WRAP *returnValue = c(obj->getTermCriteria()); END_WRAP }	cv::ml::LogisticRegression::getTermCriteria()	classcv_1_1ml_1_1LogisticRegression.html	Termination criteria of the algorithm.		See alsosetTermCriteria	1									
ml_LogisticRegression_setTermCriteria	CVAPI(ExceptionStatus) ml_LogisticRegression_setTermCriteria(cv::ml::LogisticRegression *obj, MyCvTermCriteria val) { BEGIN_WRAP obj->setTermCriteria(cpp(val)); END_WRAP }	cv::ml::LogisticRegression::setTermCriteria()	classcv_1_1ml_1_1LogisticRegression.html	See alsogetTermCriteria			1									
ml_LogisticRegression_predict	CVAPI(ExceptionStatus) ml_LogisticRegression_predict (cv::ml::LogisticRegression *obj, cv::_InputArray *samples, cv::_OutputArray *results, int flags, float *returnValue) { BEGIN_WRAP *returnValue = obj->predict(entity(samples), entity(results), flags); END_WRAP }	cv::ml::LogisticRegression::predict()	classcv_1_1ml_1_1LogisticRegression.html	Predicts responses for input samples and returns a float type.	samples : The input data for the prediction algorithm. Matrix [m x n], where each row contains variables (features) of one object being classified. Should have data type CV_32F. results : Predicted labels as a column matrix of type CV_32S. flags : Not used.	Implements cv::ml::StatModel.	3									
ml_LogisticRegression_get_learnt_thetas	CVAPI(ExceptionStatus) ml_LogisticRegression_get_learnt_thetas(cv::ml::LogisticRegression *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->get_learnt_thetas()); END_WRAP }	cv::ml::LogisticRegression::get_learnt_thetas()	classcv_1_1ml_1_1LogisticRegression.html	This function returns the trained parameters arranged across rows.		For a two class classification problem, it returns a row matrix. It returns learnt parameters of the Logistic Regression as a matrix of type CV_32F.	1									

ml_LogisticRegression_create	CVAPI(ExceptionStatus) ml_LogisticRegression_create(cv::Ptr<cv::ml::LogisticRegre ssion> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::LogisticRegression::cr eate(); *returnValue = new cv::Ptr<cv::ml::LogisticRegre ssion>(ptr); END_WRAP }	cv::ml::LogisticRegression::create()	classcv_1_1ml_1_1LogisticRegressi on.html	Creates empty model.			Creates Logistic Regression model with parameters given.	1									
ml_Ptr_LogisticRegression_delete	CVAPI(ExceptionStatus) ml_Ptr_LogisticRegression_del ete(cv::Ptr<cv::ml::LogisticR egression> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::LogisticRegression	classcv_1_1ml_1_1LogisticRegressi on.html	Implements Logistic Regression classifier. See alsoLogistic Regression									•			
ml_Ptr_LogisticRegression_get	CVAPI(ExceptionStatus) ml_Ptr_LogisticRegression_get (cv::Ptr<cv::ml::LogisticRegre ssion> *obj, cv::ml::LogisticRegression **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::LogisticRegression	classcv_1_1ml_1_1LogisticRegressi on.html	Implements Logistic Regression classifier. See alsoLogistic Regression										•		
ml_LogisticRegression_load	CVAPI(ExceptionStatus) ml_LogisticRegression_load(const char *filePath, cv::Ptr<cv::ml::LogisticRegre ssion> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml:: LogisticRegression>(filePath); *returnValue = new cv::Ptr<cv::ml::LogisticRegre ssion>(ptr); END_WRAP }	cv::ml::LogisticRegression	classcv_1_1ml_1_1LogisticRegressi on.html	Implements Logistic Regression classifier. See alsoLogistic Regression							•					
ml_LogisticRegression_loadFromString	CVAPI(ExceptionStatus) ml_LogisticRegression_loadFro mString(const char *strModel, cv::Ptr<cv::ml::LogisticRegre ssion> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::LogisticRegression::cr eate()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromStrin g<cv::ml::LogisticRegression >(strModel, objName); *returnValue = new cv::Ptr<cv::ml::LogisticRegre ssion>(ptr); END_WRAP }	cv::ml::LogisticRegression	classcv_1_1ml_1_1LogisticRegressi on.html	Implements Logistic Regression classifier. See alsoLogistic Regression							•					
ml_NormalBayesClassifier_predictProb	CVAPI(ExceptionStatus) ml_NormalBayesClassifier_pre dictProb(cv::ml::NormalBayesClassifier *obj, cv::_InputArray *inputs, cv::_OutputArray *samples, cv::_OutputArray *outputProbs, int flags, float *returnValue) { BEGIN_WRAP *returnValue = obj- >predictProb(entity(inputs), entity(samples), entity(outputProbs), flags); END_WRAP }	cv::ml::NormalBayesClassifier::predictProb()	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Predicts the response for sample(s).			The method estimates the most probable classes for input vectors. Input vectors (one or more) are stored as rows of the matrix inputs. In case of multiple input vectors, there should be one output vector outputs. The predicted class for a single input vector is returned by the method. The vector outputProbs contains the output probabilities corresponding to each element of result.	4								
ml_NormalBayesClassifier_create	CVAPI(ExceptionStatus) ml_NormalBayesClassifier_cre ate(cv::Ptr<cv::ml::NormalB ayesClassifier> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::NormalBayesClassifier ::create(); *returnValue = new cv::Ptr<cv::ml::NormalBayes Classifier>(ptr); END_WRAP }	cv::ml::NormalBayesClassifier::create()	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Creates empty model Use StatModel::train to train the model after creation.				1								
ml_Ptr_NormalBayesClassifier_delete	CVAPI(ExceptionStatus) ml_Ptr_NormalBayesClassifier _delete(cv::Ptr<cv::ml::Norm alBayesClassifier> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::NormalBayesClassifier	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Bayes classifier for normally distributed data. See alsoNormal Bayes Classifier									•			
ml_Ptr_NormalBayesClassifier_get	CVAPI(ExceptionStatus) ml_Ptr_NormalBayesClassifier _get(cv::Ptr<cv::ml::NormalBayes Classifier>* obj, cv::ml::NormalBayesClassifier **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::NormalBayesClassifier	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Bayes classifier for normally distributed data. See alsoNormal Bayes Classifier										•		
ml_NormalBayesClassifier_load	CVAPI(ExceptionStatus) ml_NormalBayesClassifier_loa d(const char *filePath, cv::Ptr<cv::ml::NormalBayes Classifier> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml:: NormalBayesClassifier>(filePa th); *returnValue = new cv::Ptr<cv::ml::NormalBayes Classifier>(ptr); END_WRAP }	cv::ml::NormalBayesClassifier	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Bayes classifier for normally distributed data. See alsoNormal Bayes Classifier							•					
ml_NormalBayesClassifier_loadFromString	CVAPI(ExceptionStatus) ml_NormalBayesClassifier_loa dFromString(const char *strModel, cv::Ptr<cv::ml::NormalBayes Classifier> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::NormalBayesClassifier ::create() ->getDefaultName(); const auto ptr = cv::Algorithm::loadFromStrin g<cv::ml::NormalBayesClassif ier>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::NormalBayes Classifier>(ptr); END_WRAP }	cv::ml::NormalBayesClassifier	classcv_1_1ml_1_1NormalBayesCla ssifier.html	Bayes classifier for normally distributed data. See alsoNormal Bayes Classifier							•					
ml_RTrees_getCalculateVarImportance	CVAPI(ExceptionStatus) ml_RTrees_getCalculateVarIm portance(cv::ml::RTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getCalculateVarImportance( ) ? 1 : 0; END_WRAP }	cv::ml::Rtrees::getCalculateVarImportance()														
ml_RTrees_setCalculateVarImportance	CVAPI(ExceptionStatus) ml_RTrees_setCalculateVarIm portance(cv::ml::RTrees *obj, int val) { BEGIN_WRAP obj ->setCalculateVarImportance(val != 0); END_WRAP }	cv::ml::Rtrees::setCalculateVarImportance()														
ml_RTrees_getActiveVarCount	CVAPI(ExceptionStatus) ml_RTrees_getActiveVarCount (cv::ml::RTrees *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getActiveVarCount(); END_WRAP }	cv::ml::Rtrees::getActiveVarCount()														

ml_RTrees_setActiveVarCount	CVAPI(ExceptionStatus) ml_RTrees_setActiveVarCount (cv::ml::RTrees *obj, int val) { BEGIN_WRAP obj->setActiveVarCount(val); END_WRAP }	cv::ml::Rtrees::setActiveVarCount()														
ml_RTrees_getTermCriteria	CVAPI(ExceptionStatus) ml_RTrees_getTermCriteria(cv::ml::RTrees *obj, MyCvTermCriteria *returnValue) { BEGIN_WRAP *returnValue = c(obj->getTermCriteria()); END_WRAP }	cv::ml::Rtrees::getTermCriteria()														
ml_RTrees_setTermCriteria	CVAPI(ExceptionStatus) ml_RTrees_setTermCriteria(cv::ml::RTrees *obj, MyCvTermCriteria val) { BEGIN_WRAP obj->setTermCriteria(cpp(val)); END_WRAP }	cv::ml::Rtrees::setTermCriteria()														
ml_RTrees_getVarImportance	CVAPI(ExceptionStatus) ml_RTrees_getVarImportance(cv::ml::RTrees *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->getVarImportance()); END_WRAP }	cv::ml::Rtrees::getVarImportance()														
ml_RTrees_create	CVAPI(ExceptionStatus) ml_RTrees_create(cv::Ptr<cv::ml::RTrees> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::RTrees::create(); *returnValue = new cv::Ptr<cv::ml::RTrees>(ptr); END_WRAP }	cv::ml::Rtrees::create()														
ml_Ptr_RTrees_delete	CVAPI(ExceptionStatus) ml_Ptr_RTrees_delete(cv::Ptr<cv::ml::RTrees> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::Rtrees												•		
ml_Ptr_RTrees_get	CVAPI(ExceptionStatus) ml_Ptr_RTrees_get(cv::Ptr<cv::ml::RTrees> *obj, cv::ml::RTrees **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::Rtrees													•	
ml_RTrees_load	CVAPI(ExceptionStatus) ml_RTrees_load(const char *filePath, cv::Ptr<cv::ml::RTrees> **returnValue) { BEGIN_WRAP const auto ptr = cv::Algorithm::load<cv::ml::RTrees>(filePath); *returnValue = new cv::Ptr<cv::ml::RTrees>(ptr); END_WRAP }	cv::ml::Rtrees								•						
ml_RTrees_loadFromString	CVAPI(ExceptionStatus) ml_RTrees_loadFromString(const char *strModel, cv::Ptr<cv::ml::RTrees> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::RTrees::create()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromString<cv::ml::RTrees>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::RTrees>(ptr); END_WRAP }	cv::ml::Rtrees								•						
ml_StatModel_clear	CVAPI(ExceptionStatus) ml_StatModel_clear(cv::ml::StatModel *obj) { BEGIN_WRAP obj->clear(); END_WRAP }	cv::ml::StatModel::clear()														
ml_StatModel_getVarCount	CVAPI(ExceptionStatus) ml_StatModel_getVarCount(cv::ml::StatModel *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getVarCount(); END_WRAP }	cv::ml::StatModel::getVarCount()	classcv_1_1ml_1_1StatModel.html	Returns the number of variables in training samples.			1									
ml_StatModel_empty	CVAPI(ExceptionStatus) ml_StatModel_empty(cv::ml::StatModel *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::ml::StatModel::empty()	classcv_1_1ml_1_1StatModel.html	Returns true if the Algorithm is empty (e.g. in the very beginning or after unsuccessful read.		Reimplemented from cv::Algorithm.	1									
ml_StatModel_isTrained	CVAPI(ExceptionStatus) ml_StatModel_isTrained(cv::ml::StatModel *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isTrained() ? 1 : 0; END_WRAP }	cv::ml::StatModel::isTrained()	classcv_1_1ml_1_1StatModel.html	Returns true if the model is trained.			1									
ml_StatModel_isClassifier	CVAPI(ExceptionStatus) ml_StatModel_isClassifier(cv::ml::StatModel *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isClassifier() ? 1 : 0; END_WRAP }	cv::ml::StatModel::isClassifier()	classcv_1_1ml_1_1StatModel.html	Returns true if the model is classifier.			1									
ml_StatModel_train2	CVAPI(ExceptionStatus) ml_StatModel_train2(cv::ml::StatModel *obj, cv::_InputArray *samples, int layout, cv::_InputArray *responses, int *returnValue) { BEGIN_WRAP *returnValue = obj->train(*samples, layout, *responses) ? 1 : 0; END_WRAP }	cv::ml::StatModel::train()	classcv_1_1ml_1_1StatModel.html	Trains the statistical model.	samples : training samples layout : See ml::SampleTypes. responses : vector of responses associated with the training samples.		3									
ml_StatModel_predict	CVAPI(ExceptionStatus) ml_StatModel_predict(cv::ml::StatModel *obj, cv::_InputArray *samples, cv::_OutputArray *results, int flags, float *returnValue) { BEGIN_WRAP *returnValue = obj->predict(entity(samples), entity(results), flags); END_WRAP }	cv::ml::StatModel::predict()	classcv_1_1ml_1_1StatModel.html	Predicts response(s) for the provided sample(s)	samples : The input samples, floating-point matrix results : The optional output matrix of results. flags : The optional flags, model-dependent. See cv::ml::StatModel::Flags.	Implemented in cv::ml::LogisticRegression, and cv::ml::EM.	3									
ml_SVM_getType	CVAPI(ExceptionStatus) ml_SVM_getType(cv::ml::SVM *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getType(); END_WRAP }	cv::ml::SVM::getType()	classcv_1_1ml_1_1SVM.html	Type of a SVM formulation. See SVM::Types. Default value is SVM::C_SVC.		See alsosetType	1									

ml_SVM_setType	CVAPI(ExceptionStatus) ml_SVM_setType(cv::ml::SVM *obj, int val) { BEGIN_WRAP obj->setType(val); END_WRAP }	cv::ml::SVM::setType()	classcv_1_1ml_1_1SVM.html	See alsogetType			1										
ml_SVM_getGamma	CVAPI(ExceptionStatus) ml_SVM_getGamma(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getGamma(); END_WRAP }	cv::ml::SVM::getGamma()	classcv_1_1ml_1_1SVM.html	Parameter ¥(¥gamma¥) of a kernel function. For SVM::POLY, SVM::RBF, SVM::SIGMOID or SVM::CHI2. Default value is 1.		See alsosetGamma	1										
ml_SVM_setGamma	CVAPI(ExceptionStatus) ml_SVM_setGamma(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setGamma(val); END_WRAP }	cv::ml::SVM::setGamma()	classcv_1_1ml_1_1SVM.html	See alsogetGamma			1										
ml_SVM_getCoef0	CVAPI(ExceptionStatus) ml_SVM_getCoef0(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getCoef0(); END_WRAP }	cv::ml::SVM::getCoef0()	classcv_1_1ml_1_1SVM.html	Parameter coef0 of a kernel function. For SVM::POLY or SVM::SIGMOID. Default value is 0.		See alsosetCoef0	1										
ml_SVM_setCoef0	CVAPI(ExceptionStatus) ml_SVM_setCoef0(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setCoef0(val); END_WRAP }	cv::ml::SVM::setCoef0()	classcv_1_1ml_1_1SVM.html	See alsogetCoef0			1										
ml_SVM_getDegree	CVAPI(ExceptionStatus) ml_SVM_getDegree(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getDegree(); END_WRAP }	cv::ml::SVM::getDegree()	classcv_1_1ml_1_1SVM.html	Parameter degree of a kernel function. For SVM::POLY. Default value is 0.		See alsosetDegree	1										
ml_SVM_setDegree	CVAPI(ExceptionStatus) ml_SVM_setDegree(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setDegree(val); END_WRAP }	cv::ml::SVM::setDegree()	classcv_1_1ml_1_1SVM.html	See alsogetDegree			1										
ml_SVM_getC	CVAPI(ExceptionStatus) ml_SVM_getCoef0(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getCoef0(); END_WRAP }	cv::ml::SVM::getCoef0()	classcv_1_1ml_1_1SVM.html	Parameter coef0 of a kernel function. For SVM::POLY or SVM::SIGMOID. Default value is 0.		See alsosetCoef0	1										
ml_SVM_setC	CVAPI(ExceptionStatus) ml_SVM_setCoef0(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setCoef0(val); END_WRAP }	cv::ml::SVM::setCoef0()	classcv_1_1ml_1_1SVM.html	See alsogetCoef0			1										
ml_SVM_getP	CVAPI(ExceptionStatus) ml_SVM_getP(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getP(); END_WRAP }	cv::ml::SVM::getP()	classcv_1_1ml_1_1SVM.html	Parameter ¥(¥epsilon¥) of a SVM optimization problem. For SVM::EPS_SVR. Default value is 0.		See alsosetP	1										
ml_SVM_setP	CVAPI(ExceptionStatus) ml_SVM_setP(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setP(val); END_WRAP }	cv::ml::SVM::setP()	classcv_1_1ml_1_1SVM.html	See alsogetP			1										
ml_SVM_getNu	CVAPI(ExceptionStatus) ml_SVM_getNu(cv::ml::SVM *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getNu(); END_WRAP }	cv::ml::SVM::getNu()	classcv_1_1ml_1_1SVM.html	Parameter ¥(¥nu¥) of a SVM optimization problem. For SVM::NU_SVC, SVM::ONE_CLASS or SVM::NU_SVR. Default value is 0.		See alsosetNu	1										
ml_SVM_setNu	CVAPI(ExceptionStatus) ml_SVM_setNu(cv::ml::SVM *obj, double val) { BEGIN_WRAP obj->setNu(val); END_WRAP }	cv::ml::SVM::setNu()	classcv_1_1ml_1_1SVM.html	See alsogetNu			1										
ml_SVM_getClassWeights	CVAPI(ExceptionStatus) ml_SVM_getClassWeights(cv::ml::SVM *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->getClassWeights()); END_WRAP }	cv::ml::SVM::getClassWeights()																
ml_SVM_setClassWeights	CVAPI(ExceptionStatus) ml_SVM_setClassWeights(cv::ml::SVM *obj, cv::Mat *val) { BEGIN_WRAP obj->setClassWeights(*val); END_WRAP }	cv::ml::SVM::setClassWeights()	classcv_1_1ml_1_1SVM.html	See alsogetClassWeights			1										
ml_SVM_getTermCriteria	CVAPI(ExceptionStatus) ml_SVM_getTermCriteria(cv::ml::SVM *obj, MyCvTermCriteria *returnValue) { BEGIN_WRAP *returnValue = c(obj->getTermCriteria()); END_WRAP }	cv::ml::SVM::getTermCriteria()																
ml_SVM_setTermCriteria	CVAPI(ExceptionStatus) ml_SVM_setTermCriteria(cv::ml::SVM *obj, MyCvTermCriteria val) { BEGIN_WRAP obj->setTermCriteria(cpp(val)); END_WRAP }	cv::ml::SVM::setTermCriteria()	classcv_1_1ml_1_1SVM.html	See alsogetTermCriteria			1										
ml_SVM_getKernelType	CVAPI(ExceptionStatus) ml_SVM_getKernelType(cv::ml::SVM *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getKernelType(); END_WRAP }	cv::ml::SVM::getKernelType()	classcv_1_1ml_1_1SVM.html	Type of a SVM kernel. See SVM::KernelTypes. Default value is SVM::RBF.			1										
ml_SVM_setKernel	CVAPI(ExceptionStatus) ml_SVM_setKernel(cv::ml::SVM *obj, int kernelType) { BEGIN_WRAP obj->setKernel(kernelType); END_WRAP }	cv::ml::SVM::setKernel()	classcv_1_1ml_1_1SVM.html	Initialize with one of predefined kernels. See SVM::KernelTypes.			1										
ml_SVM_getSupportVectors	CVAPI(ExceptionStatus) ml_SVM_getSupportVectors(cv::ml::SVM *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = new cv::Mat(obj->getSupportVectors()); END_WRAP }	cv::ml::SVM::getSupportVectors()	classcv_1_1ml_1_1SVM.html	Retrieves all the support vectors.		The method returns all the support vectors as a floating-point matrix, where support vectors are stored as matrix rows.	1										

ml_SVM_getDecisionFunction	CVAPI(ExceptionStatus) ml_SVM_getDecisionFunction(cv::ml::SVM *obj, int i, cv::_OutputArray *alpha, cv::_OutputArray *svids, double *returnValue) { BEGIN_WRAP *returnValue = obj->getDecisionFunction(i, entity(alpha), entity(svids)); END_WRAP }	cv::ml::SVM::getDecisionFunction()	classcv_1_1ml_1_1SVM.html	Retrieves the decision function.	i : the index of the decision function. If the problem solved is regression, 1-class or 2-class classification, then there will be just one decision function and the index should always be 0. Otherwise, in the case of N-class classification, there will be \(N(N-1)/2\) decision functions. alpha : the optional output vector for weights, corresponding to different support vectors. In the case of linear SVM all the alpha's will be 1's. svids : the optional output vector of indices of support vectors within the matrix of support vectors (which can be retrieved by SVM::getSupportVectors). In the case of linear SVM each decision function consists of a single "compressed" support vector.	The method returns rho parameter of the decision function, a scalar subtracted from the weighted sum of kernel responses.	3								
ml_SVM_getDefaultGrid	CVAPI(ExceptionStatus) ml_SVM_getDefaultGrid(int param_id, ParamGridStruct *returnValue) { BEGIN_WRAP *returnValue = c[cv::ml::SVM::getDefaultGrid(param_id)]; END_WRAP }	cv::ml::SVM::getDefaultGrid()	classcv_1_1ml_1_1SVM.html	Generates a grid for SVM parameters.	param_id : SVM parameters IDs that must be one of the SVM::ParamTypes. The grid is generated for the parameter with this ID.	The function generates a grid for the specified parameter of the SVM algorithm. The grid may be passed to the function SVM::trainAuto.	1								
ml_SVM_create	CVAPI(ExceptionStatus) ml_SVM_create(cv::Ptr<cv::ml::SVM> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::SVM::create(); *returnValue = new cv::Ptr<cv::ml::SVM>(ptr); END_WRAP }	cv::ml::SVM::create()	classcv_1_1ml_1_1SVM.html	Creates empty model. Use StatModel::train to train the model. Since SVM has several parameters, you may want to find the best parameters for your problem, it can be done with SVM::trainAuto.		Examples: samples/cpp/train_HOG.cpp.	1								
ml_Ptr_SVM_delete	CVAPI(ExceptionStatus) ml_Ptr_SVM_delete(cv::Ptr<cv::ml::SVM> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ml::SVM	classcv_1_1ml_1_1SVM.html	Support Vector Machines. See alsoSupport Vector Machines								•		
ml_Ptr_SVM_get	CVAPI(ExceptionStatus) ml_Ptr_SVM_get(cv::Ptr<cv::ml::SVM>* obj, cv::ml::SVM **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ml::SVM	classcv_1_1ml_1_1SVM.html	Support Vector Machines. See alsoSupport Vector Machines									•		
ml_SVM_load	CVAPI(ExceptionStatus) ml_SVM_load(const char *filePath, cv::Ptr<cv::ml::SVM> **returnValue) { BEGIN_WRAP const auto ptr = cv::ml::SVM::load(filePath); *returnValue = new cv::Ptr<cv::ml::SVM>(ptr); END_WRAP }	cv::ml::SVM::load()	classcv_1_1ml_1_1SVM.html	Loads and creates a serialized svm from a file.	filepath : path to serialized svm	Use SVM::save to serialize and store an SVM to disk. Load the SVM from this file again, by calling this function with the path to the file.	1								
ml_SVM_loadFromString	CVAPI(ExceptionStatus) ml_SVM_loadFromString(const char *strModel, cv::Ptr<cv::ml::SVM> **returnValue) { BEGIN_WRAP const auto objName = cv::ml::SVM::create()->getDefaultName(); const auto ptr = cv::Algorithm::loadFromString<cv::ml::SVM>(strModel, objName); *returnValue = new cv::Ptr<cv::ml::SVM>(ptr); END_WRAP }	cv::ml::SVM	classcv_1_1ml_1_1SVM.html	Support Vector Machines. See alsoSupport Vector Machines					•						
objdetect_groupRectangles1	CVAPI(ExceptionStatus) objdetect_groupRectangles1(std::vector<cv::Rect> *rectList, int groupThreshold, double eps) { BEGIN_WRAP cv::groupRectangles(*rectList, groupThreshold, eps); END_WRAP }	cv::groupRectangles()	group__objdetect.html	Groups the object candidate rectangles.	rectList : Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.) groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it. eps : Relative difference between sides of the rectangles to merge them into a group.	The function is a wrapper for the generic function partition . It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by eps. When eps=0 , no clustering is done at all. If \(\texttt{eps}\rightarrow +\inf\) , all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to groupThreshold rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.	3								
objdetect_groupRectangles2	CVAPI(ExceptionStatus) objdetect_groupRectangles2(std::vector<cv::Rect> *rectList, std::vector<int> *weights, int groupThreshold, double eps) { BEGIN_WRAP cv::groupRectangles(*rectList, *weights, groupThreshold, eps); END_WRAP }	cv::groupRectangles()	group__objdetect.html	Groups the object candidate rectangles.	rectList : Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.) groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it. eps : Relative difference between sides of the rectangles to merge them into a group.	The function is a wrapper for the generic function partition . It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by eps. When eps=0 , no clustering is done at all. If \(\texttt{eps}\rightarrow +\inf\) , all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to groupThreshold rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.	4								
objdetect_groupRectangles3	CVAPI(ExceptionStatus) objdetect_groupRectangles3(std::vector<cv::Rect> *rectList, int groupThreshold, double eps, std::vector<int> *weights, std::vector<double> *levelWeights) { BEGIN_WRAP cv::groupRectangles(*rectList, groupThreshold, eps, weights, levelWeights); END_WRAP }	cv::groupRectangles()	group__objdetect.html	Groups the object candidate rectangles.	rectList : Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.) groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it. eps : Relative difference between sides of the rectangles to merge them into a group.	The function is a wrapper for the generic function partition . It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by eps. When eps=0 , no clustering is done at all. If \(\texttt{eps}\rightarrow +\inf\) , all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to groupThreshold rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.	5								
objdetect_groupRectangles4	CVAPI(ExceptionStatus) objdetect_groupRectangles4(std::vector<cv::Rect> *rectList, std::vector<int> *rejectLevels, std::vector<double> *levelWeights, int groupThreshold, double eps) { BEGIN_WRAP cv::groupRectangles(*rectList, *rejectLevels, *levelWeights, groupThreshold, eps); END_WRAP }	cv::groupRectangles()	group__objdetect.html	Groups the object candidate rectangles.	rectList : Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.) groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it. eps : Relative difference between sides of the rectangles to merge them into a group.	The function is a wrapper for the generic function partition . It clusters all the input rectangles using the rectangle equivalence criteria that combines rectangles with similar sizes and similar locations. The similarity is defined by eps. When eps=0 , no clustering is done at all. If \(\texttt{eps}\rightarrow +\inf\) , all the rectangles are put in one cluster. Then, the small clusters containing less than or equal to groupThreshold rectangles are rejected. In each other cluster, the average rectangle is computed and put into the output rectangle list.	5								
objdetect_groupRectangles_meanshift	CVAPI(ExceptionStatus) objdetect_groupRectangles_meanshift(std::vector<cv::Rect> *rectList, std::vector<double> *foundWeights, std::vector<double> *foundScales, double detectThreshold, MyCvSize winDetSize) { BEGIN_WRAP cv::groupRectangles_meanshift(*rectList, *foundWeights, *foundScales, detectThreshold, cpp(winDetSize)); END_WRAP }	cv::groupRectangles_meanshift()	group__objdetect.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.			5								
objdetect_CascadeClassifier_new	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_new(cv::CascadeClassifier **returnValue) { BEGIN_WRAP *returnValue = new cv::CascadeClassifier; END_WRAP }	cv::CascadeClassifier	classcv_1_1CascadeClassifier.html	Cascade classifier class for object detection. Examples: samples/cpp/facedetect.cpp.								•			
objdetect_CascadeClassifier_newFromFile	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_newFromFile(const char *fileName, cv::CascadeClassifier **returnValue) { BEGIN_WRAP *returnValue = new cv::CascadeClassifier(fileName); END_WRAP }	cv::CascadeClassifier	classcv_1_1CascadeClassifier.html	Cascade classifier class for object detection. Examples: samples/cpp/facedetect.cpp.								•			
objdetect_CascadeClassifier_delete	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_delete(cv::CascadeClassifier *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::CascadeClassifier	classcv_1_1CascadeClassifier.html	Cascade classifier class for object detection. Examples: samples/cpp/facedetect.cpp.									•		

objdetect_CascadeClassifier_empty	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_empty(cv::CascadeClassifier *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->empty() ? 1 : 0; END_WRAP }	cv::CascadeClassifier::empty()	classcv_1_1CascadeClassifier.html	Checks whether the classifier has been loaded.		Examples: samples/cpp/facedetect.cpp.	1									
objdetect_CascadeClassifier_load	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_load(cv::CascadeClassifier *obj, const char *fileName, int *returnValue) { BEGIN_WRAP *returnValue = obj->load(fileName) ? 1 : 0; END_WRAP }	cv::CascadeClassifier::load()	classcv_1_1CascadeClassifier.html	Loads a classifier from a file.	filename : Name of the file from which the classifier is loaded. The file may contain an old HAAR classifier trained by the haartraining application or a new cascade classifier trained by the traincascade application.	Examples: samples/cpp/facedetect.cpp.	1									
objdetect_CascadeClassifier_detectMultiScale1	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_detectMultiScale1(cv::CascadeClassifier *obj, cv::Mat *image, std::vector<cv::Rect> *objects, double scaleFactor, int minNeighbors, int flags, MyCvSize minSize, MyCvSize maxSize) { BEGIN_WRAP obj->detectMultiScale(*image, *objects, scaleFactor, minNeighbors, flags, cpp(minSize), cpp(maxSize)); END_WRAP }	cv::CascadeClassifier::detectMultiScale()	classcv_1_1CascadeClassifier.html	Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.	image : Matrix of the type CV_8U containing an image where objects are detected. objects : Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image. scaleFactor : Parameter specifying how much the image size is reduced at each image scale. minNeighbors : Parameter specifying how many neighbors each candidate rectangle should have to retain it. flags : Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade. minSize : Minimum possible object size. Objects smaller than that are ignored. maxSize : Maximum possible object size. Objects larger than that are ignored. If maxSize == minSize model is evaluated on single scale.	The function is parallelized with the TBB library.Note (Python) A face detection example using cascade classifiers can be found at opencv_source_code/samples/python/facedetect.pyExamples: samples/cpp/facedetect.cpp.	7									
objdetect_CascadeClassifier_detectMultiScale2	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_detectMultiScale2(cv::CascadeClassifier *obj, cv::Mat *image, std::vector<cv::Rect> *objects, std::vector<int> *rejectLevels, std::vector<double> *levelWeights, double scaleFactor, int minNeighbors, int flags, MyCvSize minSize, MyCvSize maxSize, int outputRejectLevels) { BEGIN_WRAP obj->detectMultiScale(*image, *objects, *rejectLevels, *levelWeights, scaleFactor, minNeighbors, flags, cpp(minSize), cpp(maxSize), outputRejectLevels != 0); END_WRAP }	cv::CascadeClassifier::detectMultiScale()	classcv_1_1CascadeClassifier.html	Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.	image : Matrix of the type CV_8U containing an image where objects are detected. objects : Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image. scaleFactor : Parameter specifying how much the image size is reduced at each image scale. minNeighbors : Parameter specifying how many neighbors each candidate rectangle should have to retain it. flags : Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade. minSize : Minimum possible object size. Objects smaller than that are ignored. maxSize : Maximum possible object size. Objects larger than that are ignored. If maxSize == minSize model is evaluated on single scale.	The function is parallelized with the TBB library.Note (Python) A face detection example using cascade classifiers can be found at opencv_source_code/samples/python/facedetect.pyExamples: samples/cpp/facedetect.cpp.	10									
objdetect_CascadeClassifier_isOldFormatCascade	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_isOldFormatCascade(cv::CascadeClassifier *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isOldFormatCascade() ? 1 : 0; END_WRAP }	cv::CascadeClassifier::isOldFormatCascade()	classcv_1_1CascadeClassifier.html				1									
objdetect_CascadeClassifier_getOriginalWindow	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_getOriginalWindowSize(cv::CascadeClassifier *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->getOriginalWindowSize()); END_WRAP }	cv::CascadeClassifier::getOriginalWindowSize()	classcv_1_1CascadeClassifier.html				1									
objdetect_CascadeClassifier_getFeatureType	CVAPI(ExceptionStatus) objdetect_CascadeClassifier_getFeatureType(cv::CascadeClassifier *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getFeatureType(); END_WRAP }	cv::CascadeClassifier::getFeatureType()	classcv_1_1CascadeClassifier.html				1									
objdetect_HOGDescriptor_new1	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_new1(cv::HOGDescriptor **returnValue) { BEGIN_WRAP *returnValue = new cv::HOGDescriptor; END_WRAP }	cv::HOGDescriptor	structcv_1_1HOGDescriptor.html	Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. the HOG descriptor algorithm introduced by Navneet Dalal and Bill Triggs [50] . useful links: https://hal.inria.fr/inria-00548512/document/ https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor http://www.learnopencv.com/histogram-of-oriented-gradients http://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.							•			
objdetect_HOGDescriptor_new2	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_new2(MyCvSize winSize, MyCvSize blockSize, MyCvSize blockStride, MyCvSize cellSize, int nbins, int derivAperture, double winSigma, int histogramNormType, double L2HysThreshold, int gammaCorrection, int nlevels, cv::HOGDescriptor **returnValue) { BEGIN_WRAP *returnValue = new cv::HOGDescriptor(cpp(winSize), cpp(blockSize), cpp(blockStride), cpp(cellSize), nbins, derivAperture, winSigma, static_cast<cv::HOGDescriptor::HistogramNormType>(histogramNormType), L2HysThreshold, gammaCorrection, nlevels); END_WRAP }	cv::HOGDescriptor	structcv_1_1HOGDescriptor.html	Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. the HOG descriptor algorithm introduced by Navneet Dalal and Bill Triggs [50] . useful links: https://hal.inria.fr/inria-00548512/document/ https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor http://www.learnopencv.com/histogram-of-oriented-gradients http://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.							•			
objdetect_HOGDescriptor_new3	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_new3(const char *filename, cv::HOGDescriptor **returnValue) { BEGIN_WRAP *returnValue = new cv::HOGDescriptor(filename); END_WRAP }	cv::HOGDescriptor	structcv_1_1HOGDescriptor.html	Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. the HOG descriptor algorithm introduced by Navneet Dalal and Bill Triggs [50] . useful links: https://hal.inria.fr/inria-00548512/document/ https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor http://www.learnopencv.com/histogram-of-oriented-gradients http://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.							•			
objdetect_HOGDescriptor_delete	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_delete(cv::HOGDescriptor *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::HOGDescriptor	structcv_1_1HOGDescriptor.html	Implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector. the HOG descriptor algorithm introduced by Navneet Dalal and Bill Triggs [50] . useful links: https://hal.inria.fr/inria-00548512/document/ https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descriptor http://www.learnopencv.com/histogram-of-oriented-gradients http://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.								•		
objdetect_HOGDescriptor_getDescriptorSize	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_getDescriptorSize(cv::HOGDescriptor *obj, size_t *returnValue) { BEGIN_WRAP *returnValue = obj->getDescriptorSize(); END_WRAP }	cv::HOGDescriptor::getDescriptorSize()	structcv_1_1HOGDescriptor.html	Returns the number of coefficients required for the classification.			1									

objdetect_HOGDescriptor_checkDetectorSize	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_checkDetectorSize(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->checkDetectorSize() ? 1 : 0; END_WRAP }	cv::HOGDescriptor::checkDetectorSize()	structcv_1_1HOGDescriptor.html	Checks if detector size equal to descriptor size.			1								
objdetect_HOGDescriptor_getWinSigma	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_getWinSigma(cv::HOGDescriptor *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getWinSigma(); END_WRAP }	cv::HOGDescriptor::getWinSigma()	structcv_1_1HOGDescriptor.html	Returns winSigma value.			1								
objdetect_HOGDescriptor_setSVMDetector	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_setSVMDetector(cv::HOGDescriptor *obj, std::vector<float> *svmDetector) { BEGIN_WRAP obj->setSVMDetector(*svmDetector); END_WRAP }	cv::HOGDescriptor::setSVMDetector()	structcv_1_1HOGDescriptor.html	Sets coefficients for the linear SVM classifier.	svmdetector : coefficients for the linear SVM classifier.	Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.	1								
objdetect_HOGDescriptor_load	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_load(cv::HOGDescriptor *obj, const char *filename, const char *objName, int *returnValue) { BEGIN_WRAP std::string objNameStr; if (objName != nullptr) objNameStr = std::string(objName); *returnValue = obj->load(filename, objNameStr); END_WRAP }	cv::HOGDescriptor::load()	structcv_1_1HOGDescriptor.html	loads HOGDescriptor parameters and coefficients for the linear SVM classifier from a file.	filename : Path of the file to read. objname : The optional name of the node to read (if empty, the first top-level node will be used).	Examples: samples/cpp/train_HOG.cpp.	2								
objdetect_HOGDescriptor_save	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_save(cv::HOGDescriptor *obj, const char *filename, const char *objName) { BEGIN_WRAP std::string objNameStr; if (objName != nullptr) objNameStr = cv::String(objName); obj->save(filename, objNameStr); END_WRAP }	cv::HOGDescriptor::save()	structcv_1_1HOGDescriptor.html	saves HOGDescriptor parameters and coefficients for the linear SVM classifier to a file	filename : File name objname : Object name	Examples: samples/cpp/train_HOG.cpp.	2								
objdetect_HOGDescriptor_compute	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_compute(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<float> *descriptors, MyCvSize winStride, MyCvSize padding, cv::Point* locations, int locationsLength) { BEGIN_WRAP std::vector<cv::Point> locationsVec; if (locations != nullptr) locationsVec = std::vector<cv::Point>(locations, locations + locationsLength); obj->compute(*img, *descriptors, cpp(winStride), cpp(padding), locationsVec); END_WRAP }	cv::HOGDescriptor::compute()	structcv_1_1HOGDescriptor.html	Computes HOG descriptors of given image.	img : Matrix of the type CV_8U containing an image where HOG features will be calculated. descriptors : Matrix of the type CV_32F winStride : Window stride. It must be a multiple of block stride. padding : Padding locations : Vector of Point	Examples: samples/cpp/train_HOG.cpp.	5								
objdetect_HOGDescriptor_detect1	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detect1(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<cv::Point> *foundLocations, double hitThreshold, MyCvSize winStride, MyCvSize padding, cv::Point* searchLocations, int searchLocationsLength) { BEGIN_WRAP std::vector<cv::Point> stVec; if (searchLocations != nullptr) stVec = std::vector<cv::Point>(searchLocations, searchLocations + searchLocationsLength); obj->detect(*img,	cv::HOGDescriptor::detect()	structcv_1_1HOGDescriptor.html	Performs object detection without a multi-scale window.	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. foundLocations : Vector of point where each point contains left-top corner point of detected object boundaries. hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here. winStride : Window stride. It must be a multiple of block stride. padding : Padding searchLocations : Vector of Point includes locations to search.		6								
objdetect_HOGDescriptor_detect2	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detect2(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<cv::Point> *foundLocations, std::vector<double> *weights, double hitThreshold, MyCvSize winStride, MyCvSize padding, cv::Point* searchLocations, int searchLocationsLength) { BEGIN_WRAP std::vector<cv::Point> stVec; if (searchLocations != nullptr) stVec = std::vector<cv::Point>(searchLocations, searchLocations + searchLocationsLength);	cv::HOGDescriptor::detect()	structcv_1_1HOGDescriptor.html	Performs object detection without a multi-scale window.	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. foundLocations : Vector of point where each point contains left-top corner point of detected object boundaries. weights : Vector that will contain confidence values for each detected object. hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here. winStride : Window stride. It must be a multiple of block stride. padding : Padding searchLocations : Vector of Point includes set of requested locations to be evaluated.		7								
objdetect_HOGDescriptor_detectMultiScale1	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detectMultiScale1(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<cv::Rect> *foundLocations, double hitThreshold, MyCvSize winStride, MyCvSize padding, double scale, int groupThreshold) { BEGIN_WRAP obj->detectMultiScale(*img, *foundLocations, hitThreshold, cpp(winStride), cpp(padding), scale, groupThreshold); END_WRAP }	cv::HOGDescriptor::detectMultiScale()	structcv_1_1HOGDescriptor.html	Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. foundLocations : Vector of rectangles where each rectangle contains the detected object. foundWeights : Vector that will contain confidence values for each detected object. hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here. winStride : Window stride. It must be a multiple of block stride. padding : Padding scale : Coefficient of the detection window increase. finalThreshold : Final threshold useMeanshiftGrouping : indicates grouping algorithm	Examples: samples/cpp/peopledetect.cpp, samples/cpp/train_HOG.cpp, and samples/tapi/hog.cpp.	7								
objdetect_HOGDescriptor_detectMultiScale2	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detectMultiScale2(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<cv::Rect> *foundLocations, std::vector<double> *foundWeights, double hitThreshold, MyCvSize winStride, MyCvSize padding, double scale, int groupThreshold) { BEGIN_WRAP obj->detectMultiScale(*img, *foundLocations, *foundWeights, hitThreshold, cpp(winStride), cpp(padding), scale, groupThreshold); END_WRAP }	cv::HOGDescriptor::detectMultiScale()	structcv_1_1HOGDescriptor.html	Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. foundLocations : Vector of rectangles where each rectangle contains the detected object. hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here. winStride : Window stride. It must be a multiple of block stride. padding : Padding scale : Coefficient of the detection window increase. finalThreshold : Final threshold useMeanshiftGrouping : indicates grouping algorithm		8								

objdetect_HOGDescriptor_computeGradient	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_computeGradient(cv::HOGDescriptor *obj, cv::Mat* img, cv::Mat* grad, cv::Mat* anglesOfs, MyCvSize paddingTL, MyCvSize paddingBR) { BEGIN_WRAP obj->computeGradient(*img, *grad, *anglesOfs, cpp(paddingTL), cpp(paddingBR)); END_WRAP }	cv::HOGDescriptor::computeGradient()	structcv_1_1HOGDescriptor.html	Computes gradients and quantized gradient orientations.	img : Matrix contains the image to be computed grad : Matrix of type CV_32FC2 contains computed gradients anglesOfs : Matrix of type CV_8UC2 contains quantized gradient orientations paddingTL : Padding from top-left paddingBR : Padding from bottom-right		5								
objdetect_HOGDescriptor_detectROI	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detectROI(cv::HOGDescriptor *obj, cv::Mat *img, cv::Point *locations, int locationsLength, std::vector<cv::Point> *foundLocations, std::vector<double> *confidences, double hitThreshold, MyCvSize winStride, MyCvSize padding) { BEGIN_WRAP const std::vector<cv::Point> locationsVec(locations, locations + locationsLength); obj->detectROI(*img, locationsVec, *foundLocations, *confidences, hitThreshold, cpp(winStride), cpp(padding)); END_WRAP }	cv::HOGDescriptor::detectROI()	structcv_1_1HOGDescriptor.html	evaluate specified ROI and return confidence value for each location	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. locations : Vector of Point foundLocations : Vector of Point where each Point is detected object's top-left point. confidences : confidences hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here winStride : winStride padding : padding		7								
objdetect_HOGDescriptor_detectMultiScaleROI	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_detectMultiScaleROI(cv::HOGDescriptor *obj, cv::Mat *img, std::vector<cv::Rect> *foundLocations, std::vector<double> *roiScales, std::vector<std::vector<cv::Point> > *roiLocations, std::vector<std::vector<double> > *roiConfidences, double hitThreshold, int groupThreshold) { BEGIN_WRAP std::vector<cv::DetectionROI> locations; obj->detectMultiScaleROI(*img, *foundLocations, locations, hitThreshold, groupThreshold); END_WRAP }	cv::HOGDescriptor::detectMultiScaleROI()	structcv_1_1HOGDescriptor.html	evaluate specified ROI and return confidence value for each location in multiple scales	img : Matrix of the type CV_8U or CV_8UC3 containing an image where objects are detected. foundLocations : Vector of rectangles where each rectangle contains the detected object. locations : Vector of DetectionROI hitThreshold : Threshold for the distance between features and SVM classifying plane. Usually it is 0 and should be specified in the detector coefficients (as the last free coefficient). But if the free coefficient is omitted (which is allowed), you can specify it manually here. groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it.		5								
objdetect_HOGDescriptor_groupRectangles	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_groupRectangles(cv::HOGDescriptor *obj, std::vector<cv::Rect> *rectList, std::vector<double> *weights, int groupThreshold, double eps) { BEGIN_WRAP obj->groupRectangles(*rectList, *weights, groupThreshold, eps); END_WRAP }	cv::HOGDescriptor::groupRectangles()	structcv_1_1HOGDescriptor.html	Groups the object candidate rectangles.	rectList : Input/output vector of rectangles. Output vector includes retained and grouped rectangles. (The Python list is not modified in place.) weights : Input/output vector of weights of rectangles. Output vector includes weights of retained and grouped rectangles. (The Python list is not modified in place.) groupThreshold : Minimum possible number of rectangles minus 1. The threshold is used in a group of rectangles to retain it. eps : Relative difference between sides of the rectangles to merge them into a group.		4								
objdetect_HOGDescriptor_winSize_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_winSize_get(cv::HOGDescriptor *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->winSize); END_WRAP }	cv::HOGDescriptor::winSize	structcv_1_1HOGDescriptor.html	Detection window size. Align to block size and block stride. Default value is Size(64,128).		Examples: samples/cpp/train_HOG.cpp.				•					
objdetect_HOGDescriptor_blockSize_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_blockSize_get(cv::HOGDescriptor *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->blockSize); END_WRAP }	cv::HOGDescriptor::blockSize	structcv_1_1HOGDescriptor.html	Block size in pixels. Align to cell size. Default value is Size(16,16).						•					
objdetect_HOGDescriptor_blockStride_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_blockStride_get(cv::HOGDescriptor *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->blockStride); END_WRAP }	cv::HOGDescriptor::blockStride	structcv_1_1HOGDescriptor.html	Block stride. It must be a multiple of cell size. Default value is Size(8,8).						•					
objdetect_HOGDescriptor_cellSize_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_cellSize_get(cv::HOGDescriptor *obj, MyCvSize *returnValue) { BEGIN_WRAP *returnValue = c(obj->cellSize); END_WRAP }	cv::HOGDescriptor::cellSize	structcv_1_1HOGDescriptor.html	Cell size. Default value is Size(8,8).						•					
objdetect_HOGDescriptor_nbins_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_nbins_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->nbins; END_WRAP }	cv::HOGDescriptor::nbins	structcv_1_1HOGDescriptor.html	Number of bins used in the calculation of histogram of gradients. Default value is 9.						•					
objdetect_HOGDescriptor_derivAperture_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_derivAperture_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->derivAperture; END_WRAP }	cv::HOGDescriptor::derivAperture	structcv_1_1HOGDescriptor.html	not documented						•					
objdetect_HOGDescriptor_winSigma_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_winSigma_get(cv::HOGDescriptor *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->winSigma; END_WRAP }	cv::HOGDescriptor::winSigma	structcv_1_1HOGDescriptor.html	Gaussian smoothing window parameter.						•					
objdetect_HOGDescriptor_histogramNormType	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_histogramNormType_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->histogramNormType; END_WRAP }	cv::HOGDescriptor::histogramNormType	structcv_1_1HOGDescriptor.html	histogramNormType						•					
objdetect_HOGDescriptor_L2HysThreshold_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_L2HysThreshold_get(cv::HOGDescriptor *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->L2HysThreshold; END_WRAP }	cv::HOGDescriptor::L2HysThreshold	structcv_1_1HOGDescriptor.html	L2-Hys normalization method shrinkage.						•					
objdetect_HOGDescriptor_gammaCorrection_g	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_gammaCorrection_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->gammaCorrection ? 1 : 0; END_WRAP }	cv::HOGDescriptor::gammaCorrection	structcv_1_1HOGDescriptor.html	Flag to specify whether the gamma correction preprocessing is required or not.						•					

objdetect_HOGDescriptor_nlevels_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_nlevels_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->nlevels; END_WRAP }	cv::HOGDescriptor::nlevels	structcv_1_1HOGDescriptor.html	Maximum number of detection window increases. Default value is 64.		Examples: samples/tapi/hog.cpp.				•					
objdetect_HOGDescriptor_signedGradient_get	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_signedGradient_get(cv::HOGDescriptor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->signedGradient; END_WRAP }	cv::HOGDescriptor::signedGradient	structcv_1_1HOGDescriptor.html	Indicates signed gradient will be used or not.						•					
objdetect_HOGDescriptor_winSize_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_winSize_set(cv::HOGDescriptor *obj, MyCvSize value) { BEGIN_WRAP obj->winSize = cpp(value); END_WRAP }	cv::HOGDescriptor::winSize	structcv_1_1HOGDescriptor.html	Detection window size. Align to block size and block stride. Default value is Size(64,128).		Examples: samples/cpp/train_HOG.cpp.					•				
objdetect_HOGDescriptor_blockSize_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_blockSize_set(cv::HOGDescriptor *obj, MyCvSize value) { BEGIN_WRAP obj->blockSize = cpp(value); END_WRAP }	cv::HOGDescriptor::blockSize	structcv_1_1HOGDescriptor.html	Block size in pixels. Align to cell size. Default value is Size(16,16).							•				
objdetect_HOGDescriptor_blockStride_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_blockStride_set(cv::HOGDescriptor * obj, MyCvSize value) { BEGIN_WRAP obj->blockStride = cpp(value); END_WRAP }	cv::HOGDescriptor::blockStride	structcv_1_1HOGDescriptor.html	Block stride. It must be a multiple of cell size. Default value is Size(8,8).							•				
objdetect_HOGDescriptor_cellSize_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_cellSize_set(cv::HOGDescriptor *obj, MyCvSize value) { BEGIN_WRAP obj->cellSize = cpp(value); END_WRAP }	cv::HOGDescriptor::cellSize	structcv_1_1HOGDescriptor.html	Cell size. Default value is Size(8,8).							•				
objdetect_HOGDescriptor_nbins_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_nbins_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->nbins = value; END_WRAP }	cv::HOGDescriptor::nbins	structcv_1_1HOGDescriptor.html	Number of bins used in the calculation of histogram of gradients. Default value is 9.							•				
objdetect_HOGDescriptor_derivAperture_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_derivAperture_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->derivAperture = value; END_WRAP }	cv::HOGDescriptor::derivAperture	structcv_1_1HOGDescriptor.html	not documented							•				
objdetect_HOGDescriptor_winSigma_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_winSigma_set(cv::HOGDescriptor *obj, double value) { BEGIN_WRAP obj->winSigma = value; END_WRAP }	cv::HOGDescriptor::winSigma	structcv_1_1HOGDescriptor.html	Gaussian smoothing window parameter.							•				
objdetect_HOGDescriptor_histogramNormType	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_histogramNormType_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->histogramNormType = static_cast<cv::HOGDescriptor::HistogramNormType>(value); END_WRAP }	cv::HOGDescriptor::histogramNormType	structcv_1_1HOGDescriptor.html	histogramNormType							•				
objdetect_HOGDescriptor_L2HysThreshold_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_L2HysThreshold_set(cv::HOGDescriptor *obj, double value) { BEGIN_WRAP obj->L2HysThreshold = value; END_WRAP }	cv::HOGDescriptor::L2HysThreshold	structcv_1_1HOGDescriptor.html	L2-Hys normalization method shrinkage.							•				
objdetect_HOGDescriptor_gammaCorrection_s	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_gammaCorrection_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->gammaCorrection = (value != 0); END_WRAP }	cv::HOGDescriptor::gammaCorrection	structcv_1_1HOGDescriptor.html	Flag to specify whether the gamma correction preprocessing is required or not.							•				
objdetect_HOGDescriptor_nlevels_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_nlevels_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->nlevels = value; END_WRAP }	cv::HOGDescriptor::nlevels	structcv_1_1HOGDescriptor.html	Maximum number of detection window increases. Default value is 64.		Examples: samples/tapi/hog.cpp.					•				
objdetect_HOGDescriptor_signedGradient_set	CVAPI(ExceptionStatus) objdetect_HOGDescriptor_signedGradient_set(cv::HOGDescriptor *obj, int value) { BEGIN_WRAP obj->signedGradient = (value != 0); END_WRAP }	cv::HOGDescriptor::signedGradient	structcv_1_1HOGDescriptor.html	Indicates signed gradient will be used or not.							•				
objdetect_QRCodeDetector_new	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_new(cv::QRCodeDetector **returnValue) { BEGIN_WRAP *returnValue = new cv::QRCodeDetector(); END_WRAP }	cv::QRCodeDetector											•			
objdetect_QRCodeDetector_delete	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_delete(cv::QRCodeDetector *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::QRCodeDetector												•		
objdetect_QRCodeDetector_setEpsX	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_setEpsX(cv::QRCodeDetector *obj, double epsX) { BEGIN_WRAP obj->setEpsX(epsX); END_WRAP }	cv::QRCodeDetector::setEpsX()	classcv_1_1QRCodeDetector.html	sets the epsilon used during the horizontal scan of QR code stop marker detection.	epsX : Epsilon neighborhood, which allows you to determine the horizontal pattern of the scheme 1:1:3:1:1 according to QR code standard.		1								
objdetect_QRCodeDetector_setEpsY	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_setEpsY(cv::QRCodeDetector *obj, double epsY) { BEGIN_WRAP obj->setEpsY(epsY); END_WRAP }	cv::QRCodeDetector::setEpsY()	classcv_1_1QRCodeDetector.html	sets the epsilon used during the vertical scan of QR code stop marker detection.	epsY : Epsilon neighborhood, which allows you to determine the vertical pattern of the scheme 1:1:3:1:1 according to QR code standard.		1								
objdetect_QRCodeDetector_detect	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_detect(cv::QRCodeDetector *obj, cv::_InputArray *img, std::vector<cv::Point2f> *points, int *returnValue) { BEGIN_WRAP *returnValue = obj->detect(*img, *points) ? 1 : 0; END_WRAP }	cv::QRCodeDetector::detect()	classcv_1_1QRCodeDetector.html	Detects QR code in image and returns the quadrangle containing the code.	img : grayscale or color (BGR) image containing (or not) QR code. points : Output vector of vertices of the minimum-area quadrangle containing the code.		2								

objdetect_QRCodeDetector_decode	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_decode(cv::QRCodeDetector *obj, cv::_InputArray *img, std::vector<cv::Point2f> *points, cv::_OutputArray *straight_qrcode, std::string *returnValue) { BEGIN_WRAP *returnValue = obj->decode(*img, *points, entity(straight_qrcode)); END_WRAP }	cv::QRCodeDetector::decode()	classcv_1_1QRCodeDetector.html	Decodes QR code in image once it's found by the detect() method.	img : grayscale or color (BGR) image containing QR code. points : Quadrangle vertices found by detect() method (or some other algorithm). straight_qrcode : The optional output image containing rectified and binarized QR code	Returns UTF8-encoded output string or empty string if the code cannot be decoded.	3								
objdetect_QRCodeDetector_detectAndDecode	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_detectAndDecode(cv::QRCodeDetector *obj, cv::_InputArray *img, std::vector<cv::Point2f> *points, cv::_OutputArray *straight_qrcode, std::string *returnValue) { BEGIN_WRAP *returnValue = obj->detectAndDecode(*img, *points, entity(straight_qrcode)); END_WRAP }	cv::QRCodeDetector::detectAndDecode()	classcv_1_1QRCodeDetector.html	Both detects and decodes QR code.	img : grayscale or color (BGR) image containing QR code. points : optional output array of vertices of the found QR code quadrangle. Will be empty if not found. straight_qrcode : The optional output image containing rectified and binarized QR code		3								
objdetect_QRCodeDetector_detectMulti	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_detectMulti(cv::QRCodeDetector* obj, cv::_InputArray* img, std::vector<cv::Point2f>* points, int* returnValue) { BEGIN_WRAP *returnValue = obj->detectMulti(*img, *points) ? 1 : 0; END_WRAP }	cv::QRCodeDetector::detectMulti()	classcv_1_1QRCodeDetector.html	Detects QR codes in image and returns the vector of the quadrangles containing the codes.	img : grayscale or color (BGR) image containing (or not) QR codes. points : Output vector of vector of vertices of the minimum-area quadrangle containing the codes.		2								
objdetect_QRCodeDetector_decodeMulti	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_decodeMulti(cv::QRCodeDetector* obj, cv::_InputArray* img, std::vector<cv::Point2f>* points, std::vector<std::string>* decoded_info, std::vector<cv::Mat>* straight_qrcode, int* returnValue) { BEGIN_WRAP *returnValue = obj->decodeMulti(*img, *points, *decoded_info, *straight_qrcode) ? 1 : 0; END_WRAP }	cv::QRCodeDetector::decodeMulti()	classcv_1_1QRCodeDetector.html	Decodes QR codes in image once it's found by the detect() method.	img : grayscale or color (BGR) image containing QR codes. decoded_info : UTF8-encoded output vector of string or empty vector of string if the codes cannot be decoded. points : vector of Quadrangle vertices found by detect() method (or some other algorithm). straight_qrcode : The optional output vector of images containing rectified and binarized QR codes		4								
objdetect_QRCodeDetector_decodeMulti_NoStraightQrCode	CVAPI(ExceptionStatus) objdetect_QRCodeDetector_decodeMulti_NoStraightQrCode(cv::QRCodeDetector* obj, cv::_InputArray* img, std::vector<cv::Point2f>* points, std::vector<std::string>* decoded_info, int* returnValue) { BEGIN_WRAP *returnValue = obj->decodeMulti(*img, *points, *decoded_info) ? 1 : 0; END_WRAP }	cv::QRCodeDetector::decodeMulti()	classcv_1_1QRCodeDetector.html	Decodes QR codes in image once it's found by the detect() method.	img : grayscale or color (BGR) image containing QR codes. decoded_info : UTF8-encoded output vector of string or empty vector of string if the codes cannot be decoded. points : vector of Quadrangle vertices found by detect() method (or some other algorithm). straight_qrcode : The optional output vector of images containing rectified and binarized QR codes		3								
photo_inpaint	CVAPI(ExceptionStatus) photo_inpaint(cv::_InputArray *src, cv::_InputArray *inpaintMask, cv::_OutputArray *dst, double inpaintRadius, int flags) { BEGIN_WRAP cv::inpaint(*src, *inpaintMask, *dst, inpaintRadius, flags); END_WRAP }	cv::inpaint()	group__photo__inpaint.html	Restores the selected region in an image using the region neighborhood.	src : Input 8-bit, 16-bit unsigned or 32-bit float 1-channel or 8-bit 3-channel image. inpaintMask : Inpainting mask, 8-bit 1-channel image. Non-zero pixels indicate the area that needs to be inpainted. dst : Output image with the same size and type as src inpaintRadius : Radius of a circular neighborhood of each point inpainted that is considered by the algorithm. flags : Inpainting method that could be cv::INPAINT_NS or cv::INPAINT_TELEA	The function reconstructs the selected image area from the pixel near the area boundary. The function may be used to remove dust and scratches from a scanned photo, or to remove undesirable objects from still images or video. See http://en.wikipedia.org/wiki/Inpainting for more details.Note An example using the inpainting technique can be found at opencv_source_code/samples/cpp/inpaint.cpp (Python) An example using the inpainting technique can be found at opencv_source_code/samples/python/inpaint.py	5								
photo_fastNlMeansDenoising	CVAPI(ExceptionStatus) photo_fastNlMeansDenoising(cv::_InputArray *src, cv::_OutputArray *dst, float h, int templateWindowSize, int searchWindowSize) { BEGIN_WRAP cv::fastNlMeansDenoising(*src, *dst, h, templateWindowSize, searchWindowSize); END_WRAP }	cv::fastNlMeansDenoising()	group__photo__denoise.html	Perform image denoising using Non-local Means Denoising algorithm http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/ with several computational optimizations. Noise expected to be a gaussian white noise.	src : Input 8-bit 1-channel, 2-channel, 3-channel or 4-channel image. dst : Output image with the same size and type as src . templateWindowSize : Size in pixels of the template patch that is used to compute weights. Should be odd. Recommended value 7 pixels searchWindowSize : Size in pixels of the window that is used to compute weighted average for given pixel. Should be odd. Affect performance linearly: greater searchWindowsSize - greater denoising time. Recommended value 21 pixels h : Parameter regulating filter strength. Big h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise	This function expected to be applied to grayscale images. For colored images look at fastNlMeansDenoisingColored. Advanced usage of this functions can be manual denoising of colored image in different colorspaces. Such approach is used in fastNlMeansDenoisingColored by converting image to CIELAB colorspace and then separately denoise L and AB components with different h parameter.	5								
photo_fastNlMeansDenoisingColored	CVAPI(ExceptionStatus) photo_fastNlMeansDenoisingColored(cv::_InputArray *src, cv::_OutputArray *dst, float h, float hColor, int templateWindowSize, int searchWindowSize) { BEGIN_WRAP cv::fastNlMeansDenoisingColored(*src, *dst, h, hColor, templateWindowSize, searchWindowSize); END_WRAP }	cv::fastNlMeansDenoisingColored()	group__photo__denoise.html	Modification of fastNlMeansDenoising function for colored images.	src : Input 8-bit 3-channel image. dst : Output image with the same size and type as src templateWindowSize : Size in pixels of the template patch that is used to compute weights. Should be odd. Recommended value 7 pixels searchWindowSize : Size in pixels of the window that is used to compute weighted average for given pixel. Should be odd. Affect performance linearly: greater searchWindowsSize - greater denoising time. Recommended value 21 pixels h : Parameter regulating filter strength for luminance component. Bigger h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise hColor : The same as h but for color components. For most images value equals 10 will be enough to remove colored noise and do not distort colors	The function converts image to CIELAB colorspace and then separately denoise L and AB components with given h parameters using fastNlMeansDenoising function.	6								
photo_fastNlMeansDenoisingMulti	CVAPI(ExceptionStatus) photo_fastNlMeansDenoisingMulti(cv::_InputArray ** srcImgs, int srcImgsLength, cv::_OutputArray *dst, int imgToDenoiseIndex, int temporalWindowSize, float h, int templateWindowSize, int searchWindowSize) { BEGIN_WRAP std::vector<cv::_InputArray> srcImgsVec(srcImgsLength); for (int i = 0; i < srcImgsLength; i++) srcImgsVec[i] = *srcImgs[i]; cv::fastNlMeansDenoisingMulti(srcImgsVec, *dst, imgToDenoiseIndex, temporalWindowSize, h, templateWindowSize,	cv::fastNlMeansDenoisingMulti()	group__photo__denoise.html	Modification of fastNlMeansDenoising function for images sequence where consecutive images have been captured in small period of time. For example video. This version of the function is for grayscale images or for manual manipulation with colorspaces. For more details see http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.6394	srcImgs : Input 8-bit 1-channel, 2-channel, 3-channel or 4-channel images sequence. All images should have the same type and size. imgToDenoiseIndex : Target image to denoise index in srcImgs sequence temporalWindowSize : Number of surrounding images to use for target image denoising. Should be odd. Images from imgToDenoiseIndex - temporalWindowSize / 2 to imgToDenoiseIndex - temporalWindowSize / 2 from srcImgs will be used to denoise srcImgs[imgToDenoiseIndex] image. dst : Output image with the same size and type as srcImgs images. templateWindowSize : Size in pixels of the template patch that is used to compute weights. Should be odd. Recommended value 7 pixels searchWindowSize : Size in pixels of the window that is used to compute weighted average for given pixel. Should be odd. Affect performance linearly: greater searchWindowsSize - greater denoising time. Recommended value 21 pixels h : Parameter regulating filter strength. Bigger h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise		7								
photo_fastNlMeansDenoisingColoredMulti	CVAPI(ExceptionStatus) photo_fastNlMeansDenoisingColoredMulti(cv::_InputArray **srcImgs, int srcImgsLength, cv::_OutputArray *dst, int imgToDenoiseIndex, int temporalWindowSize, float h, float hColor, int templateWindowSize, int searchWindowSize) { BEGIN_WRAP std::vector<cv::_InputArray> srcImgsVec(srcImgsLength); for (int i = 0; i < srcImgsLength; i++) srcImgsVec[i] = *srcImgs[i]; cv::fastNlMeansDenoisingColoredMulti(srcImgsVec, *dst, imgToDenoiseIndex, temporalWindowSize, h, hColor, templateWindowSize, searchWindowSize); END_WRAP	cv::fastNlMeansDenoisingColoredMulti()	group__photo__denoise.html	Modification of fastNlMeansDenoisingMulti function for colored images sequences.	srcImgs : Input 8-bit 3-channel images sequence. All images should have the same type and size. imgToDenoiseIndex : Target image to denoise index in srcImgs sequence temporalWindowSize : Number of surrounding images to use for target image denoising. Should be odd. Images from imgToDenoiseIndex - temporalWindowSize / 2 to imgToDenoiseIndex - temporalWindowSize / 2 from srcImgs will be used to denoise srcImgs[imgToDenoiseIndex] image. dst : Output image with the same size and type as srcImgs images. templateWindowSize : Size in pixels of the template patch that is used to compute weights. Should be odd. Recommended value 7 pixels searchWindowSize : Size in pixels of the window that is used to compute weighted average for given pixel. Should be odd. Affect performance linearly: greater searchWindowsSize - greater denoising time. Recommended value 21 pixels h : Parameter regulating filter strength for luminance component. Bigger h value perfectly removes noise but also removes image details, smaller h value preserves details but also preserves some noise. hColor : The same as h but for color components.	The function converts images to CIELAB colorspace and then separately denoise L and AB components with given h parameters using fastNlMeansDenoisingMulti function.	8								

photo_denoise_TVL1	CVAPI(ExceptionStatus) photo_denoise_TVL1(cv::Mat **observations, int observationsSize, cv::Mat *result, double lambda, int niters) { BEGIN_WRAP std::vector<cv::Mat> observationsVec(observationsSize); for (int i = 0; i < observationsSize; i++) { observationsVec[i] = *observations[i]; } cv::denoise_TVL1(observationsVec, *result, lambda, niters); END_WRAP }	cv::denoise_TVL1()	group__photo__denoise.html	Primal-dual algorithm is an algorithm for solving special types of variational problems (that is, finding a function to minimize some functional). As the image denoising, in particular, may be seen as the variational problem, primal-dual algorithm then can be used to perform denoising and this is exactly what is implemented.	observations : This array should contain one or more noised versions of the image that is to be restored. result : Here the denoised image will be stored. There is no need to do pre-allocation of storage space, as it will be automatically allocated, if necessary. lambda : Corresponds to ¥(¥lambda¥) in the formulas above. As it is enlarged, the smooth (blurred) images are treated more favorably than detailed (but maybe more noised) ones. Roughly speaking, as it becomes smaller, the result will be more blur but more sever outliers will be removed. niters : Number of iterations that the algorithm will run. Of course, as more iterations as better, but it is hard to quantitatively refine this statement, so just use the default and increase it if the results are poor.	It should be noted, that this implementation was taken from the July 2013 blog entry [177] , which also contained (slightly more general) ready-to-use source code on Python. Subsequantly, that code was rewritten on C++ with the usage of openCV by Vadim Pisarevsky at the end of July 2013 and finally it was slightly adapted by later authors.Although the thorough discussion and justification of the algorithm involved may be found in [42], it might make sense to skim over it here, following [177] . To begin with, we consider the 1-byte gray-level images as the functions from the rectangular domain of pixels (it may be seen as set ¥(¥left¥{(x,y)¥in¥mathbb{N}^2¥times¥mathbb{N}^2 : 0 ¥leq x ¥leq n,¥; 1¥leq y ¥leq m¥right¥}¥) for some ¥(m,¥;n¥in¥mathbb{N}¥)) into ¥(¥{0,1,¥dots,255¥}^¥). We shall denote the noised images as ¥(f_¥) and with this view, given some image ¥(¥) of the same size, we may measure how bad it is by the formula¥[¥left¥| ¥left¥| ¥nabla	4									
photo_decolor	CVAPI(ExceptionStatus) photo_decolor(cv::_InputArray *src, cv::_OutputArray *grayscale, cv::_OutputArray *color_boost) { BEGIN_WRAP cv::decolor(*src, *grayscale, *color_boost); END_WRAP }	cv::decolor()	group__photo__decolor.html	Transforms a color image to a grayscale image. It is a basic tool in digital printing, stylized black-and-white photograph rendering, and in many single channel image processing applications [155]	src : Input 8-bit 3-channel image. grayscale : Output 8-bit 1-channel image. color_boost : Output 8-bit 3-channel image.	This function is to be applied on color images.	3									
photo_seamlessClone	CVAPI(ExceptionStatus) photo_seamlessClone(cv::_InputArray *src, cv::_InputArray *dst, cv::_InputArray *mask, MyCvPoint p, cv::_OutputArray *blend, int flags) { BEGIN_WRAP cv::seamlessClone(*src, *dst, entity(mask), cpp(p), *blend, flags); END_WRAP }	cv::seamlessClone()	group__photo__clone.html	Image editing tasks concern either global changes (color/intensity corrections, filters, deformations) or local changes concerned to a selection. Here we are interested in achieving local changes, ones that are restricted to a region manually selected (ROI), in a seamless and effortless manner. The extent of the changes ranges from slight distortions to complete replacement by novel content [192] .	src : Input 8-bit 3-channel image. dst : Input 8-bit 3-channel image. mask : Input 8-bit 1 or 3-channel image. p : Point in dst image where object is placed. blend : Output image with the same size and type as dst. flags : Cloning method that could be cv::NORMAL_CLONE, cv::MIXED_CLONE or cv::MONOCHROME_TRANSFER	Examples: samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp.	6									
photo_colorChange	CVAPI(ExceptionStatus) photo_colorChange(cv::_InputArray *src, cv::_InputArray *mask, cv::_OutputArray *dst, float red_mul, float green_mul, float blue_mul) { BEGIN_WRAP cv::colorChange(*src, entity(mask), *dst, red_mul, green_mul, blue_mul); END_WRAP }	cv::colorChange()	group__photo__clone.html	Given an original color image, two differently colored versions of this image can be mixed seamlessly.	src : Input 8-bit 3-channel image. mask : Input 8-bit 1 or 3-channel image. dst : Output image with the same size and type as src . red_mul : R-channel multiply factor. green_mul : G-channel multiply factor. blue_mul : B-channel multiply factor.	Multiplication factor is between .5 to 2.5.Examples: samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp.	6									
photo_illuminationChange	CVAPI(ExceptionStatus) photo_illuminationChange(cv::_InputArray *src, cv::_InputArray *mask, cv::_OutputArray *dst, float alpha, float beta = 0.4f) { BEGIN_WRAP cv::illuminationChange(*src, entity(mask), *dst, alpha, beta); END_WRAP }	cv::illuminationChange()	group__photo__clone.html	Applying an appropriate non-linear transformation to the gradient field inside the selection and then integrating back with a Poisson solver, modifies locally the apparent illumination of an image.	src : Input 8-bit 3-channel image. mask : Input 8-bit 1 or 3-channel image. dst : Output image with the same size and type as src. alpha : Value ranges between 0-2. beta : Value ranges between 0-2.	This is useful to highlight under-exposed foreground objects or to reduce specular reflections.Examples: samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp.	5									
photo_textureFlattening	CVAPI(ExceptionStatus) photo_textureFlattening(cv::_InputArray *src, cv::_InputArray *mask, cv::_OutputArray *dst, float low_threshold, float high_threshold, int kernel_size) { BEGIN_WRAP cv::textureFlattening(*src, entity(mask), *dst, low_threshold, high_threshold, kernel_size); END_WRAP }	cv::textureFlattening()	group__photo__clone.html	By retaining only the gradients at edge locations, before integrating with the Poisson solver, one washes out the texture of the selected region, giving its contents a flat aspect. Here Canny Edge Detector is used.	src : Input 8-bit 3-channel image. mask : Input 8-bit 1 or 3-channel image. dst : Output image with the same size and type as src. low_threshold : Range from 0 to 100. high_threshold : Value > 100. kernel_size : The size of the Sobel kernel to be used.	NoteThe algorithm assumes that the color of the source image is close to that of the destination. This assumption means that when the colors don't match, the source image color gets tinted toward the color of the destination image.Examples: samples/cpp/tutorial_code/photo/seamless_cloning/cloning_demo.cpp.	6									
photo_edgePreservingFilter	CVAPI(ExceptionStatus) photo_edgePreservingFilter(cv::_InputArray *src, cv::_OutputArray *dst, int flags, float sigma_s, float sigma_r) { BEGIN_WRAP cv::edgePreservingFilter(*src, *dst, flags, sigma_s, sigma_r); END_WRAP }	cv::edgePreservingFilter()	group__photo__render.html	Filtering is the fundamental operation in image and video processing. Edge-preserving smoothing filters are used in many different applications [86]	src : Input 8-bit 3-channel image. dst : Output 8-bit 3-channel image. flags : Edge preserving filters: cv::RECURS_FILTER or cv::NORMCONV_FILTER sigma_s : Range between 0 to 200. sigma_r : Range between 0 to 1.	Examples: samples/cpp/tutorial_code/photo/non_photorealistic_rendering/npr_demo.cpp.	5									
photo_detailEnhance	CVAPI(ExceptionStatus) photo_detailEnhance(cv::_InputArray *src, cv::_OutputArray *dst, float sigma_s, float sigma_r) { BEGIN_WRAP cv::detailEnhance(*src, *dst, sigma_s, sigma_r); END_WRAP }	cv::detailEnhance()	group__photo__render.html	This filter enhances the details of a particular image.	src : Input 8-bit 3-channel image. dst : Output image with the same size and type as src. sigma_s : Range between 0 to 200. sigma_r : Range between 0 to 1.	Examples: samples/cpp/tutorial_code/photo/non_photorealistic_rendering/npr_demo.cpp.	4									
photo_pencilSketch	CVAPI(ExceptionStatus) photo_pencilSketch(cv::_InputArray *src, cv::_OutputArray *dst1, cv::_OutputArray *dst2, float sigma_s, float sigma_r, float shade_factor) { BEGIN_WRAP cv::pencilSketch(*src, *dst1, *dst2, sigma_s, sigma_r, shade_factor); END_WRAP }	cv::pencilSketch()	group__photo__render.html	Pencil-like non-photorealistic line drawing.	src : Input 8-bit 3-channel image. dst1 : Output 8-bit 1-channel image. dst2 : Output image with the same size and type as src. sigma_s : Range between 0 to 200. sigma_r : Range between 0 to 1. shade_factor : Range between 0 to 0.1.	Examples: samples/cpp/tutorial_code/photo/non_photorealistic_rendering/npr_demo.cpp.	6									
photo_stylization	CVAPI(ExceptionStatus) photo_stylization(cv::_InputArray *src, cv::_OutputArray *dst, float sigma_s, float sigma_r) { BEGIN_WRAP cv::stylization(*src, *dst, sigma_s, sigma_r); END_WRAP }	cv::stylization()	group__photo__render.html	Stylization aims to produce digital imagery with a wide variety of effects not focused on photorealism. Edge-aware filters are ideal for stylization, as they can abstract regions of low contrast while preserving, or enhancing, high-contrast features.	src : Input 8-bit 3-channel image. dst : Output image with the same size and type as src. sigma_s : Range between 0 to 200. sigma_r : Range between 0 to 1.	Examples: samples/cpp/tutorial_code/photo/non_photorealistic_rendering/npr_demo.cpp.	4									
photo_createCalibrateDebevec	CVAPI(ExceptionStatus) photo_createCalibrateDebevec(int samples, float lambda, int random, cv::Ptr<cv::CalibrateDebevec> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::createCalibrateDebevec(samples, lambda, random != 0)); END_WRAP }	cv::createCalibrateDebevec()	group__photo__hdr.html	Creates CalibrateDebevec object.	samples : number of pixel locations to use lambda : smoothness term weight. Greater values produce smoother results, but can alter the response. random : If true sample pixel locations are chosen at random, otherwise they form a rectangular grid.		3									
photo_Ptr_CalibrateDebevec_delete	CVAPI(ExceptionStatus) photo_Ptr_CalibrateDebevec_delete(cv::Ptr<cv::CalibrateDebevec> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::CalibrateDebevec	classcv_1_1CalibrateDebevec.html	Inverse camera response function is extracted for each brightness value by minimizing an objective function as linear system. Objective function is constructed using pixel values on the same position in all images, extra term is added to make the result smoother. For more information see [55] .								•		
photo_Ptr_CalibrateDebevec_get	CVAPI(ExceptionStatus) photo_Ptr_CalibrateDebevec_get(cv::Ptr<cv::CalibrateDebevec> *obj, cv::CalibrateDebevec **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::CalibrateDebevec	classcv_1_1CalibrateDebevec.html	Inverse camera response function is extracted for each brightness value by minimizing an objective function as linear system. Objective function is constructed using pixel values on the same position in all images, extra term is added to make the result smoother. For more information see [55] .									•	
photo_CalibrateDebevec_getLambda	CVAPI(ExceptionStatus) photo_CalibrateDebevec_getLambda(cv::CalibrateDebevec *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getLambda(); END_WRAP }	cv::CalibrateDebevec::getLambda()	classcv_1_1CalibrateDebevec.html				1									

photo_CalibrateDebevec_setLambda	CVAPI(ExceptionStatus) photo_CalibrateDebevec_setLambda(cv::CalibrateDebevec *obj, float value) { BEGIN_WRAP obj->setLambda(value); END_WRAP }	cv::CalibrateDebevec::setLambda()	classcv_1_1CalibrateDebevec.html				1										
photo_CalibrateDebevec_getSamples	CVAPI(ExceptionStatus) photo_CalibrateDebevec_getSamples(cv::CalibrateDebevec *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getLambda(); END_WRAP }	cv::CalibrateDebevec::getLambda()	classcv_1_1CalibrateDebevec.html				1										
photo_CalibrateDebevec_setSamples	CVAPI(ExceptionStatus) photo_CalibrateDebevec_setSamples(cv::CalibrateDebevec *obj, int value) { BEGIN_WRAP obj->setLambda(value); END_WRAP }	cv::CalibrateDebevec::setLambda()	classcv_1_1CalibrateDebevec.html				1										
photo_CalibrateDebevec_getRandom	CVAPI(ExceptionStatus) photo_CalibrateDebevec_getRandom(cv::CalibrateDebevec *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getRandom() ? 1 : 0; END_WRAP }	cv::CalibrateDebevec::getRandom()	classcv_1_1CalibrateDebevec.html				1										
photo_CalibrateDebevec_setRandom	CVAPI(ExceptionStatus) photo_CalibrateDebevec_setRandom(cv::CalibrateDebevec *obj, int value) { BEGIN_WRAP obj->setRandom(value != 0); END_WRAP }	cv::CalibrateDebevec::setRandom()	classcv_1_1CalibrateDebevec.html				1										
photo_createCalibrateRobertson	CVAPI(ExceptionStatus) photo_createCalibrateRobertson(int max_iter, float threshold, cv::Ptr<cv::CalibrateRobertson> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::createCalibrateRobertson(max_iter, threshold)); END_WRAP }	cv::createCalibrateRobertson()	group__photo__hdr.html	Creates CalibrateRobertson object.	max_iter : maximal number of Gauss-Seidel solver iterations. threshold : target difference between results of two successive steps of the minimization.		2										
photo_Ptr_CalibrateRobertson_delete	CVAPI(ExceptionStatus) photo_Ptr_CalibrateRobertson_delete(cv::Ptr<cv::CalibrateRobertson> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::CalibrateRobertson	classcv_1_1CalibrateRobertson.html	Inverse camera response function is extracted for each brightness value by minimizing an objective function as linear system. This algorithm uses all image pixels. For more information see [204] .								•				
photo_Ptr_CalibrateRobertson_get	CVAPI(ExceptionStatus) photo_Ptr_CalibrateRobertson_get(cv::Ptr<cv::CalibrateRobertson> *obj, cv::CalibrateRobertson **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::CalibrateRobertson	classcv_1_1CalibrateRobertson.html	Inverse camera response function is extracted for each brightness value by minimizing an objective function as linear system. This algorithm uses all image pixels. For more information see [204] .									•				
photo_CalibrateRobertson_getMaxIter	CVAPI(ExceptionStatus) photo_CalibrateRobertson_getMaxIter(cv::CalibrateRobertson *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxIter(); END_WRAP }	cv::CalibrateRobertson::getMaxIter()	classcv_1_1CalibrateRobertson.html				1										
photo_CalibrateRobertson_setMaxIter	CVAPI(ExceptionStatus) photo_CalibrateRobertson_setMaxIter(cv::CalibrateRobertson *obj, int value) { BEGIN_WRAP obj->setMaxIter(value); END_WRAP }	cv::CalibrateRobertson::setMaxIter()	classcv_1_1CalibrateRobertson.html				1										
photo_CalibrateRobertson_getThreshold	CVAPI(ExceptionStatus) photo_CalibrateRobertson_getThreshold(cv::CalibrateRobertson *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getThreshold(); END_WRAP }	cv::CalibrateRobertson::getThreshold()	classcv_1_1CalibrateRobertson.html				1										
photo_CalibrateRobertson_setThreshold	CVAPI(ExceptionStatus) photo_CalibrateRobertson_setThreshold(cv::CalibrateRobertson *obj, float value) { BEGIN_WRAP obj->setThreshold(value); END_WRAP }	cv::CalibrateRobertson::setThreshold()	classcv_1_1CalibrateRobertson.html				1										
photo_CalibrateRobertson_getRadiance	CVAPI(ExceptionStatus) photo_CalibrateRobertson_getRadiance(cv::CalibrateRobertson *obj, cv::Mat *returnValue) { BEGIN_WRAP obj->getRadiance().copyTo(*returnValue); END_WRAP }	cv::CalibrateRobertson::getRadiance()	classcv_1_1CalibrateRobertson.html				1										
photo_CalibrateCRF_process	CVAPI(ExceptionStatus) photo_CalibrateCRF_process(cv::CalibrateCRF *obj, cv::Mat ** srcImgs, int srcImgsLength, cv::_OutputArray *dst, float* times) { BEGIN_WRAP // Build Mat Vector of images std::vector<cv::Mat> srcImgsVec(srcImgsLength); // Build float Vector of times std::vector<float> times_vec(srcImgsLength); for (int i = 0; i < srcImgsLength; i++) { srcImgsVec[i] = *srcImgs[i]; times_vec[i] = times[i]; } cv::CalibrateCRF::process()	classcv_1_1CalibrateCRF.html	Recovers inverse camera response.	src : vector of input images dst : 256x1 matrix with inverse camera response function times : vector of exposure time values for each image		3								
photo_createMergeDebevec	CVAPI(cv::Ptr<cv::MergeDebevec>*) photo_createMergeDebevec() { return clone(cv::createMergeDebevec()); }	cv::createMergeDebevec()	group__photo__hdr.html	Creates MergeDebevec object.			-1										
photo_Ptr_MergeDebevec_delete	CVAPI(void) photo_Ptr_MergeDebevec_delete(cv::Ptr<cv::MergeDebevec>* obj) { delete obj; }	cv::MergeDebevec	classcv_1_1MergeDebevec.html	The resulting HDR image is calculated as weighted average of the exposures considering exposure values and camera response. For more information see [55] .								•				
photo_Ptr_MergeDebevec_get	CVAPI(cv::MergeDebevec*) photo_Ptr_MergeDebevec_get(cv::Ptr<cv::MergeDebevec>* obj) { return obj->get(); }	cv::MergeDebevec	classcv_1_1MergeDebevec.html	The resulting HDR image is calculated as weighted average of the exposures considering exposure values and camera response. For more information see [55] .												•	
photo_createMergeMertens	CVAPI(cv::Ptr<cv::MergeMertens>*) photo_createMergeMertens() { return clone(cv::createMergeMertens()); }	cv::createMergeMertens()	group__photo__hdr.html	Creates MergeMertens object.	contrast_weight : contrast measure weight. See MergeMertens. saturation_weight : saturation measure weight exposure_weight : well-exposedness measure weight		-1										

photo_Ptr_MergeMertens_delete	CVAPI(void) photo_Ptr_MergeMertens_delete(cv::Ptr<cv::MergeMertens>* obj) { delete obj; }	cv::MergeMertens	classcv_1_1MergeMertens.html	Pixels are weighted using contrast, saturation and well-exposedness measures, than images are combined using laplacian pyramids. The resulting image weight is constructed as weighted average of contrast, saturation and well-exposedness measures. The resulting image doesn't require tonemapping and can be converted to 8-bit image by multiplying by 255, but it's recommended to apply gamma correction and/or linear tonemapping. For more information see [167] .									•		
photo_Ptr_MergeMertens_get	CVAPI(cv::MergeMertens*) photo_Ptr_MergeMertens_get(cv::Ptr<cv::MergeMertens>* obj) { return obj->get(); }	cv::MergeMertens	classcv_1_1MergeMertens.html	Pixels are weighted using contrast, saturation and well-exposedness measures, than images are combined using laplacian pyramids. The resulting image weight is constructed as weighted average of contrast, saturation and well-exposedness measures. The resulting image doesn't require tonemapping and can be converted to 8-bit image by multiplying by 255, but it's recommended to apply gamma correction and/or linear tonemapping. For more information see [167] .										•	
photo_MergeExposures_process	CVAPI(void) photo_MergeExposures_process(cv::MergeExposures* obj, cv::Mat** srcImgs, int srcImgsLength, cv::_OutputArray* dst, float* times, cv::_InputArray* response) { // Build Mat Vector of images std::vector<cv::Mat> srcImgsVec(srcImgsLength); // Build float Vector of times std::vector<float> times_vec(srcImgsLength); for (int i = 0; i < srcImgsLength; i++) { srcImgsVec[i] = *srcImgs[i]; times_vec[i] = times[i]; } obj->process(srcImgsVec, *dst, times_vec, *response); }	cv::MergeExposures::process()	classcv_1_1MergeExposures.html	Merges images.	src : vector of input images dst : result image times : vector of exposure time values for each image response : 256x1 matrix with inverse camera response function for each pixel value, it should have the same number of channels as images.	Implemented in cv::MergeRobertson, cv::MergeMertens, and cv::MergeDebevec.	-1									
photo_MergeMertens_process	CVAPI(void) photo_MergeMertens_process(cv::MergeMertens* obj, cv::Mat** srcImgs, int srcImgsLength, cv::_OutputArray* dst) { // Build Mat Vector of images std::vector<cv::Mat> srcImgsVec(srcImgsLength); // Build float Vector of times std::vector<float> times_vec(srcImgsLength); for (int i = 0; i < srcImgsLength; i++) { srcImgsVec[i] = *srcImgs[i]; } obj->process(srcImgsVec, *dst); }	cv::MergeMertens::process()	classcv_1_1MergeMertens.html	Merges images.	src : vector of input images dst : result image times : vector of exposure time values for each image response : 256x1 matrix with inverse camera response function for each pixel value, it should have the same number of channels as images.	Implements cv::MergeExposures.	-1									
photo_Tonemap_process	CVAPI(ExceptionStatus) photo_Tonemap_process(cv::Tonemap *obj, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP obj->process(*src, *dst); END_WRAP }	cv::Tonemap::process()	classcv_1_1Tonemap.html	Tonemaps image.	src : source image - CV_32FC3 Mat (float 32 bits 3 channels) dst : destination image - CV_32FC3 Mat with values in [0, 1] range		2									
photo_Tonemap_getGamma	CVAPI(ExceptionStatus) photo_Tonemap_getGamma(cv::Tonemap *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getGamma(); END_WRAP }	cv::Tonemap::getGamma()	classcv_1_1Tonemap.html				1									
photo_Tonemap_setGamma	CVAPI(ExceptionStatus) photo_Tonemap_setGamma(cv::Tonemap *obj, float gamma) { BEGIN_WRAP obj->setGamma(gamma); END_WRAP }	cv::Tonemap::setGamma()	classcv_1_1Tonemap.html				1									
photo_createTonemap	CVAPI(ExceptionStatus) photo_createTonemap(float gamma, cv::Ptr<cv::Tonemap> **returnValue) { BEGIN_WRAP const auto p = cv::createTonemap(gamma); *returnValue = clone(p); END_WRAP }	cv::createTonemap()	group__photo__hdr.html	Creates simple linear mapper with gamma correction.	gamma : positive value for gamma correction. Gamma value of 1.0 implies no correction, gamma equal to 2.2f is suitable for most displays. Generally gamma > 1 brightens the image and gamma < 1 darkens it.		1									
photo_Ptr_Tonemap_delete	CVAPI(ExceptionStatus) photo_Ptr_Tonemap_delete(cv::Ptr<cv::Tonemap> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::Tonemap	classcv_1_1Tonemap.html	Base class for tonemapping algorithms - tools that are used to map HDR image to 8-bit range.									•			
photo_Ptr_Tonemap_get	CVAPI(ExceptionStatus) photo_Ptr_Tonemap_get(cv::Ptr<cv::Tonemap> *ptr, cv::Tonemap **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::Tonemap	classcv_1_1Tonemap.html	Base class for tonemapping algorithms - tools that are used to map HDR image to 8-bit range.										•		
photo_TonemapDrago_getSaturation	CVAPI(ExceptionStatus) photo_TonemapDrago_getSaturation(cv::TonemapDrago *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getSaturation(); END_WRAP }	cv::TonemapDrago::getSaturation	classcv_1_1TonemapDrago.html													
photo_TonemapDrago_setSaturation	CVAPI(ExceptionStatus) photo_TonemapDrago_setSaturation(cv::TonemapDrago *obj, float saturation) { BEGIN_WRAP obj->setSaturation(saturation); END_WRAP }	cv::TonemapDrago::setSaturation()	classcv_1_1TonemapDrago.html				1									
photo_TonemapDrago_getBias	CVAPI(ExceptionStatus) photo_TonemapDrago_getBias(cv::TonemapDrago *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getBias(); END_WRAP }	cv::TonemapDrago::getBias()	classcv_1_1TonemapDrago.html				1									
photo_TonemapDrago_setBias	CVAPI(ExceptionStatus) photo_TonemapDrago_setBias(cv::TonemapDrago *obj, float bias) { BEGIN_WRAP obj->setBias(bias); END_WRAP }	cv::TonemapDrago::setBias()	classcv_1_1TonemapDrago.html				1									
photo_createTonemapDrago	CVAPI(ExceptionStatus) photo_createTonemapDrago(float gamma, float saturation, float bias, cv::Ptr<cv::TonemapDrago> **returnValue) { BEGIN_WRAP const auto p = cv::createTonemapDrago(gamma, saturation, bias); *returnValue = clone(p); END_WRAP }	cv::createTonemapDrago()	group__photo__hdr.html	Creates TonemapDrago object.	gamma : gamma value for gamma correction. See createTonemap saturation : positive saturation enhancement value. 1.0 preserves saturation, values greater than 1 increase saturation and values less than 1 decrease it. bias : value for bias function in [0, 1] range. Values from 0.7 to 0.9 usually give best results, default value is 0.85.		3									
photo_Ptr_TonemapDrago_delete	CVAPI(ExceptionStatus) photo_Ptr_TonemapDrago_delete(cv::Ptr<cv::TonemapDrago> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TonemapDrago	classcv_1_1TonemapDrago.html	Adaptive logarithmic mapping is a fast global tonemapping algorithm that scales the image in logarithmic domain. Since it's a global operator the same function is applied to all the pixels, it is controlled by the bias parameter. Optional saturation enhancement is possible as described in [71] . For more information see [58] .									•		

photo_Ptr_TonemapDrago_get	CVAPI(ExceptionStatus) photo_Ptr_TonemapDrago_get(cv::Ptr<cv::TonemapDrago> *ptr, cv::TonemapDrago **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TonemapDrago	classcv_1_1TonemapDrago.html	Adaptive logarithmic mapping is a fast global tonemapping algorithm that scales the image in logarithmic domain. Since it's a global operator the same function is applied to all the pixels, it is controlled by the bias parameter. Optional saturation enhancement is possible as described in [71] . For more information see [58] .										•		
photo_TonemapReinhard_getIntensity	CVAPI(ExceptionStatus) photo_TonemapReinhard_getIntensity(cv::TonemapReinhard *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getIntensity(); END_WRAP }	cv::TonemapReinhard::getIntensity()	classcv_1_1TonemapReinhard.html				1									
photo_TonemapReinhard_setIntensity	CVAPI(ExceptionStatus) photo_TonemapReinhard_setIntensity(cv::TonemapReinhard *obj, float intensity) { BEGIN_WRAP obj->setIntensity(intensity); END_WRAP }	cv::TonemapReinhard::setIntensity()	classcv_1_1TonemapReinhard.html				1									
photo_TonemapReinhard_getLightAdaptation	CVAPI(ExceptionStatus) photo_TonemapReinhard_getLightAdaptation(cv::TonemapReinhard *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getLightAdaptation(); END_WRAP }	cv::TonemapReinhard::getLightAdaptation()	classcv_1_1TonemapReinhard.html				1									
photo_TonemapReinhard_setLightAdaptation	CVAPI(ExceptionStatus) photo_TonemapReinhard_setLightAdaptation(cv::TonemapReinhard *obj, float light_adapt) { BEGIN_WRAP obj->setLightAdaptation(light_adapt); END_WRAP }	cv::TonemapReinhard::setLightAdaptation()	classcv_1_1TonemapReinhard.html				1									
photo_TonemapReinhard_getColorAdaptation	CVAPI(ExceptionStatus) photo_TonemapReinhard_getColorAdaptation(cv::TonemapReinhard *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getColorAdaptation(); END_WRAP }	cv::TonemapReinhard::getColorAdaptation()	classcv_1_1TonemapReinhard.html				1									
photo_TonemapReinhard_setColorAdaptation	CVAPI(ExceptionStatus) photo_TonemapReinhard_setColorAdaptation(cv::TonemapReinhard *obj, float color_adapt) { BEGIN_WRAP obj->setColorAdaptation(color_adapt); END_WRAP }	cv::TonemapReinhard::setColorAdaptation()	classcv_1_1TonemapReinhard.html				1									
photo_createTonemapReinhard	CVAPI(ExceptionStatus) photo_createTonemapReinhard(float gamma, float intensity, float light_adapt, float color_adapt, cv::Ptr<cv::TonemapReinhard> **returnValue) { BEGIN_WRAP const auto p = cv::createTonemapReinhard(gamma, intensity, light_adapt, color_adapt); *returnValue = clone(p); END_WRAP }	cv::createTonemapReinhard()	group__photo__hdr.html	Creates TonemapReinhard object.	gamma : gamma value for gamma correction. See createTonemap intensity : result intensity in [-8, 8] range. Greater intensity produces brighter results. light_adapt : light adaptation in [0, 1] range. If 1 adaptation is based only on pixel value, if 0 it's global, otherwise it's a weighted mean of this two cases. color_adapt : chromatic adaptation in [0, 1] range. If 1 channels are treated independently, if 0 adaptation level is the same for each channel.		4									
photo_Ptr_TonemapReinhard_delete	CVAPI(ExceptionStatus) photo_Ptr_TonemapReinhard_delete(cv::Ptr<cv::TonemapReinhard> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TonemapReinhard	classcv_1_1TonemapReinhard.html	This is a global tonemapping operator that models human visual system. Mapping function is controlled by adaptation parameter, that is computed using light adaptation and color adaptation. For more information see [201] .									•		
photo_Ptr_TonemapReinhard_get	CVAPI(ExceptionStatus) photo_Ptr_TonemapReinhard_get(cv::Ptr<cv::TonemapReinhard> *ptr, cv::TonemapReinhard **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TonemapReinhard	classcv_1_1TonemapReinhard.html	This is a global tonemapping operator that models human visual system. Mapping function is controlled by adaptation parameter, that is computed using light adaptation and color adaptation. For more information see [201] .										•		
photo_TonemapMantiuk_getScale	CVAPI(ExceptionStatus) photo_TonemapMantiuk_getScale(cv::TonemapMantiuk *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getScale(); END_WRAP }	cv::TonemapMantiuk::getScale()	classcv_1_1TonemapMantiuk.html				1									
photo_TonemapMantiuk_setScale	CVAPI(ExceptionStatus) photo_TonemapMantiuk_setScale(cv::TonemapMantiuk *obj, float scale) { BEGIN_WRAP obj->setScale(scale); END_WRAP }	cv::TonemapMantiuk::setScale()	classcv_1_1TonemapMantiuk.html				1									
photo_TonemapMantiuk_getSaturation	CVAPI(ExceptionStatus) photo_TonemapMantiuk_getSaturation(cv::TonemapMantiuk *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getSaturation(); END_WRAP }	cv::TonemapMantiuk::getSaturation()	classcv_1_1TonemapMantiuk.html				1									
photo_TonemapMantiuk_setSaturation	CVAPI(ExceptionStatus) photo_TonemapMantiuk_setSaturation(cv::TonemapMantiuk *obj, float saturation) { BEGIN_WRAP obj->setSaturation(saturation); END_WRAP }	cv::TonemapMantiuk::setSaturation()	classcv_1_1TonemapMantiuk.html				1									
photo_createTonemapMantiuk	CVAPI(ExceptionStatus) photo_createTonemapMantiuk(float gamma, float scale, float saturation, cv::Ptr<cv::TonemapMantiuk> **returnValue) { BEGIN_WRAP const auto p = cv::createTonemapMantiuk(gamma, scale, saturation); *returnValue = clone(p); END_WRAP }	cv::createTonemapMantiuk()	group__photo__hdr.html	Creates TonemapMantiuk object.	gamma : gamma value for gamma correction. See createTonemap scale : contrast scale factor. HVS response is multiplied by this parameter, thus compressing dynamic range. Values from 0.6 to 0.9 produce best results. saturation : saturation enhancement value. See createTonemapDrago		3									
photo_Ptr_TonemapMantiuk_delete	CVAPI(ExceptionStatus) photo_Ptr_TonemapMantiuk_delete(cv::Ptr<cv::TonemapMantiuk> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TonemapMantiuk	classcv_1_1TonemapMantiuk.html	This algorithm transforms image to contrast using gradients on all levels of gaussian pyramid, transforms contrast values to HVS response and scales the response. After this the image is reconstructed from new contrast values. For more information see [161] .									•		
photo_Ptr_TonemapMantiuk_get	CVAPI(ExceptionStatus) photo_Ptr_TonemapMantiuk_get(cv::Ptr<cv::TonemapMantiuk> *ptr, cv::TonemapMantiuk **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TonemapMantiuk	classcv_1_1TonemapMantiuk.html	This algorithm transforms image to contrast using gradients on all levels of gaussian pyramid, transforms contrast values to HVS response and scales the response. After this the image is reconstructed from new contrast values. For more information see [161] .										•		

shape_ShapeDistanceExtractor_computeDistan	CVAPI(ExceptionStatus) shape_ShapeDistanceExtracto r_computeDistance(cv::ShapeDistanceExtractor *obj, cv::_InputArray *contour1, cv::_InputArray *contour2, float *returnValue) { BEGIN_WRAP *returnValue = obj->computeDistance(*contour1, *contour2); END_WRAP }	cv::ShapeDistanceExtractor::computeDistance()	classcv_1_1ShapeDistanceExtractor.html	Compute the shape distance between two shapes defined by its contours.	contour1 : Contour defining first shape. contour2 : Contour defining second shape.		2									
shape_Ptr_ShapeContextDistanceExtract_delet	CVAPI(ExceptionStatus) shape_Ptr_ShapeContextDista nceExtractor_delete(cv::Ptr<cv::ShapeContextDist anceExtractor> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ShapeContextDistanceExtractor	classcv_1_1ShapeContextDistanceExtractor.html	Implementation of the Shape Context descriptor and matching algorithm. proposed by Belongie et al. in "Shape Matching and Object Recognition Using Shape Contexts" (PAMI 2002). This implementation is packaged in a generic scheme, in order to allow you the implementation of the common variations of the original pipeline.								•		
shape_Ptr_ShapeContextDistanceExtract_get	CVAPI(ExceptionStatus) shape_Ptr_ShapeContextDista nceExtractor_get(cv::Ptr<cv::ShapeContextDist anceExtractor> *obj, cv::ShapeContextDistanceExtr actor **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::ShapeContextDistanceExtractor	classcv_1_1ShapeContextDistanceExtractor.html	Implementation of the Shape Context descriptor and matching algorithm. proposed by Belongie et al. in "Shape Matching and Object Recognition Using Shape Contexts" (PAMI 2002). This implementation is packaged in a generic scheme, in order to allow you the implementation of the common variations of the original pipeline.									•	
shape_ShapeContextDistanceExtract_setAngul	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setAngularBins(cv::ShapeContextDistanceExtr actor *obj, int val) { BEGIN_WRAP obj->setAngularBins(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setAngularBin s()	classcv_1_1ShapeContextDistanceExtractor.html	Establish the number of angular bins for the Shape Context Descriptor used in the shape matching pipeline.	nAngularBins : The number of angular bins in the shape context descriptor.		1									
shape_ShapeContextDistanceExtract_getAngul	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getAngularBins(cv::ShapeContextDistanceExtr actor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getAngularBins(); END_WRAP }	cv::ShapeContextDistanceExtractor::getAngularBin s()	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_setRadial	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setRadialBins(cv::ShapeContextDistanceExtr actor *obj, int val) { BEGIN_WRAP obj->setRadialBins(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setRadialBins( )	classcv_1_1ShapeContextDistanceExtractor.html	Establish the number of radial bins for the Shape Context Descriptor used in the shape matching pipeline.	nRadialBins : The number of radial bins in the shape context descriptor.		1									
shape_ShapeContextDistanceExtract_getRadia	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getRadialBins(cv::ShapeContextDistanceExtr actor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getRadialBins(); END_WRAP }	cv::ShapeContextDistanceExtractor::getRadialBins( )	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_setInnerr	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setInnerRadius(cv::ShapeContextDistanceExtr actor *obj, float val) { BEGIN_WRAP obj->setInnerRadius(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setInnerRadiu s()	classcv_1_1ShapeContextDistanceExtractor.html	Set the inner radius of the shape context descriptor.	innerRadius : The value of the inner radius.		1									
shape_ShapeContextDistanceExtract_getInnerr	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getInnerRadius(cv::ShapeContextDistanceExtr actor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getInnerRadius(); END_WRAP }	cv::ShapeContextDistanceExtractor::getInnerRadiu s()	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_setOuterr	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setOuterRadius(cv::ShapeContextDistanceExtr actor *obj, float val) { BEGIN_WRAP obj->setOuterRadius(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setOuterRadiu s()	classcv_1_1ShapeContextDistanceExtractor.html	Set the outer radius of the shape context descriptor.	outerRadius : The value of the outer radius.		1									
shape_ShapeContextDistanceExtract_getOuterr	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getOuterRadius(cv::ShapeContextDistanceExtr actor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getOuterRadius(); END_WRAP }	cv::ShapeContextDistanceExtractor::getOuterRadiu s()	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_setRotati	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setRotationInvarian t(cv::ShapeContextDistanceExtr actor *obj, int val) { BEGIN_WRAP obj ->setRotationInvariant(val != 0); END_WRAP }	cv::ShapeContextDistanceExtractor::setRotationInv ariant()	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_getRotati	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getRotationInvarian t(cv::ShapeContextDistanceExtr actor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getRotationInvariant() ? 1 : 0; END_WRAP }	cv::ShapeContextDistanceExtractor::getRotationIn variant()	classcv_1_1ShapeContextDistanceExtractor.html				1									
shape_ShapeContextDistanceExtract_setShape	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_setShapeContextWe ight(cv::ShapeContextDistanceExtr actor *obj, float val) { BEGIN_WRAP obj->setShapeContextWeight(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setShapeCont extWeight()	classcv_1_1ShapeContextDistanceExtractor.html	Set the weight of the shape context distance in the final value of the shape distance. The shape context distance between two shapes is defined as the symmetric sum of shape context matching costs over best matching points. The final value of the shape distance is a user-defined linear combination of the shape context distance, an image appearance distance, and a bending energy.	shapeContextWeight : The weight of the shape context distance in the final distance value.		1									
shape_ShapeContextDistanceExtract_getShape	CVAPI(ExceptionStatus) shape_ShapeContextDistance Extractor_getShapeContextW eight(cv::ShapeContextDistanceExtr actor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getShapeContextWeight(); END_WRAP }	cv::ShapeContextDistanceExtractor::getShapeCont extWeight()	classcv_1_1ShapeContextDistanceExtractor.html				1									

shape_ShapeContextDistanceExtract_setImage	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_setImageAppearanceWeight(cv::ShapeContextDistanceExtractor *obj, float val) { BEGIN_WRAP obj->setImageAppearanceWeight(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setImageAppearanceWeight()	classcv_1_1ShapeContextDistanceExtractor.html	Set the weight of the Image Appearance cost in the final value of the shape distance. The image appearance cost is defined as the sum of squared brightness differences in Gaussian windows around corresponding image points. The final value of the shape distance is a user-defined linear combination of the shape context distance, an image appearance distance, and a bending energy. If this value is set to a number different from 0, is mandatory to set the images that correspond to each shape.	imageAppearanceWeight : The weight of the appearance cost in the final distance value.		1								
shape_ShapeContextDistanceExtract_getImage	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_getImageAppearanceWeight(cv::ShapeContextDistanceExtractor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getImageAppearanceWeight(); END_WRAP }	cv::ShapeContextDistanceExtractor::getImageAppearanceWeight()	classcv_1_1ShapeContextDistanceExtractor.html				1										
shape_ShapeContextDistanceExtract_setBendi	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_setBendingEnergyWeight(cv::ShapeContextDistanceExtractor *obj, float val) { BEGIN_WRAP obj->setBendingEnergyWeight(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setBendingEnergyWeight()	classcv_1_1ShapeContextDistanceExtractor.html	Set the weight of the Bending Energy in the final value of the shape distance. The bending energy definition depends on what transformation is being used to align the shapes. The final value of the shape distance is a user-defined linear combination of the shape context distance, an image appearance distance, and a bending energy.	bendingEnergyWeight : The weight of the Bending Energy in the final distance value.		1										
shape_ShapeContextDistanceExtract_getBendi	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_getBendingEnergyWeight(cv::ShapeContextDistanceExtractor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getBendingEnergyWeight(); END_WRAP }	cv::ShapeContextDistanceExtractor::getBendingEnergyWeight()	classcv_1_1ShapeContextDistanceExtractor.html				1										
shape_ShapeContextDistanceExtract_setImage	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_setImages(cv::ShapeContextDistanceExtractor *obj, cv::_InputArray *image1, cv::_InputArray *image2) { BEGIN_WRAP obj->setImages(*image1, *image2); END_WRAP }	cv::ShapeContextDistanceExtractor::setImages()	classcv_1_1ShapeContextDistanceExtractor.html	Set the images that correspond to each shape. This images are used in the calculation of the Image Appearance cost.	image1 : Image corresponding to the shape defined by contours1. image2 : Image corresponding to the shape defined by contours2.		2										
shape_ShapeContextDistanceExtract_getImage	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_getImages(cv::ShapeContextDistanceExtractor *obj, cv::_OutputArray *image1, cv::_OutputArray *image2) { BEGIN_WRAP obj->getImages(*image1, *image2); END_WRAP }	cv::ShapeContextDistanceExtractor::getImages()	classcv_1_1ShapeContextDistanceExtractor.html				2										
shape_ShapeContextDistanceExtract_setIterati	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_setIterations(cv::ShapeContextDistanceExtractor *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setIterations()	classcv_1_1ShapeContextDistanceExtractor.html				1										
shape_ShapeContextDistanceExtract_getIterat	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_getIterations(cv::ShapeContextDistanceExtractor *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::ShapeContextDistanceExtractor::getIterations()	classcv_1_1ShapeContextDistanceExtractor.html				1										
shape_ShapeContextDistanceExtract_setStdDe	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_setStdDev(cv::ShapeContextDistanceExtractor *obj, float val) { BEGIN_WRAP obj->setStdDev(val); END_WRAP }	cv::ShapeContextDistanceExtractor::setStdDev()	classcv_1_1ShapeContextDistanceExtractor.html	Set the value of the standard deviation for the Gaussian window for the image appearance cost.	sigma : Standard Deviation.		1										
shape_ShapeContextDistanceExtract_getStdDe	CVAPI(ExceptionStatus) shape_ShapeContextDistanceExtractor_getStdDev(cv::ShapeContextDistanceExtractor *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getStdDev(); END_WRAP }	cv::ShapeContextDistanceExtractor::getStdDev()	classcv_1_1ShapeContextDistanceExtractor.html				1										
shape_createShapeContextDistanceExtract	CVAPI(ExceptionStatus) shape_createShapeContextDistanceExtractor(int nAngularBins, int nRadialBins, float innerRadius, float outerRadius, int iterations/*, const Ptr<HistogramCostExtractor> &comparer = createChiHistogramCostExtractor(), const Ptr<ShapeTransformer> &transformer = createThinPlateSplineShapeTransformer()*/, cv::Ptr<cv::ShapeContextDistanceExtractor> **returnValue) { BEGIN_WRAP const auto p = cv::createShapeContextDistanceExtractor(nAngularBins,	cv::createShapeContextDistanceExtractor()	group__shape.html	Examples: modules/shape/samples/shape_example.cpp.			5										
shape_Ptr_HausdorffDistanceExtractor_delete	CVAPI(ExceptionStatus) shape_Ptr_HausdorffDistanceExtractor_delete(cv::Ptr<cv::HausdorffDistanceExtractor> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::HausdorffDistanceExtractor	classcv_1_1HausdorffDistanceExtractor.html	A simple Hausdorff distance measure between shapes defined by contours. according to the paper "Comparing Images using the Hausdorff distance." by D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. (PAMI 1993). :								•			
shape_Ptr_HausdorffDistanceExtractor_get	CVAPI(ExceptionStatus) shape_Ptr_HausdorffDistanceExtractor_get(cv::Ptr<cv::HausdorffDistanceExtractor> *obj, cv::HausdorffDistanceExtractor **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::HausdorffDistanceExtractor	classcv_1_1HausdorffDistanceExtractor.html	A simple Hausdorff distance measure between shapes defined by contours. according to the paper "Comparing Images using the Hausdorff distance." by D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. (PAMI 1993). :									•		
shape_HausdorffDistanceExtractor_setDistanc	CVAPI(ExceptionStatus) shape_HausdorffDistanceExtractor_setDistanceFlag(cv::HausdorffDistanceExtractor *obj, int val) { BEGIN_WRAP obj->setDistanceFlag(val); END_WRAP }	cv::HausdorffDistanceExtractor::setDistanceFlag()	classcv_1_1HausdorffDistanceExtractor.html	Set the norm used to compute the Hausdorff value between two shapes. It can be L1 or L2 norm.	distanceFlag : Flag indicating which norm is used to compute the Hausdorff distance (NORM_L1, NORM_L2).		1										

shape_HausdorffDistanceExtractor_getDistanc	CVAPI(ExceptionStatus) shape_HausdorffDistanceExtra ctor_getDistanceFlag(cv::HausdorffDistanceExtracto r *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj- >getDistanceFlag(); END_WRAP }	cv::HausdorffDistanceExtractor::getDistanceFlag()	classcv_1_1HausdorffDistanceExtra ctor.html				1										
shape_HausdorffDistanceExtractor_setRankPro	CVAPI(ExceptionStatus) shape_HausdorffDistanceExtra ctor_setRankProportion(cv::HausdorffDistanceExtracto r *obj, float val) { BEGIN_WRAP obj- >setRankProportion(val); END_WRAP }	cv::HausdorffDistanceExtractor::setRankProportion ()	classcv_1_1HausdorffDistanceExtra ctor.html	This method sets the rank proportion (or fractional value) that establish the Kth ranked value of the partial Hausdorff distance. Experimentally had been shown that 0.6 is a good value to compare shapes.	rankProportion : fractional value (between 0 and 1).		1										
shape_HausdorffDistanceExtractor_getRankPro	CVAPI(ExceptionStatus) shape_HausdorffDistanceExtra ctor_getRankProportion(cv::HausdorffDistanceExtracto r *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj- >getRankProportion(); END_WRAP }	cv::HausdorffDistanceExtractor::getRankProportion ()	classcv_1_1HausdorffDistanceExtra ctor.html				1										
shape_createHausdorffDistanceExtractor	CVAPI(ExceptionStatus) shape_createHausdorffDistanc eExtractor(int distanceFlag, float rankProp, cv::Ptr<cv::HausdorffDistanc eExtractor> **returnValue) { BEGIN_WRAP const auto p = cv::createHausdorffDistanceE xtractor(distanceFlag, rankProp); *returnValue = clone(p); END_WRAP }	cv::createHausdorffDistanceExtractor()	group__shape.html				2										
stitching_Stitcher_create	CVAPI(ExceptionStatus) stitching_Stitcher_create(int mode, cv::Ptr<cv::Stitcher> **returnValue) { BEGIN_WRAP const auto ptr = cv::Stitcher::create(static_ca st<cv::Stitcher::Mode>(mode)); *returnValue = clone(ptr); END_WRAP }	cv::Stitcher::create()	classcv_1_1Stitcher.html	Creates a Stitcher configured in one of the stitching modes.	mode : Scenario for stitcher operation. This is usually determined by source of images to stitch and their transformation. Default parameters will be chosen for operation in given scenario.		1										
stitching_Ptr_Stitcher_delete	CVAPI(ExceptionStatus) stitching_Ptr_Stitcher_delete(cv::Ptr<cv::Stitcher> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::Stitcher	classcv_1_1Stitcher.html	High level image stitcher. It's possible to use this class without being aware of the entire stitching pipeline. However, to be able to achieve higher stitching stability and quality of the final images at least being familiar with the theory is recommended. Note A basic example on image stitching can be found at opencv_source_code/samples/cpp/stitching.cpp A basic example on image stitching in Python can be found at opencv_source_code/samples/python/stitching.py A detailed example on image stitching can be found at opencv_source_code/samples/cpp/stitching_detailed.cpp							•		
stitching_Ptr_Stitcher_get	CVAPI(ExceptionStatus) stitching_Ptr_Stitcher_get(cv: :Ptr<cv::Stitcher> *obj, cv::Stitcher **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::Stitcher	classcv_1_1Stitcher.html	High level image stitcher. It's possible to use this class without being aware of the entire stitching pipeline. However, to be able to achieve higher stitching stability and quality of the final images at least being familiar with the theory is recommended. Note A basic example on image stitching can be found at opencv_source_code/samples/cpp/stitching.cpp A basic example on image stitching in Python can be found at opencv_source_code/samples/python/stitching.py A detailed example on image stitching can be found at opencv_source_code/samples/cpp/stitching_detailed.cpp								•	
stitching_Stitcher_registrationResol	CVAPI(ExceptionStatus) stitching_Stitcher_registration Resol(cv::Stitcher *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj- >registrationResol(); END_WRAP }	cv::Stitcher::registrationResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_setRegistrationResol	CVAPI(ExceptionStatus) stitching_Stitcher_setRegistra tionResol(cv::Stitcher *obj, const double resol_mpx) { BEGIN_WRAP obj- >setRegistrationResol(resol_ mpx); END_WRAP }	cv::Stitcher::setRegistrationResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_seamEstimationResol	CVAPI(ExceptionStatus) stitching_Stitcher_seamEstim ationResol(cv::Stitcher *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj- >seamEstimationResol(); END_WRAP }	cv::Stitcher::seamEstimationResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_setSeamEstimationResol	CVAPI(ExceptionStatus) stitching_Stitcher_setSeamEst imationResol(cv::Stitcher *obj, const double resol_mpx) { BEGIN_WRAP obj- >setSeamEstimationResol(res ol_mpx); END_WRAP }	cv::Stitcher::setSeamEstimationResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_compositingResol	CVAPI(ExceptionStatus) stitching_Stitcher_compositing Resol(cv::Stitcher *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj- >compositingResol(); END_WRAP }	cv::Stitcher::compositingResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_setCompositingResol	CVAPI(ExceptionStatus) stitching_Stitcher_setComposi tingResol(cv::Stitcher *obj, const double resol_mpx) { BEGIN_WRAP obj- >setCompositingResol(resol_ mpx); END_WRAP }	cv::Stitcher::setCompositingResol()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_panoConfidenceThresh	CVAPI(ExceptionStatus) stitching_Stitcher_panoConfid enceThresh(cv::Stitcher *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj- >panoConfidenceThresh(); END_WRAP }	cv::Stitcher::panoConfidenceThresh()	classcv_1_1Stitcher.html				1										
stitching_Stitcher_setPanoConfidenceThresh	CVAPI(ExceptionStatus) stitching_Stitcher_setPanoCon fidenceThresh(cv::Stitcher *obj, const double conf_thresh) { BEGIN_WRAP obj- >setPanoConfidenceThresh(co nf_thresh); END_WRAP }	cv::Stitcher::setPanoConfidenceThresh()	classcv_1_1Stitcher.html				1										

stitching_Stitcher_waveCorrection	CVAPI(ExceptionStatus) stitching_Stitcher_waveCorrection(cv::Stitcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->waveCorrection() ? 1 : 0; END_WRAP }	cv::Stitcher::waveCorrection()	classcv_1_1Stitcher.html				1									
stitching_Stitcher_setWaveCorrection	CVAPI(ExceptionStatus) stitching_Stitcher_setWaveCorrection(cv::Stitcher *obj, const int flag) { BEGIN_WRAP obj->setWaveCorrection(flag != 0); END_WRAP }	cv::Stitcher::setWaveCorrection()	classcv_1_1Stitcher.html				1									
stitching_Stitcher_waveCorrectKind	CVAPI(ExceptionStatus) stitching_Stitcher_waveCorrectKind(cv::Stitcher *obj, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->waveCorrectKind()); END_WRAP }	cv::Stitcher::waveCorrectKind()	classcv_1_1Stitcher.html				1									
stitching_Stitcher_setWaveCorrectKind	CVAPI(ExceptionStatus) stitching_Stitcher_setWaveCorrectKind(cv::Stitcher *obj, int kind) { BEGIN_WRAP obj->setWaveCorrectKind(static_cast<cv::detail::WaveCorrectKind>(kind)); END_WRAP }	cv::Stitcher::setWaveCorrectKind()	classcv_1_1Stitcher.html				1									
stitching_Stitcher_estimateTransform_InputArr	CVAPI(ExceptionStatus) stitching_Stitcher_estimateTransform_InputArray1(cv::Stitcher *obj, cv::_InputArray *images, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->estimateTransform(*images)); END_WRAP }	cv::Stitcher::estimateTransform()	classcv_1_1Stitcher.html	These functions try to match the given images and to estimate rotations of each camera.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional).	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	1								
stitching_Stitcher_estimateTransform_InputArr	CVAPI(ExceptionStatus) stitching_Stitcher_estimateTransform_InputArray2(cv::Stitcher *obj, cv::_InputArray *images, const CvRect **rois, const int roisSize1, const int *roisSize2, int *returnValue) { BEGIN_WRAP std::vector<std::vector<cv::Rect> > roisVec; toVec(rois, roisSize1, roisSize2, roisVec); *returnValue = static_cast<int>(obj->estimateTransform(*images, roisVec)); END_WRAP }	cv::Stitcher::estimateTransform()	classcv_1_1Stitcher.html	These functions try to match the given images and to estimate rotations of each camera.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional).	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	2								
stitching_Stitcher_estimateTransform_MatArra	CVAPI(ExceptionStatus) stitching_Stitcher_estimateTransform_MatArray1(cv::Stitcher *obj, const cv::Mat **images, const int imagesSize, int *returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesSize, imagesVec); *returnValue = static_cast<int>(obj->estimateTransform(imagesVec)); END_WRAP }	cv::Stitcher::estimateTransform()	classcv_1_1Stitcher.html	These functions try to match the given images and to estimate rotations of each camera.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional).	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	1								
stitching_Stitcher_estimateTransform_MatArra	CVAPI(ExceptionStatus) stitching_Stitcher_estimateTransform_MatArray2(cv::Stitcher *obj, const cv::Mat **images, const int imagesSize, const CvRect **rois, const int roisSize1, const int *roisSize2, int *returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesSize, imagesVec); std::vector<std::vector<cv::Rect> > roisVec; toVec(rois, roisSize1, roisSize2, roisVec); *returnValue = static_cast<int>(obj->estimateTransform(imagesVec, roisVec)); END_WRAP }	cv::Stitcher::estimateTransform()	classcv_1_1Stitcher.html	These functions try to match the given images and to estimate rotations of each camera.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional).	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	2								
stitching_Stitcher_composePanorama1	CVAPI(ExceptionStatus) stitching_Stitcher_composePanorama1(cv::Stitcher *obj, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->composePanorama(*pano)); END_WRAP }	cv::Stitcher::composePanorama()	classcv_1_1Stitcher.html	These functions try to compose the given images (or images stored internally from the other function calls) into the final pano under the assumption that the image transformations were estimated before.	images : Input images. pano : Final pano.	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	1								
stitching_Stitcher_composePanorama2_InputA	CVAPI(ExceptionStatus) stitching_Stitcher_composePanorama2_InputArray(cv::Stitcher *obj, cv::_InputArray *images, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->composePanorama(*images, *pano)); END_WRAP }	cv::Stitcher::composePanorama()	classcv_1_1Stitcher.html	These functions try to compose the given images (or images stored internally from the other function calls) into the final pano under the assumption that the image transformations were estimated before.	images : Input images. pano : Final pano.	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	2								
stitching_Stitcher_composePanorama2_MatArr	CVAPI(ExceptionStatus) stitching_Stitcher_composePanorama2_MatArray(cv::Stitcher *obj, const cv::Mat **images, const int imagesSize, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesSize, imagesVec); *returnValue = static_cast<int>(obj->composePanorama(imagesVec, *pano)); END_WRAP }	cv::Stitcher::composePanorama()	classcv_1_1Stitcher.html	These functions try to compose the given images (or images stored internally from the other function calls) into the final pano under the assumption that the image transformations were estimated before.	images : Input images. pano : Final pano.	NoteUse the functions only if you're aware of the stitching pipeline, otherwise use Stitcher::stitch.	2								
stitching_Stitcher_stitch1_InputArray	CVAPI(ExceptionStatus) stitching_Stitcher_stitch1_InputArray(cv::Stitcher *obj, cv::_InputArray *images, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP *returnValue = static_cast<int>(obj->stitch(*images, *pano)); END_WRAP }	cv::Stitcher::stitch()	classcv_1_1Stitcher.html	These functions try to stitch the given images.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional). pano : Final pano.		2								

stitching_Stitcher_stitch1_MatArray	CVAPI(ExceptionStatus) stitching_Stitcher_stitch1_MatArray(cv::Stitcher *obj, const cv::Mat **images, const int imagesSize, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesSize, imagesVec); *returnValue = static_cast<int>(obj->stitch(imagesVec, *pano)); END_WRAP }	cv::Stitcher::stitch()	classcv_1_1Stitcher.html	These functions try to stitch the given images.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional). pano : Final pano.		2									
stitching_Stitcher_stitch2_InputArray	CVAPI(ExceptionStatus) stitching_Stitcher_stitch2_InputArray(cv::Stitcher *obj, cv::_InputArray *images, const CvRect **rois, const int roisSize1, int *roisSize2, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP std::vector<std::vector<cv::Rect> > roisVec; toVec(rois, roisSize1, roisSize2, roisVec); *returnValue = static_cast<int>(obj->stitch(*images, roisVec, *pano)); END_WRAP }	cv::Stitcher::stitch()	classcv_1_1Stitcher.html	These functions try to stitch the given images.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional). pano : Final pano.		3									
stitching_Stitcher_stitch2_MatArray	CVAPI(ExceptionStatus) stitching_Stitcher_stitch2_MatArray(cv::Stitcher *obj, const cv::Mat **images, const int imagesSize, const CvRect **rois, const int roisSize1, int *roisSize2, cv::_OutputArray *pano, int *returnValue) { BEGIN_WRAP std::vector<cv::Mat> imagesVec; toVec(images, imagesSize, imagesVec); std::vector<std::vector<cv::Rect> > roisVec; toVec(rois, roisSize1, roisSize2, roisVec); *returnValue = static_cast<int>(obj->stitch(imagesVec, roisVec,	cv::Stitcher::stitch()	classcv_1_1Stitcher.html	These functions try to stitch the given images.	images : Input images. masks : Masks for each input image specifying where to look for keypoints (optional). pano : Final pano.		3									
stitching_Stitcher_component	CVAPI(ExceptionStatus) stitching_Stitcher_component(cv::Stitcher *obj, std::vector<int>* returnValue) { BEGIN_WRAP const auto component = obj->component(); std::copy(component.begin(), component.end(), std::back_inserter(*returnValue)); END_WRAP }	cv::Stitcher::component()	classcv_1_1Stitcher.html				3									
stitching_Stitcher_workScale	CVAPI(ExceptionStatus) stitching_Stitcher_workScale(cv::Stitcher *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->workScale(); END_WRAP }	cv::Stitcher::workScale()	classcv_1_1Stitcher.html				1									
stitching_computeImageFeatures1	CVAPI(ExceptionStatus) stitching_computeImageFeatures1(cv::Feature2D *featuresFinder, cv::Mat **images, int imagesLength, std::vector<cv::detail::ImageFeatures> *featuresVec, cv::Mat **masks) { BEGIN_WRAP // Do not free Feature2D const cv::Ptr<cv::Feature2D> featuresFinderPtr(featuresFinder, [](cv::Feature2D*){}); std::vector<cv::Mat> imagesVec(imagesLength); for (int i = 0; i < imagesLength; i++) { imagesVec[i] =	cv::detail::computeImageFeatures()	group__stitching__match.html	Examples: samples/cpp/stitching_detailed.cpp.			4									
stitching_computeImageFeatures2	CVAPI(ExceptionStatus) stitching_computeImageFeatures2(cv::Feature2D *featuresFinder, cv::_InputArray *image, detail_ImageFeatures *features, cv::_InputArray *mask) { BEGIN_WRAP // Do not free Feature2D const cv::Ptr<cv::Feature2D> featuresFinderPtr(featuresFinder, [](cv::Feature2D*){}); cv::detail::ImageFeatures rawFeature; cv::detail::computeImageFeatures(featuresFinderPtr, *image, rawFeature, emtity(mask));	cv::detail::computeImageFeatures()	group__stitching__match.html	Examples: samples/cpp/stitching_detailed.cpp.			6									
stitching_FeaturesMatcher_apply	CVAPI(ExceptionStatus) stitching_FeaturesMatcher_apply(cv::detail::FeaturesMatcher* obj, detail_ImageFeatures *features1, detail_ImageFeatures *features2, int *out_src_img_idx, int *out_dst_img_idx, std::vector<cv::DMatch> *out_matches, std::vector<uchar> *out_inliers_mask, int *out_num_inliers, cv::Mat *out_H, double *out_confidence) { BEGIN_WRAP cv::detail::ImageFeatures features1Cpp(features1->img_idx, cpp(features1->img_size),	cv::detail::FeaturesMatcher	classcv_1_1detail_1_1FeaturesMatcher.html	Feature matchers base class.					•							
stitching_FeaturesMatcher_apply2	CVAPI(ExceptionStatus) stitching_FeaturesMatcher_apply2(cv::detail::FeaturesMatcher* obj, detail_ImageFeatures* features, int featuresSize, cv::Mat *mask, std::vector<int> *out_src_img_idx, std::vector<int> *out_dst_img_idx, std::vector< std::vector<cv::DMatch> > *out_matches, std::vector< std::vector<uchar> > *out_inliers_mask, std::vector<int> *out_num_inliers, std::vector<cv::Mat> *out_H, std::vector<double> *out_confidence) { BEGIN_WRAP	cv::detail::FeaturesMatcher	classcv_1_1detail_1_1FeaturesMatcher.html	Feature matchers base class.					•							

stitching_FeaturesMatcher_isThreadSafe	CVAPI(ExceptionStatus) stitching_FeaturesMatcher_isThreadSafe(cv::detail::FeaturesMatcher* obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->isThreadSafe() ? 1 : 0; END_WRAP }	cv::detail::FeaturesMatcher::isThreadSafe()	classcv_1_1detail_1_1FeaturesMatcher.html				1										
stitching_FeaturesMatcher_collectGarbage	CVAPI(ExceptionStatus) stitching_FeaturesMatcher_collectGarbage(cv::detail::FeaturesMatcher* obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::detail::FeaturesMatcher::collectGarbage()	classcv_1_1detail_1_1FeaturesMatcher.html	Frees unused memory allocated before if there is any.		Reimplemented in cv::detail::BestOf2NearestMatcher.	1										
stitching_BestOf2NearestMatcher_new	CVAPI(ExceptionStatus) stitching_BestOf2NearestMatcher_new(int try_use_gpu, float match_conf, int num_matches_thresh1,int num_matches_thresh2, cv::detail::BestOf2NearestMatcher **returnValue) { BEGIN_WRAP *returnValue = new cv::detail::BestOf2NearestMatcher(try_use_gpu != 0, match_conf, num_matches_thresh1, num_matches_thresh2); END_WRAP }	cv::detail::BestOf2NearestMatcher	classcv_1_1detail_1_1BestOf2NearestMatcher.html	Features matcher which finds two best matches for each feature and leaves the best one only if the ratio between descriptor distances is greater than the threshold match_conf. See alsodetail::FeaturesMatcher							•				
stitching_BestOf2NearestMatcher_delete	CVAPI(ExceptionStatus) stitching_BestOf2NearestMatcher_delete(cv::detail::BestOf2NearestMatcher* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::detail::BestOf2NearestMatcher	classcv_1_1detail_1_1BestOf2NearestMatcher.html	Features matcher which finds two best matches for each feature and leaves the best one only if the ratio between descriptor distances is greater than the threshold match_conf. See alsodetail::FeaturesMatcher								•			
stitching_BestOf2NearestMatcher_collectGarbage	CVAPI(ExceptionStatus) stitching_BestOf2NearestMatcher_collectGarbage(cv::detail::BestOf2NearestMatcher* obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::detail::BestOf2NearestMatcher::collectGarbage()	classcv_1_1detail_1_1BestOf2NearestMatcher.html	Frees unused memory allocated before if there is any.		Reimplemented from cv::detail::FeaturesMatcher.	1										
stitching_AffineBestOf2NearestMatcher_new	CVAPI(ExceptionStatus) stitching_AffineBestOf2NearestMatcher_new(int full_affine, int try_use_gpu, float match_conf, int num_matches_thresh1, cv::detail::AffineBestOf2NearestMatcher** returnValue) { BEGIN_WRAP *returnValue = new cv::detail::AffineBestOf2NearestMatcher(full_affine != 0, try_use_gpu != 0, match_conf, num_matches_thresh1); END_WRAP }	cv::detail::AffineBestOf2NearestMatcher	classcv_1_1detail_1_1AffineBestOf2NearestMatcher.html	Features matcher similar to cv::detail::BestOf2NearestMatcher which finds two best matches for each feature and leaves the best one only if the ratio between descriptor distances is greater than the threshold match_conf. Unlike cv::detail::BestOf2NearestMatcher this matcher uses affine transformation (affine transformation estimate will be placed in matches_info). See alsocv::detail::FeaturesMatcher cv::detail::BestOf2NearestMatcher							•				
stitching_AffineBestOf2NearestMatcher_delete	CVAPI(ExceptionStatus) stitching_AffineBestOf2NearestMatcher_delete(cv::detail::AffineBestOf2NearestMatcher* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::detail::AffineBestOf2NearestMatcher	classcv_1_1detail_1_1AffineBestOf2NearestMatcher.html	Features matcher similar to cv::detail::BestOf2NearestMatcher which finds two best matches for each feature and leaves the best one only if the ratio between descriptor distances is greater than the threshold match_conf. Unlike cv::detail::BestOf2NearestMatcher this matcher uses affine transformation (affine transformation estimate will be placed in matches_info). See alsocv::detail::FeaturesMatcher cv::detail::BestOf2NearestMatcher								•			
superres_DenseOpticalFlowExt_calc	CVAPI(ExceptionStatus) superres_DenseOpticalFlowExt_calc(cv::superres::DenseOpticalFlowExt *obj, cv::_InputArray *frame0, cv::_InputArray *frame1, cv::_OutputArray *flow1, cv::_OutputArray *flow2) { BEGIN_WRAP obj->calc(*frame0, *frame1, *flow1, entity(flow2)); END_WRAP }	cv::superres::DenseOpticalFlowExt::calc()	classcv_1_1superres_1_1DenseOpticalFlowExt.html				4										
superres_DenseOpticalFlowExt_collectGarbage	CVAPI(ExceptionStatus) superres_DenseOpticalFlowExt_collectGarbage(cv::superres::DenseOpticalFlowExt *obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::superres::DenseOpticalFlowExt::collectGarbage()	classcv_1_1superres_1_1DenseOpticalFlowExt.html				1										
superres_createOptFlow_Farneback	CVAPI(ExceptionStatus) superres_createOptFlow_Farneback(cv::Ptr<cv::superres::FarnebackOpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_Farneback()); END_WRAP }	cv::superres::createOptFlow_Farneback()	group__superres.html				1										
superres_createOptFlow_Farneback_CUDA	CVAPI(ExceptionStatus) superres_createOptFlow_Farneback_CUDA(cv::Ptr<cv::superres::FarnebackOpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_Farneback_CUDA()); END_WRAP }	cv::superres::createOptFlow_Farneback_CUDA()	group__superres.html				1										
superres_Ptr_FarnebackOpticalFlow_get	CVAPI(ExceptionStatus) superres_Ptr_FarnebackOpticalFlow_get(cv::Ptr<cv::superres::FarnebackOpticalFlow> *ptr, cv::superres::FarnebackOpticalFlow **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::FarnebackOpticalFlow												•		
superres_Ptr_FarnebackOpticalFlow_delete	CVAPI(ExceptionStatus) superres_Ptr_FarnebackOpticalFlow_delete(cv::Ptr<cv::superres::FarnebackOpticalFlow> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::FarnebackOpticalFlow											•			
superres_FarnebackOpticalFlow_getPyrScale	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getPyrScale(cv::superres::FarnebackOpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getPyrScale(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getPyrScale()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetPyrScale			1										
superres_FarnebackOpticalFlow_setPyrScale	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setPyrScale(cv::superres::FarnebackOpticalFlow *obj, double val) { BEGIN_WRAP obj->setPyrScale(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setPyrScale()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetPyrScale			1										
superres_FarnebackOpticalFlow_getLevelsNum	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getLevelsNumber(cv::superres::FarnebackOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getLevelsNumber(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getLevelsNumber()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetLevelsNumber			1										

superres_FarnebackOpticalFlow_setLevelsNumber	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setLevelsNumber(cv::superres::FarnebackOpticalFlow *obj, int val) { BEGIN_WRAP obj->setLevelsNumber(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setLevelsNumber()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetLevelsNumber			1									
superres_FarnebackOpticalFlow_getWindowSize	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getWindowSize(cv::superres::FarnebackOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getWindowSize(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getWindowSize()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetWindowSize			1									
superres_FarnebackOpticalFlow_setWindowSize	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setWindowSize(cv::superres::FarnebackOpticalFlow *obj, int val) { BEGIN_WRAP obj->setWindowSize(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setWindowSize()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetWindowSize			1									
superres_FarnebackOpticalFlow_getIterations	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getIterations(cv::superres::FarnebackOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getIterations()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetIterations			1									
superres_FarnebackOpticalFlow_setIterations	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setIterations(cv::superres::FarnebackOpticalFlow *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setIterations()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetIterations			1									
superres_FarnebackOpticalFlow_getPolyN	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getPolyN(cv::superres::FarnebackOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getPolyN(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getPolyN()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetPolyN			1									
superres_FarnebackOpticalFlow_setPolyN	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setPolyN(cv::superres::FarnebackOpticalFlow *obj, int val) { BEGIN_WRAP obj->setPolyN(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setPolyN()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetPolyN			1									
superres_FarnebackOpticalFlow_getPolySigma	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getPolySigma(cv::superres::FarnebackOpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getPolySigma(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getPolySigma()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetPolySigma			1									
superres_FarnebackOpticalFlow_setPolySigma	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setPolySigma(cv::superres::FarnebackOpticalFlow *obj, double val) { BEGIN_WRAP obj->setPolySigma(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setPolySigma()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetPolySigma			1									
superres_FarnebackOpticalFlow_getFlags	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_getFlags(cv::superres::FarnebackOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getFlags(); END_WRAP }	cv::superres::FarnebackOpticalFlow::getFlags()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsosetFlags			1									
superres_FarnebackOpticalFlow_setFlags	CVAPI(ExceptionStatus) superres_FarnebackOpticalFlow_setFlags(cv::superres::FarnebackOpticalFlow *obj, int val) { BEGIN_WRAP obj->setFlags(val); END_WRAP }	cv::superres::FarnebackOpticalFlow::setFlags()	classcv_1_1superres_1_1FarnebackOpticalFlow.html	See alsogetFlags			1									
superres_createOptFlow_DualTVL1	CVAPI(ExceptionStatus) superres_createOptFlow_DualTVL1(cv::Ptr<cv::superres::DualTVL1OpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_DualTVL1()); END_WRAP }	cv::superres::createOptFlow_DualTVL1()	group__superres.html				1									
superres_createOptFlow_DualTVL1_CUDA	CVAPI(ExceptionStatus) superres_createOptFlow_DualTVL1_CUDA(cv::Ptr<cv::superres::DualTVL1OpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_DualTVL1_CUDA()); END_WRAP }	cv::superres::createOptFlow_DualTVL1_CUDA()	group__superres.html				1									
superres_Ptr_DualTVL1OpticalFlow_get	CVAPI(ExceptionStatus) superres_Ptr_DualTVL1OpticalFlow_get(cv::Ptr<cv::superres::DualTVL1OpticalFlow> *ptr, cv::superres::DualTVL1OpticalFlow **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::DualTVL1OpticalFlow											•			
superres_Ptr_DualTVL1OpticalFlow_delete	CVAPI(ExceptionStatus) superres_Ptr_DualTVL1OpticalFlow_delete(cv::Ptr<cv::superres::DualTVL1OpticalFlow> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::DualTVL1OpticalFlow												•			
superres_DualTVL1OpticalFlow_getTau	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getTau(cv::superres::DualTVL1OpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getTau(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getTau()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetTau			1									
superres_DualTVL1OpticalFlow_setTau	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setTau(cv::superres::DualTVL1OpticalFlow *obj, double val) { BEGIN_WRAP obj->setTau(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setTau()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetTau			1									
superres_DualTVL1OpticalFlow_getLambda	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getLambda(cv::superres::DualTVL1OpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getLambda(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getLambda()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetLambda			1									
superres_DualTVL1OpticalFlow_setLambda	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setLambda(cv::superres::DualTVL1OpticalFlow *obj, double val) { BEGIN_WRAP obj->setLambda(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setLambda()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetLambda			1									
superres_DualTVL1OpticalFlow_getTheta	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getTheta(cv::superres::DualTVL1OpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getTheta(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getTheta()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetTheta			1									
superres_DualTVL1OpticalFlow_setTheta	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setTheta(cv::superres::DualTVL1OpticalFlow *obj, double val) { BEGIN_WRAP obj->setTheta(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setTheta()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetTheta			1									
superres_DualTVL1OpticalFlow_getScalesNumber	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getScalesNumber(cv::superres::DualTVL1OpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getScalesNumber(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getScalesNumber()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetScalesNumber			1									

superres_DualTVL1OpticalFlow_setScalesNumb	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setScalesNumber(cv::superres::DualTVL1OpticalFlow *obj, int val) { BEGIN_WRAP obj->setScalesNumber(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setScalesNumber()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetScalesNumber			1										
superres_DualTVL1OpticalFlow_getWarpingsNu	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getWarpingsNumber(cv::superres::DualTVL1OpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getWarpingsNumber(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getWarpingsNumber()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetWarpingsNumber			1										
superres_DualTVL1OpticalFlow_setWarpingsNu	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setWarpingsNumber(cv::superres::DualTVL1OpticalFlow *obj, int val) { BEGIN_WRAP obj->setWarpingsNumber(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setWarpingsNumber()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetWarpingsNumber			1										
superres_DualTVL1OpticalFlow_getEpsilon	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getEpsilon(cv::superres::DualTVL1OpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getEpsilon(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getEpsilon()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetEpsilon			1										
superres_DualTVL1OpticalFlow_setEpsilon	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setEpsilon(cv::superres::DualTVL1OpticalFlow *obj, double val) { BEGIN_WRAP obj->setEpsilon(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setEpsilon()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetEpsilon			1										
superres_DualTVL1OpticalFlow_getIterations	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getIterations(cv::superres::DualTVL1OpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::superres::DualTVL1OpticalFlow::getIterations()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetIterations			1										
superres_DualTVL1OpticalFlow_setIterations	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setIterations(cv::superres::DualTVL1OpticalFlow *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setIterations()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetIterations			1										
superres_DualTVL1OpticalFlow_getUseInitialFl	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_getUseInitialFlow(cv::superres::DualTVL1OpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getUseInitialFlow() ? 1 : 0; END_WRAP }	cv::superres::DualTVL1OpticalFlow::getUseInitialFlow()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsosetUseInitialFlow			1										
superres_DualTVL1OpticalFlow_setUseInitialFl	CVAPI(ExceptionStatus) superres_DualTVL1OpticalFlow_setUseInitialFlow(cv::superres::DualTVL1OpticalFlow *obj, int val) { BEGIN_WRAP obj->setUseInitialFlow(val != 0); END_WRAP }	cv::superres::DualTVL1OpticalFlow::setUseInitialFlow()	classcv_1_1superres_1_1DualTVL1OpticalFlow.html	See alsogetUseInitialFlow			1										
superres_createOptFlow_Brox_CUDA	CVAPI(ExceptionStatus) superres_createOptFlow_Brox_CUDA(cv::Ptr<cv::superres::BroxOpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_Brox_CUDA()); END_WRAP }	cv::superres::createOptFlow_Brox_CUDA()	group__superres.html				1										
superres_Ptr_BroxOpticalFlow_get	CVAPI(ExceptionStatus) superres_Ptr_BroxOpticalFlow_get(cv::Ptr<cv::superres::BroxOpticalFlow> *ptr, cv::superres::BroxOpticalFlow **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::BroxOpticalFlow											•				
superres_Ptr_BroxOpticalFlow_delete	CVAPI(ExceptionStatus) superres_Ptr_BroxOpticalFlow_delete(cv::Ptr<cv::superres::BroxOpticalFlow> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::BroxOpticalFlow												•				
superres_BroxOpticalFlow_getAlpha	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getAlpha(cv::superres::BroxOpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getAlpha(); END_WRAP }	cv::superres::BroxOpticalFlow::getAlpha()	classcv_1_1superres_1_1BroxOpticalFlow.html	Flow smoothness.		See alsosetAlpha	1										
superres_BroxOpticalFlow_setAlpha	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setAlpha(cv::superres::BroxOpticalFlow *obj, double val) { BEGIN_WRAP obj->setAlpha(val); END_WRAP }	cv::superres::BroxOpticalFlow::setAlpha()	classcv_1_1superres_1_1BroxOpticalFlow.html	Flow smoothness.		See alsogetAlpha	1										
superres_BroxOpticalFlow_getGamma	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getGamma(cv::superres::BroxOpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getGamma(); END_WRAP }	cv::superres::BroxOpticalFlowgetGamma()																
superres_BroxOpticalFlow_setGamma	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setGamma(cv::superres::BroxOpticalFlow *obj, double val) { BEGIN_WRAP obj->setGamma(val); END_WRAP }	cv::superres::BroxOpticalFlow::setGamma()	classcv_1_1superres_1_1BroxOpticalFlow.html	Gradient constancy importance.		See alsogetGamma	1										
superres_BroxOpticalFlow_getScaleFactor	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getScaleFactor(cv::superres::BroxOpticalFlow *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getScaleFactor(); END_WRAP }	cv::superres::BroxOpticalFlow::getScaleFactor()	classcv_1_1superres_1_1BroxOpticalFlow.html	Pyramid scale factor.		See alsosetScaleFactor	1										
superres_BroxOpticalFlow_setScaleFactor	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setScaleFactor(cv::superres::BroxOpticalFlow *obj, double val) { BEGIN_WRAP obj->setScaleFactor(val); END_WRAP }	cv::superres::BroxOpticalFlow::setScaleFactor()	classcv_1_1superres_1_1BroxOpticalFlow.html	Pyramid scale factor.		See alsogetScaleFactor	1										
superres_BroxOpticalFlow_getInnerIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getInnerIterations(cv::superres::BroxOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getInnerIterations(); END_WRAP }	cv::superres::BroxOpticalFlow::getInnerIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of lagged non-linearity iterations (inner loop)		See alsosetInnerIterations	1										
superres_BroxOpticalFlow_setInnerIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setInnerIterations(cv::superres::BroxOpticalFlow *obj, int val) { BEGIN_WRAP obj->setInnerIterations(val); END_WRAP }	cv::superres::BroxOpticalFlow::setInnerIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of lagged non-linearity iterations (inner loop)		See alsogetInnerIterations	1										
superres_BroxOpticalFlow_getOuterIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getOuterIterations(cv::superres::BroxOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getOuterIterations(); END_WRAP }	cv::superres::BroxOpticalFlow::getOuterIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of warping iterations (number of pyramid levels)		See alsosetOuterIterations	1										
superres_BroxOpticalFlow_setOuterIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setOuterIterations(cv::superres::BroxOpticalFlow *obj, int val) { BEGIN_WRAP obj->setOuterIterations(val); END_WRAP }	cv::superres::BroxOpticalFlow::setOuterIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of warping iterations (number of pyramid levels)		See alsogetOuterIterations	1										
superres_BroxOpticalFlow_getSolverIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_getSolverIterations(cv::superres::BroxOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getSolverIterations(); END_WRAP }	cv::superres::BroxOpticalFlow::getSolverIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of linear system solver iterations.		See alsosetSolverIterations	1										

superres_BroxOpticalFlow_setSolverIterations	CVAPI(ExceptionStatus) superres_BroxOpticalFlow_setSolverIterations(cv::superres::BroxOpticalFlow *obj, int val) { BEGIN_WRAP obj->setSolverIterations(val); END_WRAP }	cv::superres::BroxOpticalFlow::setSolverIterations()	classcv_1_1superres_1_1BroxOpticalFlow.html	Number of linear system solver iterations.		See alsogetSolverIterations	1									
superres_createOptFlow_PyrLK_CUDA	CVAPI(ExceptionStatus) superres_createOptFlow_PyrLK_CUDA(cv::Ptr<cv::superres::PyrLKOpticalFlow> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createOptFlow_PyrLK_CUDA()); END_WRAP }	cv::superres::createOptFlow_PyrLK_CUDA()	group__superres.html				1									
superres_Ptr_PyrLKOpticalFlow_get	CVAPI(ExceptionStatus) superres_Ptr_PyrLKOpticalFlow_get(cv::Ptr<cv::superres::PyrLKOpticalFlow> *ptr, cv::superres::PyrLKOpticalFlow **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::PyrLKOpticalFlow												•		
superres_Ptr_PyrLKOpticalFlow_delete	CVAPI(ExceptionStatus) superres_Ptr_PyrLKOpticalFlow_delete(cv::Ptr<cv::superres::PyrLKOpticalFlow> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::PyrLKOpticalFlow											•			
superres_PyrLKOpticalFlow_getWindowSize	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_getWindowSize(cv::superres::PyrLKOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getWindowSize(); END_WRAP }	cv::superres::PyrLKOpticalFlow::getWindowSize()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsosetWindowSize			1									
superres_PyrLKOpticalFlow_setWindowSize	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_setWindowSize(cv::superres::PyrLKOpticalFlow *obj, int val) { BEGIN_WRAP obj->setWindowSize(val); END_WRAP }	cv::superres::PyrLKOpticalFlow::setWindowSize()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsogetWindowSize			1									
superres_PyrLKOpticalFlow_getMaxLevel	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_getMaxLevel(cv::superres::PyrLKOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxLevel(); END_WRAP }	cv::superres::PyrLKOpticalFlow::getMaxLevel()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsosetMaxLevel			1									
superres_PyrLKOpticalFlow_setMaxLevel	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_setMaxLevel(cv::superres::PyrLKOpticalFlow *obj, int val) { BEGIN_WRAP obj->setMaxLevel(val); END_WRAP }	cv::superres::PyrLKOpticalFlow::setMaxLevel()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsogetMaxLevel			1									
superres_PyrLKOpticalFlow_getIterations	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_getIterations(cv::superres::PyrLKOpticalFlow *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::superres::PyrLKOpticalFlow::getIterations()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsosetIterations			1									
superres_PyrLKOpticalFlow_setIterations	CVAPI(ExceptionStatus) superres_PyrLKOpticalFlow_setIterations(cv::superres::PyrLKOpticalFlow *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::superres::PyrLKOpticalFlow::setIterations()	classcv_1_1superres_1_1PyrLKOpticalFlow.html	See alsogetIterations			1									
superres_FrameSource_nextFrame	CVAPI(ExceptionStatus) superres_FrameSource_nextFrame(cv::Ptr<cv::superres::FrameSource> *obj, cv::_OutputArray *frame) { BEGIN_WRAP (*obj)->nextFrame(*frame); END_WRAP }	cv::superres::FrameSource::nextFrame()	classcv_1_1superres_1_1FrameSource.html	Implemented in cv::superres::SuperResolution.			1									
superres_FrameSource_reset	CVAPI(ExceptionStatus) superres_FrameSource_reset(cv::Ptr<cv::superres::FrameSource> *obj) { BEGIN_WRAP (*obj)->reset(); END_WRAP }	cv::superres::FrameSource::reset()	classcv_1_1superres_1_1FrameSource.html	Implemented in cv::superres::SuperResolution.			1									
superres_createFrameSource_Empty	CVAPI(ExceptionStatus) superres_createFrameSource_Empty(cv::Ptr<cv::superres::FrameSource> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createFrameSource_Empty()); END_WRAP }	cv::superres::createFrameSource_Empty()	group__superres.html				1									
superres_createFrameSource_Video	CVAPI(ExceptionStatus) superres_createFrameSource_Video(const char *fileName, cv::Ptr<cv::superres::FrameSource> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createFrameSource_Video(fileName)); END_WRAP }	cv::superres::createFrameSource_Video()	group__superres.html				1									
superres_createFrameSource_Video_CUDA	CVAPI(ExceptionStatus) superres_createFrameSource_Video_CUDA(const char *fileName, cv::Ptr<cv::superres::FrameSource> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createFrameSource_Video_CUDA(fileName)); END_WRAP }	cv::superres::createFrameSource_Video_CUDA()	group__superres.html				1									
superres_createFrameSource_Camera	CVAPI(ExceptionStatus) superres_createFrameSource_Camera(int deviceId, cv::Ptr<cv::superres::FrameSource> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createFrameSource_Camera(deviceId)); END_WRAP }	cv::superres::createFrameSource_Camera()	group__superres.html				1									
superres_Ptr_FrameSource_get	CVAPI(ExceptionStatus) superres_Ptr_FrameSource_get(cv::Ptr<cv::superres::FrameSource> *ptr, cv::superres::FrameSource **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::FrameSource												•		
superres_Ptr_FrameSource_delete	CVAPI(ExceptionStatus) superres_Ptr_FrameSource_delete(cv::Ptr<cv::superres::FrameSource> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::FrameSource											•			

superres_SuperResolution_setInput	CVAPI(ExceptionStatus) superres_SuperResolution_setInput(cv::superres::SuperResolution *obj, cv::Ptr<cv::superres::FrameSource> *frameSource) { BEGIN_WRAP obj->setInput(*frameSource); END_WRAP }	cv::superres::SuperResolution::setInput()	classcv_1_1superres_1_1SuperResolution.html	Set input frame source for Super Resolution algorithm.	frameSource : Input frame source		1										
superres_SuperResolution_nextFrame	CVAPI(ExceptionStatus) superres_SuperResolution_nextFrame(cv::superres::SuperResolution *obj, cv::_OutputArray *frame) { BEGIN_WRAP obj->nextFrame(*frame); END_WRAP }	cv::superres::SuperResolution::nextFrame()	classcv_1_1superres_1_1SuperResolution.html	Process next frame from input and return output result.	frame : Output result	Implements cv::superres::FrameSource.	1										
superres_SuperResolution_reset	CVAPI(ExceptionStatus) superres_SuperResolution_reset(cv::superres::SuperResolution *obj) { BEGIN_WRAP obj->reset(); END_WRAP }	cv::superres::SuperResolution::reset()	classcv_1_1superres_1_1SuperResolution.html	Implements cv::superres::FrameSource.			1										
superres_SuperResolution_collectGarbage	CVAPI(ExceptionStatus) superres_SuperResolution_collectGarbage(cv::superres::SuperResolution *obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::superres::SuperResolution::collectGarbage()	classcv_1_1superres_1_1SuperResolution.html	Clear all inner buffers.			1										
superres_createSuperResolution_BTVL1	CVAPI(ExceptionStatus) superres_createSuperResolution_BTVL1(cv::Ptr<cv::superres::SuperResolution>**returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createSuperResolution_BTVL1()); END_WRAP }	cv::superres::createSuperResolution_BTVL1()	group__superres.html	Create Bilateral TV-L1 Super Resolution.		This class implements Super Resolution algorithm described in the papers [70] and [175] .Here are important members of the class that control the algorithm, which you can set after constructing the class instance:int scale Scale factor. int iterations Iteration count. double tau Asymptotic value of steepest descent method. double lambda Weight parameter to balance data term and smoothness term. double alpha Parameter of spacial distribution in Bilateral-TV. int btvKernelSize Kernel size of Bilateral-TV filter. int blurKernelSize Gaussian blur kernel size. double blurSigma Gaussian blur sigma. int temporalAreaRadius Radius of the temporal search area. Ptr<DenseOpticalFlowExt> opticalFlow	1										
superres_createSuperResolution_BTVL1_CUDA	CVAPI(ExceptionStatus) superres_createSuperResolution_BTVL1_CUDA(cv::Ptr<cv::superres::SuperResolution>**returnValue) { BEGIN_WRAP *returnValue = clone(cv::superres::createSuperResolution_BTVL1_CUDA()); END_WRAP }	cv::superres::createSuperResolution_BTVL1_CUDA()	group__superres.html				1										
superres_Ptr_SuperResolution_get	CVAPI(ExceptionStatus) superres_Ptr_SuperResolution_get(cv::Ptr<cv::superres::SuperResolution> *ptr, cv::superres::SuperResolution **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::superres::SuperResolution	classcv_1_1superres_1_1SuperResolution.html	Base class for Super Resolution algorithms. The class is only used to define the common interface for the whole family of Super Resolution algorithms.										•		
superres_Ptr_SuperResolution_delete	CVAPI(ExceptionStatus) superres_Ptr_SuperResolution_delete(cv::Ptr<cv::superres::SuperResolution> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::superres::SuperResolution	classcv_1_1superres_1_1SuperResolution.html	Base class for Super Resolution algorithms. The class is only used to define the common interface for the whole family of Super Resolution algorithms.								•			
superres_SuperResolution_getScale	CVAPI(ExceptionStatus) superres_SuperResolution_getScale(cv::superres::SuperResolution *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getScale(); END_WRAP }	cv::superres::SuperResolution::getScale()	classcv_1_1superres_1_1SuperResolution.html	Scale factor.		See alsosetScale	1										
superres_SuperResolution_setScale	CVAPI(ExceptionStatus) superres_SuperResolution_setScale(cv::superres::SuperResolution *obj, int val) { BEGIN_WRAP obj->setScale(val); END_WRAP }	cv::superres::SuperResolution::setScale()	classcv_1_1superres_1_1SuperResolution.html	Scale factor.		See alsogetScale	1										
superres_SuperResolution_getIterations	CVAPI(ExceptionStatus) superres_SuperResolution_getIterations(cv::superres::SuperResolution *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getIterations(); END_WRAP }	cv::superres::SuperResolution::getIterations()	classcv_1_1superres_1_1SuperResolution.html	Iterations count.		See alsosetIterations	1										
superres_SuperResolution_setIterations	CVAPI(ExceptionStatus) superres_SuperResolution_setIterations(cv::superres::SuperResolution *obj, int val) { BEGIN_WRAP obj->setIterations(val); END_WRAP }	cv::superres::SuperResolution::setIterations()	classcv_1_1superres_1_1SuperResolution.html	Iterations count.		See alsogetIterations	1										
superres_SuperResolution_getTau	CVAPI(ExceptionStatus) superres_SuperResolution_getTau(cv::superres::SuperResolution *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getTau(); END_WRAP }	cv::superres::SuperResolution::getTau()	classcv_1_1superres_1_1SuperResolution.html	Asymptotic value of steepest descent method.		See alsosetTau	1										
superres_SuperResolution_setTau	CVAPI(ExceptionStatus) superres_SuperResolution_setTau(cv::superres::SuperResolution *obj, double val) { BEGIN_WRAP obj->setTau(val); END_WRAP }	cv::superres::SuperResolution::setTau()	classcv_1_1superres_1_1SuperResolution.html	Asymptotic value of steepest descent method.		See alsogetTau	1										
superres_SuperResolution_getLambda	CVAPI(ExceptionStatus) superres_SuperResolution_getLambda(cv::superres::SuperResolution *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getLambda(); END_WRAP }	cv::superres::SuperResolution::getLambda()	classcv_1_1superres_1_1SuperResolution.html	Weight parameter to balance data term and smoothness term.		See alsosetLambda	1										
superres_SuperResolution_setLambda	CVAPI(ExceptionStatus) superres_SuperResolution_setLambda(cv::superres::SuperResolution *obj, double val) { BEGIN_WRAP obj->setLambda(val); END_WRAP }	cv::superres::SuperResolution::setLambda()	classcv_1_1superres_1_1SuperResolution.html	Weight parameter to balance data term and smoothness term.		See alsogetLambda	1										
superres_SuperResolution_getAlpha	CVAPI(ExceptionStatus) superres_SuperResolution_getAlpha(cv::superres::SuperResolution *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getAlpha(); END_WRAP }	cv::superres::SuperResolution::getAlpha()	classcv_1_1superres_1_1SuperResolution.html	Parameter of spacial distribution in Bilateral-TV.		See alsosetAlpha	1										
superres_SuperResolution_setAlpha	CVAPI(ExceptionStatus) superres_SuperResolution_setAlpha(cv::superres::SuperResolution *obj, double val) { BEGIN_WRAP obj->setAlpha(val); END_WRAP }	cv::superres::SuperResolution::setAlpha()	classcv_1_1superres_1_1SuperResolution.html	Parameter of spacial distribution in Bilateral-TV.		See alsogetAlpha	1										
superres_SuperResolution_getKernelSize	CVAPI(ExceptionStatus) superres_SuperResolution_getKernelSize(cv::superres::SuperResolution *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getKernelSize(); END_WRAP }	cv::superres::SuperResolution::getKernelSize()	classcv_1_1superres_1_1SuperResolution.html	Kernel size of Bilateral-TV filter.		See alsosetKernelSize	1										
superres_SuperResolution_setKernelSize	CVAPI(ExceptionStatus) superres_SuperResolution_setKernelSize(cv::superres::SuperResolution *obj, int val) { BEGIN_WRAP obj->setKernelSize(val); END_WRAP }	cv::superres::SuperResolution::setKernelSize()	classcv_1_1superres_1_1SuperResolution.html	Kernel size of Bilateral-TV filter.		See alsogetKernelSize	1										

superres_SuperResolution_getBlurKernelSize	CVAPI(ExceptionStatus) superres_SuperResolution_getBlurKernelSize(cv::superres::SuperResolution *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getBlurKernelSize(); END_WRAP }	cv::superres::SuperResolution::getBlurKernelSize()	classcv_1_1superres_1_1SuperResolution.html	Gaussian blur kernel size.		See alsosetBlurKernelSize	1									
superres_SuperResolution_setBlurKernelSize	CVAPI(ExceptionStatus) superres_SuperResolution_setBlurKernelSize(cv::superres::SuperResolution *obj, int val) { BEGIN_WRAP obj->setBlurKernelSize(val); END_WRAP }	cv::superres::SuperResolution::setBlurKernelSize()	classcv_1_1superres_1_1SuperResolution.html	Gaussian blur kernel size.		See alsogetBlurKernelSize	1									
superres_SuperResolution_getBlurSigma	CVAPI(ExceptionStatus) superres_SuperResolution_getBlurSigma(cv::superres::SuperResolution *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getBlurSigma(); END_WRAP }	cv::superres::SuperResolution::getBlurSigma()	classcv_1_1superres_1_1SuperResolution.html	Gaussian blur sigma.		See alsosetBlurSigma	1									
superres_SuperResolution_setBlurSigma	CVAPI(ExceptionStatus) superres_SuperResolution_setBlurSigma(cv::superres::SuperResolution *obj, double val) { BEGIN_WRAP obj->setBlurSigma(val); END_WRAP }	cv::superres::SuperResolution::setBlurSigma()	classcv_1_1superres_1_1SuperResolution.html	Gaussian blur sigma.		See alsogetBlurSigma	1									
superres_SuperResolution_getTemporalAreaRadius	CVAPI(ExceptionStatus) superres_SuperResolution_getTemporalAreaRadius(cv::superres::SuperResolution *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getTemporalAreaRadius(); END_WRAP }	cv::superres::SuperResolution::getTemporalAreaRadius()	classcv_1_1superres_1_1SuperResolution.html	Radius of the temporal search area.		See alsosetTemporalAreaRadius	1									
superres_SuperResolution_setTemporalAreaRadius	CVAPI(ExceptionStatus) superres_SuperResolution_setTemporalAreaRadius(cv::superres::SuperResolution *obj, int val) { BEGIN_WRAP obj->setTemporalAreaRadius(val); END_WRAP }	cv::superres::SuperResolution::setTemporalAreaRadius()	classcv_1_1superres_1_1SuperResolution.html	Radius of the temporal search area.		See alsogetTemporalAreaRadius	1									
superres_SuperResolution_getOpticalFlow	CVAPI(ExceptionStatus) superres_SuperResolution_getOpticalFlow(cv::superres::SuperResolution *obj, cv::Ptr<cv::superres::DenseOpticalFlowExt> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::superres::DenseOpticalFlowExt>(obj->getOpticalFlow()); END_WRAP }	cv::superres::SuperResolution::getOpticalFlow()														
superres_SuperResolution_setOpticalFlow	CVAPI(ExceptionStatus) superres_SuperResolution_setOpticalFlow(cv::superres::SuperResolution *obj, cv::Ptr<cv::superres::DenseOpticalFlowExt> *val) { BEGIN_WRAP obj->setOpticalFlow(*val); END_WRAP }	cv::superres::SuperResolution::setOpticalFlow()	classcv_1_1superres_1_1SuperResolution.html	Dense optical flow algorithm.		See alsogetOpticalFlow	1									
text_OCRTesseract_run1	CVAPI(ExceptionStatus) text_OCRTesseract_run1(cv::text::OCRTesseract *obj, cv::Mat *image, std::string *output_text, std::vector<cv::Rect>* component_rects, std::vector<std::string>* component_texts, std::vector<float>* component_confidences, int component_level) { BEGIN_WRAP obj->run(*image, *output_text, component_rects, component_texts, component_confidences, component_level); END_WRAP }	cv::text::OCRTesseract::run()	classcv_1_1text_1_1OCRTesseract.html	Recognize text using the tesseract-ocr API.	image : Input image CV_8UC1 or CV_8UC3 output_text : Output text of the tesseract-ocr. component_rects : If provided the method will output a list of Rects for the individual text elements found (e.g. words or text lines). component_texts : If provided the method will output a list of text strings for the recognition of individual text elements found (e.g. words or text lines). component_confidences : If provided the method will output a list of confidence values for the recognition of individual text elements found (e.g. words or text lines). component_level : OCR_LEVEL_WORD (by default), or OCR_LEVEL_TEXTLINE.	Takes image on input and returns recognized text in the output_text parameter. Optionally provides also the Rects for individual text elements found (e.g. words), and the list of those text elements with their confidence values.Implements cv::text::BaseOCR.	6									
text_OCRTesseract_run2	CVAPI(ExceptionStatus) text_OCRTesseract_run2(cv::text::OCRTesseract *obj, cv::Mat *image, cv::Mat *mask, std::string *output_text, std::vector<cv::Rect>* component_rects, std::vector<std::string>* component_texts, std::vector<float>* component_confidences, int component_level) { BEGIN_WRAP obj->run(*image, *mask, *output_text, component_rects, component_texts, component_confidences, component_level); END_WRAP }	cv::text::OCRTesseract::run()	classcv_1_1text_1_1OCRTesseract.html	Recognize text using the tesseract-ocr API.	image : Input image CV_8UC1 or CV_8UC3 output_text : Output text of the tesseract-ocr. component_rects : If provided the method will output a list of Rects for the individual text elements found (e.g. words or text lines). component_texts : If provided the method will output a list of text strings for the recognition of individual text elements found (e.g. words or text lines). component_confidences : If provided the method will output a list of confidence values for the recognition of individual text elements found (e.g. words or text lines). component_level : OCR_LEVEL_WORD (by default), or OCR_LEVEL_TEXTLINE.	Takes image on input and returns recognized text in the output_text parameter. Optionally provides also the Rects for individual text elements found (e.g. words), and the list of those text elements with their confidence values.Implements cv::text::BaseOCR.	7									
text_OCRTesseract_setWhiteList	CVAPI(ExceptionStatus) text_OCRTesseract_setWhiteList(cv::text::OCRTesseract *obj, const char *char_whitelist) { BEGIN_WRAP obj->setWhiteList(char_whitelist); END_WRAP }	cv::text::OCRTesseract::setWhiteList()	classcv_1_1text_1_1OCRTesseract.html				1									
text_OCRTesseract_create	CVAPI(ExceptionStatus) text_OCRTesseract_create(const char* datapath, const char* language, const char* char_whitelist, int oem, int psmode, cv::Ptr<cv::text::OCRTesseract> **returnValue) { BEGIN_WRAP const auto result = cv::text::OCRTesseract::create(datapath, language, char_whitelist, oem, psmode); *returnValue = clone(result); END_WRAP }	cv::text::OCRTesseract::create()	classcv_1_1text_1_1OCRTesseract.html	Creates an instance of the OCRTesseract class. Initializes Tesseract.	datapath : the name of the parent directory of tessdata ended with "/", or NULL to use the system's default directory. language : an ISO 639-3 code or NULL will default to "eng". char_whitelist : specifies the list of characters used for recognition. NULL defaults to "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ". oem : tesseract-ocr offers different OCR Engine Modes (OEM), by default tesseract::OEM_DEFAULT is used. See the tesseract-ocr API documentation for other possible values. psmode : tesseract-ocr offers different Page Segmentation Modes (PSM) tesseract::PSM_AUTO (fully automatic layout analysis) is used. See the tesseract-ocr API documentation for other possible values.		5									
text_Ptr_OCRTesseract_delete	CVAPI(ExceptionStatus) text_Ptr_OCRTesseract_delete(cv::Ptr<cv::text::OCRTesseract> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::text::OCRTesseract	classcv_1_1text_1_1OCRTesseract.html	OCRTesseract class provides an interface with the tesseract-ocr API (v3.02.02) in C++. Notice that it is compiled only when tesseract-ocr is correctly installed. Note (C++) An example of OCRTesseract recognition combined with scene text detection can be found at the end_to_end_recognition demo: https://github.com/opencv/opencv_contrib/blob/master/modules/text/samples/end_to_end_recognition.cpp (C++) Another example of OCRTesseract recognition combined with scene text detection can be found at the webcam_demo: https://github.com/opencv/opencv_contrib/blob/master/modules/text/samples/webcam_demo.cpp							•		
text_OCRTesseract_get	CVAPI(ExceptionStatus) text_OCRTesseract_get(cv::Ptr<cv::text::OCRTesseract> *obj, cv::text::OCRTesseract **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::text::OCRTesseract	classcv_1_1text_1_1OCRTesseract.html	OCRTesseract class provides an interface with the tesseract-ocr API (v3.02.02) in C++. Notice that it is compiled only when tesseract-ocr is correctly installed. Note (C++) An example of OCRTesseract recognition combined with scene text detection can be found at the end_to_end_recognition demo: https://github.com/opencv/opencv_contrib/blob/master/modules/text/samples/end_to_end_recognition.cpp (C++) Another example of OCRTesseract recognition combined with scene text detection can be found at the webcam_demo: https://github.com/opencv/opencv_contrib/blob/master/modules/text/samples/webcam_demo.cpp								•	

text_detectTextSWT	CVAPI(ExceptionStatus) text_detectTextSWT(cv::_InputArray *input, std::vector<cv::Rect> *result, int dark_on_light, cv::_OutputArray *draw, cv::_OutputArray *chainBBs) { BEGIN_WRAP cv::text::detectTextSWT(*inp ut, *result, dark_on_light != 0, entity(draw), entity(chainBBs)); END_WRAP }	cv::text::detectTextSWT()	namespacecv_1_1text.html	Applies the Stroke Width Transform operator followed by filtering of connected components of similar Stroke Widths to return letter candidates. It also chain them by proximity and size, saving the result in chainBBs.	input : the input image with 3 channels. result : a vector of resulting bounding boxes where probability of finding text is high dark_on_light : a boolean value signifying whether the text is darker or lighter than the background, it is observed to reverse the gradient obtained from Scharr operator, and significantly affect the result. draw : an optional Mat of type CV_8UC3 which visualises the detected letters using bounding boxes. chainBBs : an optional parameter which chains the letter candidates according to heuristics in the paper and returns all possible regions where text is likely to occur.		5								
text_TextDetector_detect	CVAPI(ExceptionStatus) text_TextDetector_detect(cv: :text::TextDetector *obj, cv::_InputArray *inputImage, std::vector<cv::Rect> *Bbox, std::vector<float> *confidence) { BEGIN_WRAP obj->detect(*inputImage, *Bbox, *confidence); END_WRAP }	cv::text::TextDetector::detect()	classcv_1_1text_1_1TextDetector.h tml	Method that provides a quick and simple interface to detect text inside an image.	inputImage : an image to process Bbox : a vector of Rect that will store the detected word bounding box confidence : a vector of float that will be updated with the confidence the classifier has for the selected bounding box	Implemented in cv::text::TextDetectorCNN.	3								
text_TextDetectorCNN_detect	CVAPI(ExceptionStatus) text_TextDetectorCNN_detect (cv::text::TextDetectorCNN *obj, cv::_InputArray *inputImage, std::vector<cv::Rect> *Bbox, std::vector<float> *confidence) { BEGIN_WRAP obj->detect(*inputImage, *Bbox, *confidence); END_WRAP }	cv::text::TextDetectorCNN::detect()	classcv_1_1text_1_1TextDetectorC NN.html	This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.	inputImage : an image expected to be a CV_U8C3 of any size Bbox : a vector of Rect that will store the detected word bounding box confidence : a vector of float that will be updated with the confidence the classifier has for the selected bounding box	Implements cv::text::TextDetector.	3								
text_TextDetectorCNN_create1	CVAPI(ExceptionStatus) text_TextDetectorCNN_creata t(const char *modelArchFilename, const char *modelWeightsFilename, MyCvSize *detectionSizes, int detectionSizesLength, cv::Ptr<cv::text::TextDetecto rCNN> **returnValue) { BEGIN_WRAP std::vector<cv::Size> detectionSizesVec; if (detectionSizes != null)ptr) { detectionSizesVec.resize(dete ctionSizesLength); for (int i = 0; i < detectionSizesLength; i++) detectionSizesVec[i] = cpp(detectionSizes[i]); } 	cv::text::TextDetectorCNN::create()	classcv_1_1text_1_1TextDetectorC NN.html	Creates an instance of the TextDetectorCNN class using the provided parameters.	modelArchFilename : the relative or absolute path to the prototxt file describing the classifiers architecture. modelWeightsFilename : the relative or absolute path to the file containing the pretrained weights of the model in caffe-binary form. detectionSizes : a list of sizes for multiscale detection. The values[(300,300),(700,500),(700,300),(700,700),(160 0,1600)] are recommended in [146] to achieve the best quality.		3								
text_TextDetectorCNN_create2	CVAPI(ExceptionStatus) text_TextDetectorCNN_creata 2(const char *modelArchFilename, const char *modelWeightsFilename, cv::Ptr<cv::text::TextDetecto rCNN> **returnValue) { BEGIN_WRAP const auto ptr = cv::text::TextDetectorCNN::c reate(modelArchFilename, modelWeightsFilename); *returnValue = clone(ptr); END_WRAP }	cv::text::TextDetectorCNN::create()	classcv_1_1text_1_1TextDetectorC NN.html	Creates an instance of the TextDetectorCNN class using the provided parameters.	modelArchFilename : the relative or absolute path to the prototxt file describing the classifiers architecture. modelWeightsFilename : the relative or absolute path to the file containing the pretrained weights of the model in caffe-binary form. detectionSizes : a list of sizes for multiscale detection. The values[(300,300),(700,500),(700,300),(700,700),(160 0,1600)] are recommended in [146] to achieve the best quality.		2								
text_Ptr_TextDetectorCNN_delete	CVAPI(ExceptionStatus) text_Ptr_TextDetectorCNN_de lete(cv::Ptr<cv::text::TextDe tectorCNN> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::text::TextDetectorCNN	classcv_1_1text_1_1TextDetectorC NN.html	TextDetectorCNN class provides the functionality of text bounding box detection. This class is representing to find bounding boxes of text words given an input image. This class uses OpenCV dnn module to load pre-trained model described in [146]. The original repository with the modified SSD Caffe version: https://github.com/MhLiao/TextBoxes. Model can be downloaded from DropBox. Modified .prototxt file with the model description can be found in opencv_contrib/modules/text/samples/textbox.pro totxt.							•		
text_Ptr_TextDetectorCNN_get	CVAPI(ExceptionStatus) text_Ptr_TextDetectorCNN_ge t(cv::Ptr<cv::text::TextDetec torCNN>* obj, cv::text::TextDetectorCNN **returnValue) { BEGIN_WRAP *returnValue = obj->get(); END_WRAP }	cv::text::TextDetectorCNN	classcv_1_1text_1_1TextDetectorC NN.html	TextDetectorCNN class provides the functionality of text bounding box detection. This class is representing to find bounding boxes of text words given an input image. This class uses OpenCV dnn module to load pre-trained model described in [146]. The original repository with the modified SSD Caffe version: https://github.com/MhLiao/TextBoxes. Model can be downloaded from DropBox. Modified .prototxt file with the model description can be found in opencv_contrib/modules/text/samples/textbox.pro totxt.								•	
tracking_TrackerKCF_create1	CVAPI(ExceptionStatus) tracking_TrackerKCF_create1(cv::Ptr<cv::TrackerKCF> **returnValue) { BEGIN_WRAP const auto p = cv::TrackerKCF::create(); *returnValue = clone(p); END_WRAP }	cv::TrackerKCF::create()	classcv_1_1TrackerKCF.html	Create KCF tracker instance.	parameters : KCF parameters TrackerKCF::Params		1								
tracking_TrackerKCF_create2	CVAPI(ExceptionStatus) tracking_TrackerKCF_create2(cv::TrackerKCF::Params *parameters, cv::Ptr<cv::TrackerKCF> **returnValue) { BEGIN_WRAP const auto p = cv::TrackerKCF::create(*para meters); *returnValue = clone(p); END_WRAP }	cv::TrackerKCF::create()	classcv_1_1TrackerKCF.html	Create KCF tracker instance.	parameters : KCF parameters TrackerKCF::Params		1								
tracking_Ptr_TrackerKCF_delete	CVAPI(ExceptionStatus) tracking_Ptr_TrackerKCF_dele te(cv::Ptr<cv::TrackerKCF> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TrackerKCF	classcv_1_1TrackerKCF.html	the KCF (Kernelized Correlation Filter) tracker KCF is a novel tracking framework that utilizes properties of circulant matrix to enhance the processing speed. This tracking method is an implementation of [108] which is extended to KCF with color-names features ([51]). The original paper of KCF is available at http://www.robots.ox.ac.uk/~joao/publications/ha nriques_tpami2015.pdf as well as the matlab implementation. For more information about KCF with color-names features, please refer to http://www.cvl.isy.liu.se/research/objrec/visualtra cking/colvistrack/index.html.							•		
tracking_Ptr_TrackerKCF_get	CVAPI(ExceptionStatus) tracking_Ptr_TrackerKCF_get(cv::Ptr<cv::TrackerKCF> *ptr, cv::TrackerKCF **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TrackerKCF	classcv_1_1TrackerKCF.html	the KCF (Kernelized Correlation Filter) tracker KCF is a novel tracking framework that utilizes properties of circulant matrix to enhance the processing speed. This tracking method is an implementation of [108] which is extended to KCF with color-names features ([51]). The original paper of KCF is available at http://www.robots.ox.ac.uk/~joao/publications/ha nriques_tpami2015.pdf as well as the matlab implementation. For more information about KCF with color-names features, please refer to http://www.cvl.isy.liu.se/research/objrec/visualtra cking/colvistrack/index.html.								•	
tracking_TrackerCSRT_create1	CVAPI(ExceptionStatus) tracking_TrackerCSRT_create 1(cv::Ptr<cv::TrackerCSRT> **returnValue) { BEGIN_WRAP const auto p = cv::TrackerCSRT::create(); *returnValue = clone(p); END_WRAP }	cv::TrackerCSRT::create()	classcv_1_1TrackerCSRT.html	Create CSRT tracker instance.	parameters : CSRT parameters TrackerCSRT::Params		1								
tracking_TrackerCSRT_create2	CVAPI(ExceptionStatus) tracking_TrackerCSRT_create 2(tracker_TrackerCSRT_Para ms* parameters, cv::Ptr<cv::TrackerCSRT> **returnValue) { BEGIN_WRAP const auto p = tracking_TrackerCSRT_Param _ToCpp(parameters); const auto obj = cv::TrackerCSRT::create(p); *returnValue = clone(obj); END_WRAP }	cv::TrackerCSRT::create()	classcv_1_1TrackerCSRT.html	Create CSRT tracker instance.	parameters : CSRT parameters TrackerCSRT::Params		1								
tracking_Ptr_TrackerCSRT_delete	CVAPI(ExceptionStatus) tracking_Ptr_TrackerCSRT_del ete(cv::Ptr<cv::TrackerCSRT > * ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TrackerCSRT	classcv_1_1TrackerCSRT.html	the CSRT tracker The implementation is based on [156] Discriminative Correlation Filter with Channel and Spatial Reliability								•		

tracking_Ptr_TrackerCSRT_get	CVAPI(ExceptionStatus) tracking_Ptr_TrackerCSRT_get(cv::Ptr<cv::TrackerCSRT>* ptr, cv::TrackerCSRT **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TrackerCSRT	classcv_1_1TrackerCSRT.html	the CSRT tracker The implementation is based on [156] Discriminative Correlation Filter with Channel and Spatial Reliability										•	
tracking_TrackerCSRT_setInitialMask	CVAPI(ExceptionStatus) tracking_TrackerCSRT_setInitialMask(cv::TrackerCSRT *tracker, cv::_InputArray *mask) { BEGIN_WRAP tracker->setInitialMask(*mask); END_WRAP }	cv::TrackerCSRT::setInitialMask()	classcv_1_1TrackerCSRT.html				1								
video_BackgroundSubtractor_getBackgroundImage	CVAPI(ExceptionStatus) video_BackgroundSubtractor_getBackgroundImage(cv::BackgroundSubtractor *obj, cv::_OutputArray *backgroundImage) { BEGIN_WRAP obj->getBackgroundImage(*backgroundImage); END_WRAP }	cv::BackgroundSubtractor::getBackgroundImage()	classcv_1_1BackgroundSubtractor.html	Computes a background image.	backgroundImage : The output background image.	NoteSometimes the background image can be very blurry, as it contain the average background statistics.Implemented in cv::bgsegm::BackgroundSubtractorLSBP, cv::bgsegm::BackgroundSubtractorGSOC, and cv::bgsegm::BackgroundSubtractorCNT.	1								
video_BackgroundSubtractor_apply	CVAPI(ExceptionStatus) video_BackgroundSubtractor_apply(cv::BackgroundSubtractor *obj, cv::_InputArray *image, cv::_OutputArray *fgmask, double learningRate) { BEGIN_WRAP obj->apply(*image, *fgmask, learningRate); END_WRAP }	cv::BackgroundSubtractor::apply()	classcv_1_1BackgroundSubtractor.html	Computes a foreground mask.	image : Next video frame. fgmask : The output foreground mask as an 8-bit binary image. learningRate : The value between 0 and 1 that indicates how fast the background model is learnt. Negative parameter value makes the algorithm to use some automatically chosen learning rate. 0 means that the background model is not updated at all, 1 means that the background model is completely reinitialized from the last frame.	Implemented in cv::bgsegm::BackgroundSubtractorLSBP, cv::bgsegm::BackgroundSubtractorGSOC, cv::BackgroundSubtractorMOG2, and cv::bgsegm::BackgroundSubtractorCNT.	3								
video_Ptr_BackgroundSubtractor_delete	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractor_delete(cv::Ptr<cv::BackgroundSubtractor> *ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::BackgroundSubtractor	classcv_1_1BackgroundSubtractor.html	Base class for background/foreground segmentation. : The class is only used to define the common interface for the whole family of background/foreground segmentation algorithms.								•		
video_Ptr_BackgroundSubtractor_get	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractor_get(cv::Ptr<cv::BackgroundSubtractor> *ptr, cv::BackgroundSubtractor **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::BackgroundSubtractor	classcv_1_1BackgroundSubtractor.html	Base class for background/foreground segmentation. : The class is only used to define the common interface for the whole family of background/foreground segmentation algorithms.									•	
video_createBgSubtractorMOG2	CVAPI(ExceptionStatus) video_createBackgroundSubtractorMOG2(int history, double varThreshold, int detectShadows, cv::Ptr<cv::BackgroundSubtractorMOG2> **returnValue) { BEGIN_WRAP const auto ptr = cv::createBackgroundSubtractorMOG2(history, varThreshold, detectShadows != 0); *returnValue = clone(ptr); END_WRAP }	cv::createBackgroundSubtractorMOG2()	group__video__motion.html	Creates MOG2 Background Subtractor.	history : Length of the history. varThreshold : Threshold on the squared Mahalanobis distance between the pixel and the model to decide whether a pixel is well described by the background model. This parameter does not affect the background update. detectShadows : If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false.	Examples: samples/cpp/segment_objects.cpp.	3								
video_Ptr_BgSubtractorMOG2_delete	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractorMOG2_delete(cv::Ptr<cv::BackgroundSubtractorMOG2> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::BackgroundSubtractorMOG2	classcv_1_1BackgroundSubtractorMOG2.html	Gaussian Mixture-based Background/Foreground Segmentation Algorithm. The class implements the Gaussian mixture model background subtraction described in [299] and [298] .								•		
video_Ptr_BgSubtractorMOG2_get	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractorMOG2_get(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, cv::BackgroundSubtractorMOG2 **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::BackgroundSubtractorMOG2	classcv_1_1BackgroundSubtractorMOG2.html	Gaussian Mixture-based Background/Foreground Segmentation Algorithm. The class implements the Gaussian mixture model background subtraction described in [299] and [298] .									•	
video_BgSubtractorMOG2_getHistory	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getHistory(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getHistory(); END_WRAP }	cv::BackgroundSubtractorMOG2::getHistory()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the number of last frames that affect the background model.			1								
video_BgSubtractorMOG2_setHistory	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setHistory(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int value) { BEGIN_WRAP (*ptr)->setHistory(value); END_WRAP }	cv::BackgroundSubtractorMOG2()setHistory()													
video_BgSubtractorMOG2_getNMixtures	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getNMixtures(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getNMixtures(); END_WRAP }	cv::BackgroundSubtractorMOG2::getNMixtures()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the number of gaussian components in the background model.			1								
video_BgSubtractorMOG2_setNMixtures	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setNMixtures(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int value) { BEGIN_WRAP (*ptr)->setNMixtures(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setNMixtures()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the number of gaussian components in the background model.		The model needs to be reinitalized to reserve memory.	1								
video_BgSubtractorMOG2_getBackgroundRatio	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getBackgroundRatio(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getBackgroundRatio(); END_WRAP }	cv::BackgroundSubtractorMOG2::getBackgroundRatio()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the "background ratio" parameter of the algorithm.		If a foreground pixel keeps semi-constant value for about backgroundRatio*history frames, it's considered background and added to the model as a center of a new component. It corresponds to TB parameter in the paper.	1								
video_BgSubtractorMOG2_setBackgroundRatio	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setBackgroundRatio(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setBackgroundRatio(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setBackgroundRatio()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the "background ratio" parameter of the algorithm.			1								
video_BgSubtractorMOG2_getVarThreshold	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getVarThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getVarThreshold(); END_WRAP }	cv::BackgroundSubtractorMOG2::getVarThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the variance threshold for the pixel-model match.		The main threshold on the squared Mahalanobis distance to decide if the sample is well described by the background model or not. Related to Cthr from the paper.	1								

video_BgSubtractorMOG2_setVarThreshold	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setVarThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setVarThreshold(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setVarThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the variance threshold for the pixel-model match.			1										
video_BgSubtractorMOG2_getVarThresholdGen	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getVarThresholdGen(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getVarThresholdGen(); END_WRAP }	cv::BackgroundSubtractorMOG2::getVarThresholdGen()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the variance threshold for the pixel-model match used for new mixture component generation.		Threshold for the squared Mahalanobis distance that helps decide when a sample is close to the existing components (corresponds to Tg in the paper). If a pixel is not close to any component, it is considered foreground or added as a new component. 3 sigma => Tg=3*3=9 is default. A smaller Tg value generates more components. A higher Tg value may result in a small number of components but they can grow too large.	1									
video_BgSubtractorMOG2_setVarThresholdGen	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setVarThresholdGen(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setVarThresholdGen(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setVarThresholdGen()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the variance threshold for the pixel-model match used for new mixture component generation.			1										
video_BgSubtractorMOG2_getVarInit	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getVarInit(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getVarInit(); END_WRAP }	cv::BackgroundSubtractorMOG2::getVarInit()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the initial variance of each gaussian component.			1										
video_BgSubtractorMOG2_setVarInit	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setVarInit(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setVarInit(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setVarInit()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the initial variance of each gaussian component.			1										
video_BgSubtractorMOG2_getVarMin	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getVarMin(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getVarMin(); END_WRAP }	cv::BackgroundSubtractorMOG2::getVarMin()	classcv_1_1BackgroundSubtractorMOG2.html				1										
video_BgSubtractorMOG2_setVarMin	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setVarMin(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setVarMin(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setVarMin()	classcv_1_1BackgroundSubtractorMOG2.html				1										
video_BgSubtractorMOG2_getVarMax	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getVarMax(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getVarMax(); END_WRAP }	cv::BackgroundSubtractorMOG2::getVarMax()	classcv_1_1BackgroundSubtractorMOG2.html				1										
video_BgSubtractorMOG2_setVarMax	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setVarMax(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setVarMax(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setVarMax()	classcv_1_1BackgroundSubtractorMOG2.html				1										
video_BgSubtractorMOG2_getComplexityReduc	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getComplexityReductionThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getComplexityReductionThreshold(); END_WRAP }	cv::BackgroundSubtractorMOG2::getComplexityReductionThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the complexity reduction threshold.		This parameter defines the number of samples needed to accept to prove the component exists. CT=0.05 is a default value for all the samples. By setting CT=0 you get an algorithm very similar to the standard Stauffer&Grimson algorithm.	1									
video_BgSubtractorMOG2_setComplexityReduc	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setComplexityReductionThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setComplexityReductionThreshold(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setComplexityReductionThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the complexity reduction threshold.			1										
video_BgSubtractorMOG2_getDetectShadows	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getDetectShadows(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getDetectShadows() ? 1 : 0; END_WRAP }	cv::BackgroundSubtractorMOG2::getDetectShadows()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the shadow detection flag.		If true, the algorithm detects shadows and marks them. See createBackgroundSubtractorMOG2 for details.	1									
video_BgSubtractorMOG2_setDetectShadows	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setDetectShadows(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int value) { BEGIN_WRAP (*ptr)->setDetectShadows(value != 0); END_WRAP }	cv::BackgroundSubtractorMOG2::setDetectShadows()	classcv_1_1BackgroundSubtractorMOG2.html	Enables or disables shadow detection.			1										
video_BgSubtractorMOG2_getShadowValue	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getShadowValue(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getShadowValue(); END_WRAP }	cv::BackgroundSubtractorMOG2::getShadowValue()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the shadow value.		Shadow value is the value used to mark shadows in the foreground mask. Default value is 127. Value 0 in the mask always means background, 255 means foreground.	1									
video_BgSubtractorMOG2_setShadowValue	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setShadowValue(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, int value) { BEGIN_WRAP (*ptr)->setShadowValue(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setShadowValue()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the shadow value.			1										
video_BgSubtractorMOG2_getShadowThreshol	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_getShadowThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getShadowThreshold(); END_WRAP }	cv::BackgroundSubtractorMOG2::getShadowThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Returns the shadow threshold.		A shadow is detected if pixel is a darker version of the background. The shadow threshold (Tau in the paper) is a threshold defining how much darker the shadow can be. Tau= 0.5 means that if a pixel is more than twice darker then it is not shadow. See Prati, Mikic, Trivedi and Cucchiara, Detecting Moving Shadows...*, IEEE PAMI,2003.	1									
video_BgSubtractorMOG2_setShadowThreshol	CVAPI(ExceptionStatus) video_BackgroundSubtractorMOG2_setShadowThreshold(cv::Ptr<cv::BackgroundSubtractorMOG2> *ptr, double value) { BEGIN_WRAP (*ptr)->setShadowThreshold(value); END_WRAP }	cv::BackgroundSubtractorMOG2::setShadowThreshold()	classcv_1_1BackgroundSubtractorMOG2.html	Sets the shadow threshold.			1										

video_createBackgroundSubtractorKNN	CVAPI(ExceptionStatus) video_createBackgroundSubtractorKNN(int history, double dist2Threshold, int detectShadows, cv::Ptr<cv::BackgroundSubtractorKNN> **returnValue) { BEGIN_WRAP const auto ptr = cv::createBackgroundSubtractorKNN(history, dist2Threshold, detectShadows != 0); *returnValue = clone(ptr); END_WRAP }	cv::createBackgroundSubtractorKNN()	group__video__motion.html	Creates KNN Background Subtractor.	history : Length of the history. dist2Threshold : Threshold on the squared distance between the pixel and the sample to decide whether a pixel is close to that sample. This parameter does not affect the background update. detectShadows : If true, the algorithm will detect shadows and mark them. It decreases the speed a bit, so if you do not need this feature, set the parameter to false.		3								
video_Ptr_BackgroundSubtractorKNN_delete	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractorKNN_delete(cv::Ptr<cv::BackgroundSubtractorKNN>*obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::BackgroundSubtractorKNN	classcv_1_1BackgroundSubtractorKNN.html	K-nearest neighbours - based Background/Foreground Segmentation Algorithm. The class implements the K-nearest neighbours background subtraction described in [298] . Very efficient if number of foreground pixels is low.								•		
video_Ptr_BackgroundSubtractorKNN_get	CVAPI(ExceptionStatus) video_Ptr_BackgroundSubtractorKNN_get(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, cv::BackgroundSubtractorKNN **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::BackgroundSubtractorKNN	classcv_1_1BackgroundSubtractorKNN.html	K-nearest neighbours - based Background/Foreground Segmentation Algorithm. The class implements the K-nearest neighbours background subtraction described in [298] . Very efficient if number of foreground pixels is low.									•		
video_BackgroundSubtractorKNN_getHistory	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getHistory(cv::Ptr<cv::BackgroundSubtractorKNN>*ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getHistory(); END_WRAP }	cv::BackgroundSubtractorKNN::getHistory()	classcv_1_1BackgroundSubtractorKNN.html	Returns the number of last frames that affect the background model.			1								
video_BackgroundSubtractorKNN_setHistory	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setHistory(cv::Ptr<cv::BackgroundSubtractorKNN>*ptr, int value) { BEGIN_WRAP (*ptr)->setHistory(value); END_WRAP }	cv::BackgroundSubtractorKNN::setHistory()	classcv_1_1BackgroundSubtractorKNN.html	Sets the number of last frames that affect the background model.			1								
video_BackgroundSubtractorKNN_getNSamples	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getNSamples(cv::Ptr<cv::BackgroundSubtractorKNN>*ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getNSamples(); END_WRAP }	cv::BackgroundSubtractorKNN::getNSamples()	classcv_1_1BackgroundSubtractorKNN.html	Returns the number of data samples in the background model.			1								
video_BackgroundSubtractorKNN_setNSamples	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setNSamples(cv::Ptr<cv::BackgroundSubtractorKNN>*ptr, int value) { BEGIN_WRAP (*ptr)->setNSamples(value); END_WRAP }	cv::BackgroundSubtractorKNN::setNSamples()	classcv_1_1BackgroundSubtractorKNN.html	Sets the number of data samples in the background model.		The model needs to be reinitalized to reserve memory.	1								
video_BackgroundSubtractorKNN_getDist2Thre	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getDist2Threshold(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getDist2Threshold(); END_WRAP }	cv::BackgroundSubtractorKNN::getDist2Threshold() }	classcv_1_1BackgroundSubtractorKNN.html	Returns the threshold on the squared distance between the pixel and the sample.		The threshold on the squared distance between the pixel and the sample to decide whether a pixel is close to a data sample.	1								
video_BackgroundSubtractorKNN_setDist2Thre	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setDist2Threshold(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, double value) { BEGIN_WRAP (*ptr)->setDist2Threshold(value); END_WRAP }	cv::BackgroundSubtractorKNN::setDist2Threshold() }	classcv_1_1BackgroundSubtractorKNN.html	Sets the threshold on the squared distance.			1								
video_BackgroundSubtractorKNN_getkNNSamp	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getkNNSamples(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getkNNSamples(); END_WRAP }	cv::BackgroundSubtractorKNN::getkNNSamples()	classcv_1_1BackgroundSubtractorKNN.html	Returns the number of neighbours, the k in the kNN.		K is the number of samples that need to be within dist2Threshold in order to decide that that pixel is matching the kNN background model.	1								
video_BackgroundSubtractorKNN_setkNNSamp	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setkNNSamples(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int value) { BEGIN_WRAP (*ptr)->setkNNSamples(value); END_WRAP }	cv::BackgroundSubtractorKNN::setkNNSamples()	classcv_1_1BackgroundSubtractorKNN.html	Sets the k in the kNN. How many nearest neighbours need to match.			1								
video_BackgroundSubtractorKNN_getDetectSh	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getDetectShadows(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getDetectShadows() ? 1 : 0; END_WRAP }	cv::BackgroundSubtractorKNN::getDetectShadows() }	classcv_1_1BackgroundSubtractorKNN.html	Returns the shadow detection flag.		If true, the algorithm detects shadows and marks them. See createBackgroundSubtractorKNN for details.	1								
video_BackgroundSubtractorKNN_setDetectSh	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setDetectShadows(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int value) { BEGIN_WRAP (*ptr)->setDetectShadows(value != 0); END_WRAP }	cv::BackgroundSubtractorKNN::setDetectShadows() }	classcv_1_1BackgroundSubtractorKNN.html	Enables or disables shadow detection.			1								
video_BackgroundSubtractorKNN_getShadowV	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getShadowValue(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getShadowValue(); END_WRAP }	cv::BackgroundSubtractorKNN::getShadowValue()	classcv_1_1BackgroundSubtractorKNN.html	Returns the shadow value.		Shadow value is the value used to mark shadows in the foreground mask. Default value is 127. Value 0 in the mask always means background, 255 means foreground.	1								
video_BackgroundSubtractorKNN_setShadowV	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setShadowValue(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, int value) { BEGIN_WRAP (*ptr)->setShadowValue(value); END_WRAP }	cv::BackgroundSubtractorKNN::setShadowValue()	classcv_1_1BackgroundSubtractorKNN.html	Sets the shadow value.			1								
video_BackgroundSubtractorKNN_getShadowT	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_getShadowThreshold(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, double *returnValue) { BEGIN_WRAP *returnValue = (*ptr)->getShadowThreshold(); END_WRAP }	cv::BackgroundSubtractorKNN::getShadowThreshold() }	classcv_1_1BackgroundSubtractorKNN.html	Returns the shadow threshold.		A shadow is detected if pixel is a darker version of the background. The shadow threshold (Tau in the paper) is a threshold defining how much darker the shadow can be. Tau= 0.5 means that if a pixel is more than twice darker then it is not shadow. See Prati, Mikic, Trivedi and Cucchiara, Detecting Moving Shadows...*, IEEE PAMI,2003.	1								

video_BackgroundSubtractorKNN_setShadowTl	CVAPI(ExceptionStatus) video_BackgroundSubtractorKNN_setShadowThreshold(cv::Ptr<cv::BackgroundSubtractorKNN> *ptr, double value) { BEGIN_WRAP (*ptr)->setShadowThreshold(value); END_WRAP }	cv::BackgroundSubtractorKNN::setShadowThreshold()	classcv_1_1BackgroundSubtractorKNN.html	Sets the shadow threshold.			1								
video_CamShift	CVAPI(ExceptionStatus) video_CamShift(cv::_InputArray *probImage, MyCvRect *window, MyCvTermCriteria criteria, MyCvBox2D *returnValue) { BEGIN_WRAP cv::Rect window0 = cpp(*window); const auto ret = cv::CamShift(*probImage, window0, cpp(criteria)); *window = c(window0); *returnValue = c(ret); END_WRAP }	cv::CamShift()	group__video__track.html	Finds an object center, size, and orientation.	probImage : Back projection of the object histogram. See calcBackProject. window : Initial search window. criteria : Stop criteria for the underlying meanShift. returns (in old interfaces) Number of iterations CAMSHIFT took to converge The function implements the CAMSHIFT object tracking algorithm [33] . First, it finds an object center using meanShift and then adjusts the window size and finds the optimal rotation. The function returns the rotated rectangle structure that includes the object position, size, and orientation. The next position of the search window can be obtained with RotatedRect::boundingRect()	See the OpenCV sample camshiftdemo.c that tracks colored objects.Note (Python) A sample explaining the camshift tracking algorithm can be found at opencv_source_code/samples/python/camshift.pyExamples: samples/cpp/camshiftdemo.cpp.	3								
video_meanShift	CVAPI(ExceptionStatus) video_meanShift(cv::_InputArray *probImage, MyCvRect *window, MyCvTermCriteria criteria, int *returnValue) { BEGIN_WRAP cv::Rect window0 = cpp(*window); const auto ret = cv::meanShift(*probImage, window0, cpp(criteria)); *window = c(window0); *returnValue = ret; END_WRAP }	cv::meanShift()	group__video__track.html	Finds an object on a back projection image.	probImage : Back projection of the object histogram. See calcBackProject for details. window : Initial search window. criteria : Stop criteria for the iterative search algorithm. returns : Number of iterations CAMSHIFT took to converge. The function implements the iterative object search algorithm. It takes the input back projection of an object and the initial position. The mass center in window of the back projection image is computed and the search window center shifts to the mass center. The procedure is repeated until the specified number of iterations criteria.maxCount is done or until the window center shifts by less than criteria.epsilon. The algorithm is used inside CamShift and, unlike CamShift , the search window size or orientation do not change during the search. You can simply pass the output of calcBackProject to this function. But better results can be obtained if you pre-filter the back projection and remove the noise. For example, you can do this by retrieving connected components with findContours , throwing away contours with small area (contourArea), and rendering the remaining contours with drawContours.		3								
video_buildOpticalFlowPyramid1	CVAPI(ExceptionStatus) video_buildOpticalFlowPyramid1(cv::_InputArray* img, cv::_OutputArray* pyramid, MyCvSize winSize, int maxLevel, int withDerivatives, int pyrBorder, int derivBorder, int tryReuseInputImage, int* returnValue) { BEGIN_WRAP * returnValue = cv::buildOpticalFlowPyramid(*img, *pyramid, cpp(winSize), maxLevel, withDerivatives != 0, pyrBorder, derivBorder, tryReuseInputImage != 0); END_WRAP }	cv::buildOpticalFlowPyramid()	group__video__track.html	Constructs the image pyramid which can be passed to calcOpticalFlowPyrLK.	img : 8-bit input image. pyramid : output pyramid. winSize : window size of optical flow algorithm. Must be not less than winSize argument of calcOpticalFlowPyrLK. It is needed to calculate required padding for pyramid levels. maxLevel : 0-based maximal pyramid level number. withDerivatives : set to precompute gradients for the every pyramid level. If pyramid is constructed without the gradients then calcOpticalFlowPyrLK will calculate them internally. pyrBorder : the border mode for pyramid layers. derivBorder : the border mode for gradients. tryReuseInputImage : put ROI of input image into the pyramid if possible. You can pass false to force data copying.		8								
video_buildOpticalFlowPyramid2	CVAPI(ExceptionStatus) video_buildOpticalFlowPyramid2(cv::_InputArray* img, std::vector<cv::Mat>* pyramidVec, MyCvSize winSize, int maxLevel, int withDerivatives, int pyrBorder, int derivBorder, int tryReuseInputImage, int* returnValue) { BEGIN_WRAP * returnValue = cv::buildOpticalFlowPyramid(*img, *pyramidVec, cpp(winSize), maxLevel, withDerivatives != 0, pyrBorder, derivBorder, tryReuseInputImage != 0); END_WRAP }	cv::buildOpticalFlowPyramid()	group__video__track.html	Constructs the image pyramid which can be passed to calcOpticalFlowPyrLK.	img : 8-bit input image. pyramid : output pyramid. winSize : window size of optical flow algorithm. Must be not less than winSize argument of calcOpticalFlowPyrLK. It is needed to calculate required padding for pyramid levels. maxLevel : 0-based maximal pyramid level number. withDerivatives : set to precompute gradients for the every pyramid level. If pyramid is constructed without the gradients then calcOpticalFlowPyrLK will calculate them internally. pyrBorder : the border mode for pyramid layers. derivBorder : the border mode for gradients. tryReuseInputImage : put ROI of input image into the pyramid if possible. You can pass false to force data copying.		8								
video_calcOpticalFlowPyrLK_InputArray	CVAPI(ExceptionStatus) video_calcOpticalFlowPyrLK_InputArray(cv::_InputArray* prevImg, cv::_InputArray* nextImg, cv::_InputArray* prevPts, cv::_InputOutputArray* nextPts, cv::_OutputArray* status, cv::_OutputArray* err, MyCvSize winSize, int maxLevel, MyCvTermCriteria criteria, int flags, double minEigThreshold) { BEGIN_WRAP cv::calcOpticalFlowPyrLK(*prevImg, *nextImg, *prevPts, *nextPts, *status, *err, cpp(winSize), maxLevel, cpp(criteria), flags, minEigThreshold); END_WRAP }	cv::calcOpticalFlowPyrLK()	group__video__track.html	Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.	prevImg : first 8-bit input image or pyramid constructed by buildOpticalFlowPyramid. nextImg : second input image or pyramid of the same size and the same type as prevImg. prevPts : vector of 2D points for which the flow needs to be found; point coordinates must be single-precision floating-point numbers. nextPts : output vector of 2D points (with single-precision floating-point coordinates) containing the calculated new positions of input features in the second image; when OPTFLOW_USE_INITIAL_FLOW flag is passed, the vector must have the same size as in the input. status : output status vector (of unsigned chars); each element of the vector is set to 1 if the flow for the corresponding features has been found, otherwise, it is set to 0. err : output vector of errors; each element of the vector is set to an error for the corresponding feature, type of the error measure can be set in flags parameter; if the flow wasn't found then the error is not defined (use the status parameter to find such cases). winSize : size of the search window at each pyramid level. maxLevel : 0-based maximal pyramid level number; if	The function implements a sparse iterative version of the Lucas-Kanade optical flow in pyramids. See [30] . The function is parallelized with the TBB library.NoteAn example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/cpp/lkdemo.cpp (Python) An example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/python/lk_track.py (Python) An example using the Lucas-Kanade tracker for homography matching can be found at opencv_source_code/samples/python/lk_homography.pyExamples: samples/cpp/lkdemo.cpp.	11								
video_calcOpticalFlowPyrLK_vector	CVAPI(ExceptionStatus) video_calcOpticalFlowPyrLK_vector(cv::_InputArray* prevImg, cv::_InputArray* nextImg, cv::Point2f* prevPts, int prevPtsSize, std::vector<cv::Point2f>* nextPts, std::vector<uchar>* status, std::vector<float>* err, MyCvSize winSize, int maxLevel, MyCvTermCriteria criteria, int flags, double minEigThreshold) { BEGIN_WRAP const std::vector<cv::Point2f> prevPtsVec(prevPts, prevPts + prevPtsSize); cv::calcOpticalFlowPyrLK(*prevImg, *nextImg, prevPtsVec,	cv::calcOpticalFlowPyrLK()	group__video__track.html	Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.	prevImg : first 8-bit input image or pyramid constructed by buildOpticalFlowPyramid. nextImg : second input image or pyramid of the same size and the same type as prevImg. prevPts : vector of 2D points for which the flow needs to be found; point coordinates must be single-precision floating-point numbers. nextPts : output vector of 2D points (with single-precision floating-point coordinates) containing the calculated new positions of input features in the second image; when OPTFLOW_USE_INITIAL_FLOW flag is passed, the vector must have the same size as in the input. status : output status vector (of unsigned chars); each element of the vector is set to 1 if the flow for the corresponding features has been found, otherwise, it is set to 0. err : output vector of errors; each element of the vector is set to an error for the corresponding feature, type of the error measure can be set in flags parameter; if the flow wasn't found then the error is not defined (use the status parameter to find such cases). winSize : size of the search window at each pyramid level. maxLevel : 0-based maximal pyramid level number; if	The function implements a sparse iterative version of the Lucas-Kanade optical flow in pyramids. See [30] . The function is parallelized with the TBB library.NoteAn example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/cpp/lkdemo.cpp (Python) An example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/python/lk_track.py (Python) An example using the Lucas-Kanade tracker for homography matching can be found at opencv_source_code/samples/python/lk_homography.pyExamples: samples/cpp/lkdemo.cpp.	11								
video_calcOpticalFlowFarneback	CVAPI(ExceptionStatus) video_calcOpticalFlowFarneback(cv::_InputArray* prev, cv::_InputArray* next, cv::_InputOutputArray* flow, double pyrScale, int levels, int winSize, int iterations, int polyN, double polySigma, int flags) { BEGIN_WRAP cv::calcOpticalFlowFarneback(*prev, *next, *flow, pyrScale, levels, winSize, iterations, polyN, polySigma, flags); END_WRAP }	cv::calcOpticalFlowFarneback()	group__video__track.html	Computes a dense optical flow using the Gunnar Farneback's algorithm.	prev : first 8-bit single-channel input image. next : second input image of the same size and the same type as prev. flow : computed flow image that has the same size as prev and type CV_32FC2. pyr_scale : parameter, specifying the image scale (<1) to build pyramids for each image; pyr_scale=0.5 means a classical pyramid, where each next layer is twice smaller than the previous one. levels : number of pyramid layers including the initial image; levels=1 means that no extra layers are created and only the original images are used. winsize : averaging window size; larger values increase the algorithm robustness to image noise and give more chances for fast motion detection, but yield more blurred motion field. iterations : number of iterations the algorithm does at each pyramid level. poly_n : size of the pixel neighborhood used to find polynomial expansion in each pixel; larger values mean that the image will be approximated with smoother surfaces, yielding more robust algorithm and more blurred motion field, typically poly_n =5 or 7. poly_sigma : standard deviation of the Gaussian that is used to smooth derivatives used as a basis for the polynomial expansion; for poly_n=5, you can set	The function finds an optical flow for each prev pixel using the [69] algorithm so that¥[¥texttt{prev} (y,x) ¥sim ¥texttt{next} (y + ¥texttt{flow} (y,x)[1], x + ¥texttt{flow} (y,x)[0])¥]NoteAn example using the optical flow algorithm described by Gunnar Farneback can be found at opencv_source_code/samples/cpp/fback.cpp (Python) An example using the optical flow algorithm described by Gunnar Farneback can be found at opencv_source_code/samples/python/opt_flow.py	10								
video_computeECC	CVAPI(ExceptionStatus) video_computeECC(cv::_InputArray *templateImage, cv::_InputArray *inputImage, cv::_InputArray *inputMask, double *returnValue) { BEGIN_WRAP *returnValue = cv::computeECC(*templateImage, *inputImage, entity(inputMask)); END_WRAP }	cv::computeECC()	group__video__track.html	Computes the Enhanced Correlation Coefficient value between two images [67] .	templateImage : single-channel template image; CV_8U or CV_32F array. inputImage : single-channel input image to be warped to provide an image similar to templateImage, same type as templateImage. inputMask : An optional mask to indicate valid values of inputImage.	See alsofindTransformECC	3								

video_findTransformECC1	CVAPI(ExceptionStatus) video_findTransformECC1(cv::_InputArray *templateImage, cv::_InputArray *inputImage, cv::_InputOutputArray *warpMatrix, int motionType, MyCvTermCriteria criteria, cv::_InputArray *inputMask, int gaussFiltSize, double *returnValue) { BEGIN_WRAP *returnValue = cv::findTransformECC(*templateImage, *inputImage, *warpMatrix, motionType, cpp(criteria),entity(inputMask) , gaussFiltSize); END_WRAP }	cv::findTransformECC()	group__video__track.html	Finds the geometric transform (warp) between two images in terms of the ECC criterion [67] .	templateImage : single-channel template image; CV_8U or CV_32F array. inputImage : single-channel input image which should be warped with the final warpMatrix in order to provide an image similar to templateImage, same type as templateImage. warpMatrix : floating-point ¥(2¥times 3¥) or ¥(3¥times 3¥) mapping matrix (warp). motionType : parameter, specifying the type of motion: MOTION_TRANSLATION sets a translational motion model; warpMatrix is ¥(2¥times 3¥) with the first ¥(2¥times 2¥) part being the unity matrix and the rest two parameters being estimated. MOTION_EUCLIDEAN sets a Euclidean (rigid) transformation as motion model; three parameters are estimated; warpMatrix is ¥(2¥times 3¥). MOTION_AFFINE sets an affine motion model (DEFAULT); six parameters are estimated; warpMatrix is ¥(2¥times 3¥). MOTION_HOMOGRAPHY sets a homography as a motion model; eight parameters are estimated;` warpMatrix` is ¥(3¥times 3¥). criteria : parameter, specifying the termination criteria of the ECC algorithm; criteria.epsilon defines the threshold of the increment in the correlation coefficient	The function estimates the optimum transformation (warpMatrix) with respect to ECC criterion ([67]), that ¡æ¥[¥textt{warpMatrix} = ¥arg¥max_{W} ¥textt{ECC}(¥textit{templateImage}(x,y),¥textit{inputImage}(x',y'))¥]¥where¥[¥begin{bmatrix} x' ¥¥ y' ¥end{bmatrix} = W ¥cdot ¥begin{bmatrix} x ¥¥ y ¥¥ 1 ¥end{bmatrix}¥](the equation holds with homogeneous coordinates for homography). It returns the final enhanced correlation coefficient, that is the correlation coefficient between the template image and the final warped input image. When a ¥(3¥times 3¥) matrix is given with motionType =0, 1 or 2, the third row is ignored.Unlike findHomography and estimateRigidTransform, the function findTransformECC implements an area-based alignment that builds on intensity similarities. In essence, the function updates the initial transformation that roughly aligns the images. If this information is missing, the identity warp	7								
video_findTransformECC2	CVAPI(ExceptionStatus) video_findTransformECC2(cv::_InputArray *templateImage, cv::_InputArray *inputImage, cv::_InputOutputArray *warpMatrix, int motionType, MyCvTermCriteria criteria, cv::_InputArray *inputMask, double* returnValue) { BEGIN_WRAP *returnValue = cv::findTransformECC(*templateImage, *inputImage, *warpMatrix, motionType, cpp(criteria), entity(inputMask)); END_WRAP }	cv::findTransformECC()	group__video__track.html	Finds the geometric transform (warp) between two images in terms of the ECC criterion [67] .	templateImage : single-channel template image; CV_8U or CV_32F array. inputImage : single-channel input image which should be warped with the final warpMatrix in order to provide an image similar to templateImage, same type as templateImage. warpMatrix : floating-point ¥(2¥times 3¥) or ¥(3¥times 3¥) mapping matrix (warp). motionType : parameter, specifying the type of motion: MOTION_TRANSLATION sets a translational motion model; warpMatrix is ¥(2¥times 3¥) with the first ¥(2¥times 2¥) part being the unity matrix and the rest two parameters being estimated. MOTION_EUCLIDEAN sets a Euclidean (rigid) transformation as motion model; three parameters are estimated; warpMatrix is ¥(2¥times 3¥). MOTION_AFFINE sets an affine motion model (DEFAULT); six parameters are estimated; warpMatrix is ¥(2¥times 3¥). MOTION_HOMOGRAPHY sets a homography as a motion model; eight parameters are estimated;` warpMatrix` is ¥(3¥times 3¥). criteria : parameter, specifying the termination criteria of the ECC algorithm; criteria.epsilon defines the threshold of the increment in the correlation coefficient	The function estimates the optimum transformation (warpMatrix) with respect to ECC criterion ([67]), that ¡æ¥[¥textt{warpMatrix} = ¥arg¥max_{W} ¥textt{ECC}(¥textit{templateImage}(x,y),¥textit{inputImage}(x',y'))¥]¥where¥[¥begin{bmatrix} x' ¥¥ y' ¥end{bmatrix} = W ¥cdot ¥begin{bmatrix} x ¥¥ y ¥¥ 1 ¥end{bmatrix}¥](the equation holds with homogeneous coordinates for homography). It returns the final enhanced correlation coefficient, that is the correlation coefficient between the template image and the final warped input image. When a ¥(3¥times 3¥) matrix is given with motionType =0, 1 or 2, the third row is ignored.Unlike findHomography and estimateRigidTransform, the function findTransformECC implements an area-based alignment that builds on intensity similarities. In essence, the function updates the initial transformation that roughly aligns the images. If this information is missing, the identity warp	6								
video_KalmanFilter_new1	CVAPI(ExceptionStatus) video_KalmanFilter_new1(cv::KalmanFilter **returnValue) { BEGIN_WRAP *returnValue = new cv::KalmanFilter; END_WRAP }	cv::KalmanFilter	classcv_1_1KalmanFilter.html	Kalman filter class. The class implements a standard Kalman filter http://en.wikipedia.org/wiki/Kalman_filter, [272] . However, you can modify transitionMatrix, controlMatrix, and measurementMatrix to get an extended Kalman filter functionality. NoteIn C API when CvKalman* kalmanFilter structure is not needed anymore, it should be released with cvReleaseKalman(&kalmanFilter) Examples: samples/cpp/kalman.cpp.							•				
video_KalmanFilter_new2	CVAPI(ExceptionStatus) video_KalmanFilter_new2(int dynamParams, int measureParams, int controlParams, int type, cv::KalmanFilter **returnValue) { BEGIN_WRAP *returnValue = new cv::KalmanFilter(dynamParams, measureParams, controlParams, type); END_WRAP }	cv::KalmanFilter	classcv_1_1KalmanFilter.html	Kalman filter class. The class implements a standard Kalman filter http://en.wikipedia.org/wiki/Kalman_filter, [272] . However, you can modify transitionMatrix, controlMatrix, and measurementMatrix to get an extended Kalman filter functionality. NoteIn C API when CvKalman* kalmanFilter structure is not needed anymore, it should be released with cvReleaseKalman(&kalmanFilter) Examples: samples/cpp/kalman.cpp.											
video_KalmanFilter_init	CVAPI(ExceptionStatus) video_KalmanFilter_init((cv::KalmanFilter *obj, int dynamParams, int measureParams, int controlParams, int type) { BEGIN_WRAP obj->init(dynamParams, measureParams, controlParams, type); END_WRAP }	cv::KalmanFilter::init()	classcv_1_1KalmanFilter.html	Re-initializes Kalman filter. The previous content is destroyed.	dynamParams : Dimensionality of the state. measureParams : Dimensionality of the measurement. controlParams : Dimensionality of the control vector. type : Type of the created matrices that should be CV_32F or CV_64F.		4									
video_KalmanFilter_delete	CVAPI(ExceptionStatus) video_KalmanFilter_delete(cv::KalmanFilter *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::KalmanFilter	classcv_1_1KalmanFilter.html	Kalman filter class. The class implements a standard Kalman filter http://en.wikipedia.org/wiki/Kalman_filter, [272] . However, you can modify transitionMatrix, controlMatrix, and measurementMatrix to get an extended Kalman filter functionality. NoteIn C API when CvKalman* kalmanFilter structure is not needed anymore, it should be released with cvReleaseKalman(&kalmanFilter) Examples: samples/cpp/kalman.cpp.								•			
video_KalmanFilter_predict	CVAPI(ExceptionStatus) video_KalmanFilter_predict(cv::KalmanFilter *obj, cv::Mat *control, cv::Mat **returnValue) { BEGIN_WRAP const auto result = obj->predict(*entity(control)); *returnValue = new cv::Mat(result); END_WRAP }	cv::KalmanFilter::predict()	classcv_1_1KalmanFilter.html	Computes a predicted state.	control : The optional input control	Examples: samples/cpp/kalman.cpp.	1									
video_KalmanFilter_correct	CVAPI(ExceptionStatus) video_KalmanFilter_correct(cv::KalmanFilter *obj, cv::Mat *measurement, cv::Mat **returnValue) { BEGIN_WRAP const auto result = obj->correct(*measurement); *returnValue = new cv::Mat(result); END_WRAP }	cv::KalmanFilter::correct()	classcv_1_1KalmanFilter.html	Updates the predicted state from the measurement.	measurement : The measured system parameters	Examples: samples/cpp/kalman.cpp.	1									
video_KalmanFilter_statePre	CVAPI(ExceptionStatus) video_KalmanFilter_statePre(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->statePre); END_WRAP }	cv::KalmanFilter::statePre	classcv_1_1KalmanFilter.html	predicted state (x'(k)): x(k)=A*x(k-1)+B*u(k)								•				
video_KalmanFilter_statePost	CVAPI(ExceptionStatus) video_KalmanFilter_statePost(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->statePost); END_WRAP }	cv::KalmanFilter::statePost	classcv_1_1KalmanFilter.html	corrected state (x(k)): x(k)=x'(k)+K(k)*(z(k)-H*x'(k))		Examples: samples/cpp/kalman.cpp.						•				
video_KalmanFilter_transitionMatrix	CVAPI(ExceptionStatus) video_KalmanFilter_transitionMatrix(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->transitionMatrix); END_WRAP }	cv::KalmanFilter::transitionMatrix	classcv_1_1KalmanFilter.html	state transition matrix (A)		Examples: samples/cpp/kalman.cpp.						•				
video_KalmanFilter_controlMatrix	CVAPI(ExceptionStatus) video_KalmanFilter_controlMatrix(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->controlMatrix); END_WRAP }	cv::KalmanFilter::controlMatrix	classcv_1_1KalmanFilter.html	control matrix (B) (not used if there is no control)								•				
video_KalmanFilter_measurementMatrix	CVAPI(ExceptionStatus) video_KalmanFilter_measurementMatrix(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->measurementMatrix); END_WRAP }	cv::KalmanFilter::measurementMatrix	classcv_1_1KalmanFilter.html	measurement matrix (H)		Examples: samples/cpp/kalman.cpp.						•				
video_KalmanFilter_processNoiseCov	CVAPI(ExceptionStatus) video_KalmanFilter_processNoiseCov(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->processNoiseCov); END_WRAP }	cv::KalmanFilter::processNoiseCov	classcv_1_1KalmanFilter.html	process noise covariance matrix (Q)		Examples: samples/cpp/kalman.cpp.						•				
video_KalmanFilter_measurementNoiseCov	CVAPI(ExceptionStatus) video_KalmanFilter_measurementNoiseCov(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->measurementNoiseCov); END_WRAP }	cv::KalmanFilter::measurementNoiseCov	classcv_1_1KalmanFilter.html	measurement noise covariance matrix (R)		Examples: samples/cpp/kalman.cpp.						•				

video_KalmanFilter_errorCovPre	CVAPI(ExceptionStatus) video_KalmanFilter_errorCovPre(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->errorCovPre); END_WRAP }	cv::KalmanFilter::errorCovPre	classcv_1_1KalmanFilter.html	priori error estimate covariance matrix (P'(k)): P'(k)=A*P(k-1)*At + Q)*/						•					
video_KalmanFilter_gain	CVAPI(ExceptionStatus) video_KalmanFilter_gain(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->gain); END_WRAP }	cv::KalmanFilter::gain	classcv_1_1KalmanFilter.html	Kalman gain matrix (K(k)): K(k)=P'(k)*Ht*inv(H*P'(k)*Ht+R)						•					
video_KalmanFilter_errorCovPost	CVAPI(ExceptionStatus) video_KalmanFilter_errorCovPost(cv::KalmanFilter *obj, cv::Mat **returnValue) { BEGIN_WRAP *returnValue = &(obj->errorCovPost); END_WRAP }	cv::KalmanFilter::errorCovPost	classcv_1_1KalmanFilter.html	posteriori error estimate covariance matrix (P(k)): P(k)=(I-K(k)*H)*P'(k)		Examples: samples/cpp/kalman.cpp.				•					
video_Tracker_init	CVAPI(ExceptionStatus) video_Tracker_init(cv::Tracker* tracker, const cv::Mat* image, const MyCvRect boundingBox) { BEGIN_WRAP tracker->init(*image, cpp(boundingBox)); END_WRAP }	cv::Tracker::init()	classcv_1_1Tracker.html	Initialize the tracker with a known bounding box that surrounded the target.	image : The initial frame boundingBox : The initial bounding box		2								
video_Tracker_update	CVAPI(ExceptionStatus) video_Tracker_update(cv::Tracker* tracker, const cv::Mat* image, MyCvRect* boundingBox, int* returnValue) { BEGIN_WRAP cv::Rect bb = cpp(*boundingBox); const bool ret = tracker->update(*image, bb); if (ret) { boundingBox->x = bb.x; boundingBox->y = bb.y; boundingBox->width = bb.width; boundingBox->height = bb.height; } *returnValue = ret ? 1 : 0; END_WRAP }	cv::Tracker::update()	classcv_1_1Tracker.html	Update the tracker, find the new most likely bounding box for the target.	image : The current frame boundingBox : The bounding box that represent the new target location, if true was returned, not modified otherwise		2								
video_TrackerMIL_create1	CVAPI(ExceptionStatus) video_TrackerMIL_create1(cv::Ptr<cv::TrackerMIL>** returnValue) { BEGIN_WRAP const auto p = cv::TrackerMIL::create(); *returnValue = clone(p); END_WRAP }	cv::TrackerMIL::create()	classcv_1_1TrackerMIL.html	Create MIL tracker instance.	parameters : MIL parameters TrackerMIL::Params		1								
video_TrackerMIL_create2	CVAPI(ExceptionStatus) video_TrackerMIL_create2(cv::TrackerMIL::Params* parameters, cv::Ptr<cv::TrackerMIL>** returnValue) { BEGIN_WRAP const auto p = cv::TrackerMIL::create(*parameters); *returnValue = clone(p); END_WRAP }	cv::TrackerMIL::create()	classcv_1_1TrackerMIL.html	Create MIL tracker instance.	parameters : MIL parameters TrackerMIL::Params		1								
video_Ptr_TrackerMIL_delete	CVAPI(ExceptionStatus) video_Ptr_TrackerMIL_delete(cv::Ptr<cv::TrackerMIL>* ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TrackerMIL	classcv_1_1TrackerMIL.html	The MIL algorithm trains a classifier in an online manner to separate the object from the background. Multiple Instance Learning avoids the drift problem for a robust tracking. The implementation is based on [13] . Original code can be found here http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml								•		
video_Ptr_TrackerMIL_get	CVAPI(ExceptionStatus) video_Ptr_TrackerMIL_get(cv::Ptr<cv::TrackerMIL>* ptr, cv::TrackerMIL** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TrackerMIL	classcv_1_1TrackerMIL.html	The MIL algorithm trains a classifier in an online manner to separate the object from the background. Multiple Instance Learning avoids the drift problem for a robust tracking. The implementation is based on [13] . Original code can be found here http://vision.ucsd.edu/~bbabenko/project_miltrack.shtml									•	
video_TrackerGOTURN_create1	CVAPI(ExceptionStatus) video_TrackerGOTURN_create1(cv::Ptr<cv::TrackerGOTURN>** returnValue) { BEGIN_WRAP const auto p = cv::TrackerGOTURN::create(); *returnValue = clone(p); END_WRAP }	cv::TrackerGOTURN::create()	classcv_1_1TrackerGOTURN.html	Constructor.	parameters : GOTURN parameters TrackerGOTURN::Params		1								
video_TrackerGOTURN_create2	CVAPI(ExceptionStatus) video_TrackerGOTURN_create2(cv::TrackerGOTURN::Params* parameters, cv::Ptr<cv::TrackerGOTURN>** returnValue) { BEGIN_WRAP const auto p = cv::TrackerGOTURN::create(*parameters); *returnValue = clone(p); END_WRAP }	cv::TrackerGOTURN::create()	classcv_1_1TrackerGOTURN.html	Constructor.	parameters : GOTURN parameters TrackerGOTURN::Params		1								
video_Ptr_TrackerGOTURN_delete	CVAPI(ExceptionStatus) video_Ptr_TrackerGOTURN_delete(cv::Ptr<cv::TrackerGOTURN>* ptr) { BEGIN_WRAP delete ptr; END_WRAP }	cv::TrackerGOTURN	classcv_1_1TrackerGOTURN.html	the GOTURN (Generic Object Tracking Using Regression Networks) tracker GOTURN ([107]) is kind of trackers based on Convolutional Neural Networks (CNN). While taking all advantages of CNN trackers, GOTURN is much faster due to offline training without online fine-tuning nature. GOTURN tracker addresses the problem of single target tracking: given a bounding box label of an object in the first frame of the video, we track that object through the rest of the video. NOTE: Current method of GOTURN does not handle occlusions; however, it is fairly robust to viewpoint changes, lighting changes, and deformations. Inputs of GOTURN are two RGB patches representing Target and Search patches resized to 227x227. Outputs of GOTURN are predicted bounding box coordinates, relative to Search patch coordinate system, in format X1,Y1,X2,Y2. Original paper is here: http://davheld.github.io/GOTURN/GOTURN.pdf As long as original authors implementation: https://github.com/davheld/GOTURN#train-the-tracker Implementation of training algorithm is placed in separately here due to 3d-party dependencies: https://github.com/Auron-X/GOTURN_Training_Toolkit GOTURN architecture goturn.prototxt and trained model								•		
video_Ptr_TrackerGOTURN_get	CVAPI(ExceptionStatus) video_Ptr_TrackerGOTURN_get(cv::Ptr<cv::TrackerGOTURN>* ptr, cv::TrackerGOTURN** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::TrackerGOTURN	classcv_1_1TrackerGOTURN.html	the GOTURN (Generic Object Tracking Using Regression Networks) tracker GOTURN ([107]) is kind of trackers based on Convolutional Neural Networks (CNN). While taking all advantages of CNN trackers, GOTURN is much faster due to offline training without online fine-tuning nature. GOTURN tracker addresses the problem of single target tracking: given a bounding box label of an object in the first frame of the video, we track that object through the rest of the video. NOTE: Current method of GOTURN does not handle occlusions; however, it is fairly robust to viewpoint changes, lighting changes, and deformations. Inputs of GOTURN are two RGB patches representing Target and Search patches resized to 227x227. Outputs of GOTURN are predicted bounding box coordinates, relative to Search patch coordinate system, in format X1,Y1,X2,Y2. Original paper is here: http://davheld.github.io/GOTURN/GOTURN.pdf As long as original authors implementation: https://github.com/davheld/GOTURN#train-the-tracker Implementation of training algorithm is placed in separately here due to 3d-party dependencies: https://github.com/Auron-X/GOTURN_Training_Toolkit GOTURN architecture goturn.prototxt and trained model										•

ximgproc_niBlackThreshold	CVAPI(ExceptionStatus) ximgproc_niBlackThreshold(cv::_InputArray *src, cv::_OutputArray *dst, double maxValue, int type, int blockSize, double k, int binarizationMethod, double r) { BEGIN_WRAP cv::ximgproc::niBlackThreshold(*src, *dst, maxValue, type, blockSize, k, binarizationMethod, r); END_WRAP }	cv::ximgproc::niBlackThreshold()	group__ximgproc.html	Performs thresholding on input images using Niblack's technique or some of the popular variations it inspired.	_src : Source 8-bit single-channel image. _dst : Destination image of the same size and the same type as src. maxValue : Non-zero value assigned to the pixels for which the condition is satisfied, used with the THRESH_BINARY and THRESH_BINARY_INV thresholding types. type : Thresholding type, see cv::ThresholdTypes. blockSize : Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on. k : The user-adjustable parameter used by Niblack and inspired techniques. For Niblack, this is normally a value between 0 and 1 that is multiplied with the standard deviation and subtracted from the mean. binarizationMethod : Binarization method to use. By default, Niblack's technique is used. Other techniques can be specified, see cv::ximgproc::LocalBinarizationMethods. r : The user-adjustable parameter used by Sauvola's technique. This is the dynamic range of standard deviation.	The function transforms a grayscale image to a binary image according to the formulae:THRESH_BINARY ¥[dst(x,y) = ¥fork{¥texttt{maxValue}}{if ¥(src(x,y) > T(x,y)¥)}{0}{otherwise}¥] THRESH_BINARY_INV ¥[dst(x,y) = ¥fork{0}{if ¥(src(x,y) > T(x,y)¥)}{¥texttt{maxValue}}{otherwise}¥] where ¥(T(x,y)¥) is a threshold calculated individually for each pixel.The threshold value ¥(T(x, y)¥) is determined based on the binarization method chosen. For classic Niblack, it is the mean minus ¥(k ¥) times standard deviation of ¥(¥texttt{blockSize}¥) ¥times¥texttt{blockSize}¥) neighborhood of ¥((x, y)¥).The function can't process the image in-place.See alsothreshold, adaptiveThreshold		8								
ximgproc_thinning	CVAPI(ExceptionStatus) ximgproc_thinning(cv::_InputArray *src, cv::_OutputArray *dst, int thinningType) { BEGIN_WRAP cv::ximgproc::thinning(*src, *dst, thinningType); END_WRAP }	cv::ximgproc::thinning()	group__ximgproc.html	Applies a binary blob thinning operation, to achieve a skeletization of the input image.	src : Source 8-bit single-channel image, containing binary blobs, with blobs having 255 pixel values. dst : Destination image of the same size and the same type as src. The function can work in-place. thinningType : Value that defines which thinning algorithm should be used. See cv::ximgproc::ThinningTypes	The function transforms a binary blob image into a skeletized form using the technique of Zhang-Suen.		3								
ximgproc_anisotropicDiffusion	CVAPI(ExceptionStatus) ximgproc_anisotropicDiffusion(cv::_InputArray *src, cv::_OutputArray *dst, float alpha, float K, int niters) { BEGIN_WRAP cv::ximgproc::anisotropicDiffusion(*src, *dst, alpha, K, niters); END_WRAP }	cv::ximgproc::anisotropicDiffusion()	group__ximgproc.html	Performs anisotropic diffusion on an image.	src : Source image with 3 channels. dst : Destination image of the same size and the same number of channels as src . alpha : The amount of time to step forward by on each iteration (normally, it's between 0 and 1). K : sensitivity to the edges niters : The number of iterations	The function applies Perona-Malik anisotropic diffusion to an image. This is the solution to the partial differential equation:¥[{¥frac {¥partial I}{¥partial t}}={¥mathrm {div}}¥left(c(x,y,t)¥nabla I¥right)=¥nabla c¥cdot ¥nabla I+c(x,y,t)¥Delta I¥]Suggested functions for c(x,y,t) are:¥[c¥left(¥|¥nabla I¥right|¤=¤{1¤¥left(¥|¥nabla I¥|/K¥right)^{2}}}¥]or¥[c¥left(¥|¥nabla I¥|¥right)=¤{¥frac {1}{1+¥left({¥frac {¥|¥nabla I¥|}{K}}¥right)^{2}}} ¥]		5								
ximgproc_BrightEdges	CVAPI(ExceptionStatus) ximgproc_BrightEdges(cv::Mat *original, cv::Mat *edgeview, int contrast, int shortrange, int longrange) { BEGIN_WRAP cv::ximgproc::BrightEdges(*original, *edgeview, contrast, shortrange, longrange); END_WRAP }	cv::ximgproc::BrightEdges()	namespacecv_1_1ximgproc.html					5								
ximgproc_createQuaternionImage	CVAPI(ExceptionStatus) ximgproc_createQuaternionImage(cv::_InputArray *img, cv::_OutputArray *qimg) { BEGIN_WRAP cv::ximgproc::createQuaternionImage(*img, *qimg); END_WRAP }	cv::ximgproc::createQuaternionImage()	group__ximgproc__filters.html	creates a quaternion image.	img : Source 8-bit, 32-bit or 64-bit image, with 3-channel image. qimg : result CV_64FC4 a quaternion image(4 chanels zero channel and B,G,R).		2								
ximgproc_qconj	CVAPI(ExceptionStatus) ximgproc_qconj(cv::_InputArray *qimg, cv::_OutputArray *qcimg) { BEGIN_WRAP cv::ximgproc::qconj(*qimg, *qcimg); END_WRAP }	cv::ximgproc::qconj()	group__ximgproc__filters.html	calculates conjugate of a quaternion image.	qimg : quaternion image. qcimg : conjugate of qimg		2								
ximgproc_qunitary	CVAPI(ExceptionStatus) ximgproc_qunitary(cv::_InputArray *qimg, cv::_OutputArray *qnimg) { BEGIN_WRAP cv::ximgproc::qunitary(*qimg, *qnimg); END_WRAP }	cv::ximgproc::qunitary()	group__ximgproc__filters.html	divides each element by its modulus.	qimg : quaternion image. qnimg : conjugate of qimg		2								
ximgproc_qmultiply	CVAPI(ExceptionStatus) ximgproc_qmultiply(cv::_InputArray *src1, cv::_InputArray *src2, cv::_OutputArray *dst) { BEGIN_WRAP cv::ximgproc::qmultiply(*src1, *src2, *dst); END_WRAP }	cv::ximgproc::qmultiply()	group__ximgproc__filters.html	Calculates the per-element quaternion product of two arrays.	src1 : quaternion image. src2 : quaternion image. dst : product dst(I)=src1(I) . src2(I)		3								
ximgproc_qdft	CVAPI(ExceptionStatus) ximgproc_qdft(cv::_InputArray *img, cv::_OutputArray *qimg, int flags, int sideLeft) { BEGIN_WRAP cv::ximgproc::qdft(*img, *qimg, flags, sideLeft != 0); END_WRAP }	cv::ximgproc::qdft()	group__ximgproc__filters.html	Performs a forward or inverse Discrete quaternion Fourier transform of a 2D quaternion array.	img : quaternion image. qimg : quaternion image in dual space. flags : quaternion image in dual space. only DFT_INVERSE flags is supported sideLeft : true the hypercomplex exponential is to be multiplied on the left (false on the right).		4								
ximgproc_colorMatchTemplate	CVAPI(ExceptionStatus) ximgproc_colorMatchTemplate(cv::_InputArray *img, cv::_InputArray *templ, cv::_OutputArray *result) { BEGIN_WRAP cv::ximgproc::colorMatchTemplate(*img, *templ, *result); END_WRAP }	cv::ximgproc::colorMatchTemplate()	group__ximgproc__filters.html	Compares a color template against overlapped color image regions.	img : Image where the search is running. It must be 3 channels image templ : Searched template. It must be not greater than the source image and have 3 channels result : Map of comparison results. It must be single-channel 64-bit floating-point		3								
ximgproc_GradientDericheY	CVAPI(ExceptionStatus) ximgproc_GradientDericheY(cv::_InputArray *op, cv::_OutputArray *dst, double alpha, double omega) { BEGIN_WRAP cv::ximgproc::GradientDericheY(*op, *dst, alpha, omega); END_WRAP }	cv::ximgproc::GradientDericheY()	group__ximgproc__filters.html	Applies Y Deriche filter to an image.	op : Source 8-bit or 16bit image, 1-channel or 3-channel image. dst : result CV_32FC image with same number of channel than _op. alpha : double see paper omega : double see paper	For more details about this implementation, please see http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.476.5736&rep=rep1&type=pdf	4								
ximgproc_GradientDericheX	CVAPI(ExceptionStatus) ximgproc_GradientDericheX(cv::_InputArray *op, cv::_OutputArray *dst, double alpha, double omega) { BEGIN_WRAP cv::ximgproc::GradientDericheX(*op, *dst, alpha, omega); END_WRAP }	cv::ximgproc::GradientDericheX()	group__ximgproc__filters.html	Applies X Deriche filter to an image.	op : Source 8-bit or 16bit image, 1-channel or 3-channel image. dst : result CV_32FC image with same number of channel than _op. alpha : double see paper omega : double see paper	For more details about this implementation, please see http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.476.5736&rep=rep1&type=pdf	4								
ximgproc_edgePreservingFilter	CVAPI(ExceptionStatus) ximgproc_edgePreservingFilter(cv::_InputArray *src, cv::_OutputArray *dst, int d, double threshold) { BEGIN_WRAP cv::ximgproc::edgePreservingFilter(*src, *dst, d, threshold); END_WRAP }	cv::ximgproc::edgePreservingFilter()	group__ximgproc.html	Smoothes an image using the Edge-Preserving filter.	src : Source 8-bit 3-channel image. dst : Destination image of the same size and type as src. d : Diameter of each pixel neighborhood that is used during filtering. Must be greater or equal 3. threshold : Threshold, which distinguishes between noise, outliers, and data.	The function smoothes Gaussian noise as well as salt & pepper noise. For more details about this implementation, please see [RaiWoe18] Reich, S. and Wörgötter, F. and Dellen, B. (2018). A Real-Time Edge-Preserving Denoising Filter. Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp, 85-94, 4. DOI: 10.5220/0006509000850094.	4								
ximgproc_covarianceEstimation	CVAPI(ExceptionStatus) ximgproc_covarianceEstimation(cv::_InputArray *src, cv::_OutputArray *dst, int windowRows, int windowCols) { BEGIN_WRAP cv::ximgproc::covarianceEstimation(*src, *dst, windowRows, windowCols); END_WRAP }	cv::ximgproc::covarianceEstimation()	namespacecv_1_1ximgproc.html	Computes the estimated covariance matrix of an image using the sliding window forumlation.	src : The source image. Input image must be of a complex type. dst : The destination estimated covariance matrix. Output matrix will be size (windowRows*windowCols, windowRows*windowCols). windowRows : The number of rows in the window. windowCols : The number of cols in the window. The window size parameters control the accuracy of the estimation. The sliding window moves over the entire image from the top-left corner to the bottom right corner. Each location of the window represents a sample. If the window is the size of the image, then this gives the exact covariance matrix. For all other cases, the sizes of the window will impact the number of samples and the number of elements in the estimated covariance matrix.		4								
ximgproc_FastHoughTransform	CVAPI(ExceptionStatus) ximgproc_FastHoughTransform(cv::_InputArray* src, cv::_OutputArray* dst, int dstMatDepth, int angleRange, int op, int makeSkew) { BEGIN_WRAP cv::ximgproc::FastHoughTransform(*src, *dst, dstMatDepth, angleRange, op, makeSkew); END_WRAP }	cv::ximgproc::FastHoughTransform()	namespacecv_1_1ximgproc.html	Calculates 2D Fast Hough transform of an image.	dst : The destination image, result of transformation. src : The source (input) image. dstMatDepth : The depth of destination image op : The operation to be applied, see cv::HoughOp angleRange : The part of Hough space to calculate, see cv::AngleRangeOption makeSkew : Specifies to do or not to do image skewing, see cv::HoughDeskewOption	The function calculates the fast Hough transform for full, half or quarter range of angles.		6								

ximgproc_HoughPoint2Line	CVAPI(ExceptionStatus) ximgproc_HoughPoint2Line(MyCvPoint houghPoint, cv::_InputArray* srcImgInfo, int angleRange, int makeSkew, int rules, CvVec4i* returnValue) { BEGIN_WRAP *returnValue = c(cv::ximgproc::HoughPoint2Line(cpp(houghPoint), *srcImgInfo, angleRange, makeSkew, rules)); END_WRAP }	cv::ximgproc::HoughPoint2Line()	namespacecv_1_1ximgproc.html	Calculates coordinates of line segment corresponded by point in Hough space.	houghPoint : Point in Hough space. srcImgInfo : The source (input) image of Hough transform. angleRange : The part of Hough space where point is situated, see cv::AngleRangeOption makeSkew : Specifies to do or not to do image skewing, see cv::HoughDeskewOption rules : Specifies strictness of line segment calculating, see cv::RulesOption	Return values [Vec4i]Coordinates of line segment corresponded by point in Hough space.Remarksif rules parameter set to RO_STRICT then returned line cut along the border of source image. If rules parameter set to RO_WEAK then in case of point, which belongs the incorrect part of Hough image, returned line will not intersect source image.The function calculates coordinates of line segment corresponded by point in Hough space.	5								
ximgproc_GradientPaillouY	CVAPI(ExceptionStatus) ximgproc_GradientPaillouY(cv::_InputArray* op, cv::_OutputArray* dst, double alpha, double omega) { BEGIN_WRAP cv::ximgproc::GradientPaillouY(*op, *dst, alpha, omega); END_WRAP }	cv::ximgproc::GradientPaillouY()	group__ximgproc__filters.html	Applies Paillou filter to an image.	op : Source CV_8U(S) or CV_16U(S), 1-channel or 3-channels image. _dst : result CV_32F image with same number of channel than op. omega : double see paper alpha : double see paper	For more details about this implementation, please see [189]See alsoGradientPaillouX, GradientPaillouY	4								
ximgproc_GradientPaillouX	CVAPI(ExceptionStatus) ximgproc_GradientPaillouX(cv::_InputArray* op, cv::_OutputArray* dst, double alpha, double omega) { BEGIN_WRAP cv::ximgproc::GradientPaillouX(*op, *dst, alpha, omega); END_WRAP }	cv::ximgproc::GradientPaillouX()	group__ximgproc__filters.html				4								
ximgproc_PeiLinNormalization_Mat23d	CVAPI(ExceptionStatus) ximgproc_PeiLinNormalization_Mat23d(cv::_InputArray *I, double *returnValue) { BEGIN_WRAP auto ret = cv::ximgproc::PeiLinNormalization(*I); for (int r = 0; r < 2; r++) { for (int c = 0; c < 3; ++c) { returnValue[r * 3 + c] = ret(r, c); } } END_WRAP }	cv::ximgproc::PeiLinNormalization()	group__ximgproc.html	Calculates an affine transformation that normalize given image using Pei&Lin Normalization.	I : Given transformed image.	Assume given image ¥(I=T(¥bar{I}¥) where ¥(¥bar{I}¥) is a normalized image and ¥(T¥) is an affine transformation distorting this image by translation, rotation, scaling and skew. The function returns an affine transformation matrix corresponding to the transformation ¥(T^{-1}¥) described in [PeiLin95]. For more details about this implementation, please see [PeiLin95] Soo-Chang Pei and Chao-Nan Lin. Image normalization for pattern recognition. Image and Vision Computing, Vol. 13, N.10, pp. 711-723, 1995.	2								
ximgproc_PeiLinNormalization_OutputArray	CVAPI(ExceptionStatus) ximgproc_PeiLinNormalization_OutputArray(cv::_InputArray *I, cv::_OutputArray *T) { BEGIN_WRAP cv::ximgproc::PeiLinNormalization(*I, *T); END_WRAP }	cv::ximgproc::PeiLinNormalization()	group__ximgproc.html	Calculates an affine transformation that normalize given image using Pei&Lin Normalization.	I : Given transformed image.	Assume given image ¥(I=T(¥bar{I}¥) where ¥(¥bar{I}¥) is a normalized image and ¥(T¥) is an affine transformation distorting this image by translation, rotation, scaling and skew. The function returns an affine transformation matrix corresponding to the transformation ¥(T^{-1}¥) described in [PeiLin95]. For more details about this implementation, please see [PeiLin95] Soo-Chang Pei and Chao-Nan Lin. Image normalization for pattern recognition. Image and Vision Computing, Vol. 13, N.10, pp. 711-723, 1995.	2								
ximgproc_rl_threshold	CVAPI(ExceptionStatus) ximgproc_rl_threshold(cv::_InputArray *src, cv::_OutputArray *rlDest, double thresh, int type) { BEGIN_WRAP cv::ximgproc::rl::threshold(*src, *rlDest, thresh, type); END_WRAP }	cv::ximgproc::rl::threshold()	group__ximgproc__run__length__morphology.html	Applies a fixed-level threshold to each array element.	src : input array (single-channel). rlDest : resulting run length encoded image. thresh : threshold value. type : thresholding type (only cv::THRESH_BINARY and cv::THRESH_BINARY_INV are supported)		4								
ximgproc_rl_dilate	CVAPI(ExceptionStatus) ximgproc_rl_dilate(cv::_InputArray *rlSrc, cv::_OutputArray *rlDest, cv::_InputArray *rlKernel, MyCvPoint anchor) { BEGIN_WRAP cv::ximgproc::rl::dilate(*rlSrc, *rlDest, *rlKernel, cpp(anchor)); END_WRAP }	cv::ximgproc::rl::dilate()	group__ximgproc__run__length__morphology.html	Dilates an run-length encoded binary image by using a specific structuring element.	rlSrc : input image rlDest : result rlKernel : kernel anchor : position of the anchor within the element; default value (0, 0) is usually the element center.		4								
ximgproc_rl_erode	CVAPI(ExceptionStatus) ximgproc_rl_erode(cv::_InputArray *rlSrc, cv::_OutputArray *rlDest, cv::_InputArray *rlKernel, int bBoundaryOn, MyCvPoint anchor) { BEGIN_WRAP cv::ximgproc::rl::erode(*rlSrc, *rlDest, *rlKernel, bBoundaryOn != 0, cpp(anchor)); END_WRAP }	cv::ximgproc::rl::erode()	group__ximgproc__run__length__morphology.html	Erodes an run-length encoded binary image by using a specific structuring element.	rlSrc : input image rlDest : result rlKernel : kernel bBoundaryOn : indicates whether pixel outside the image boundary are assumed to be on (True: works in the same way as the default of cv::erode, False: is a little faster) anchor : position of the anchor within the element; default value (0, 0) is usually the element center.		5								
ximgproc_rl_getStructuringElement	CVAPI(ExceptionStatus) ximgproc_rl_getStructuringElement(int shape, MyCvSize ksize, cv::Mat *outValue) { BEGIN_WRAP auto result = cv::ximgproc::rl::getStructuringElement(shape, cpp(ksize)); result.copyTo(*outValue); END_WRAP }	cv::ximgproc::rl::getStructuringElement()															
ximgproc_rl_paint	CVAPI(ExceptionStatus) ximgproc_rl_paint(cv::_InputOutputArray *image, cv::_InputArray *rlSrc, MyCvScalar value) { BEGIN_WRAP cv::ximgproc::rl::paint(*image, *rlSrc, cpp(value)); END_WRAP }	cv::ximgproc::rl::paint()	group__ximgproc__run__length__morphology.html	Paint run length encoded binary image into an image.	image : image to paint into (currently only single channel images). rlSrc : run length encoded image value : all foreground pixel of the binary image are set to this value		3								
ximgproc_rl_isRLMorphologyPossible	CVAPI(ExceptionStatus) ximgproc_rl_isRLMorphologyPossible(cv::_InputArray *rlStructuringElement, int *outValue) { BEGIN_WRAP *outValue = cv::ximgproc::rl::isRLMorphologyPossible(*rlStructuringElement) ? 1 : 0; END_WRAP }	cv::ximgproc::rl::isRLMorphologyPossible()	group__ximgproc__run__length__morphology.html	Check whether a custom made structuring element can be used with run length morphological operations. (It must consist of a continuous array of single runs per row)	rlStructuringElement : mask to be tested		1								
ximgproc_rl_createRLEImage	CVAPI(ExceptionStatus) ximgproc_rl_createRLEImage(MyCvPoint3D32i *runs, size_t runsLength, cv::_OutputArray *res, MyCvSize size) { BEGIN_WRAP std::vector<cv::Point3i> runsVec(runsLength); for (size_t i = 0; i < runsLength; i++) { runsVec[i] = cpp(runs[i]); } cv::ximgproc::rl::createRLEImage(runsVec, *res, cpp(size)); END_WRAP }	cv::ximgproc::rl::createRLEImage()	group__ximgproc__run__length__morphology.html	Creates a run-length encoded image from a vector of runs (column begin, column end, row)	runs : vector of runs res : result size : image size (to be used if an "on" boundary should be used in erosion, using the default means that the size is computed from the extension of the input)		3								
ximgproc_rl_morphologyEx	CVAPI(ExceptionStatus) ximgproc_rl_morphologyEx(cv::_InputArray *rlSrc, cv::_OutputArray *rlDest, int op, cv::_InputArray *rlKernel, int bBoundaryOnForErosion, MyCvPoint anchor) { BEGIN_WRAP cv::ximgproc::rl::morphologyEx(*rlSrc, *rlDest, op, *rlKernel, bBoundaryOnForErosion != 0, cpp(anchor)); END_WRAP }	cv::ximgproc::rl::morphologyEx()	group__ximgproc__run__length__morphology.html	Applies a morphological operation to a run-length encoded binary image.	rlSrc : input image rlDest : result op : all operations supported by cv::morphologyEx (except cv::MORPH_HITMISS) rlKernel : kernel bBoundaryOnForErosion : indicates whether pixel outside the image boundary are assumed to be on for erosion operations (True: works in the same way as the default of cv::erode, False: is a little faster) anchor : position of the anchor within the element; default value (0, 0) is usually the element center.		6								

ximgproc_weightedMedianFilter	CVAPI(ExceptionStatus) ximgproc_weightedMedianFilter(cv::_InputArray* joint, cv::_InputArray* src, cv::_OutputArray* dst, int r, double sigma, int weightType, cv::Mat* mask) { BEGIN_WRAP cv::ximgproc::weightedMedianFilter(*joint, *src, *dst, r, sigma, static_cast<cv::ximgproc::WMFWeightType>(weightType), entity(mask)); END_WRAP }	cv::ximgproc::weightedMedianFilter()	namespacecv_1_1ximgproc.html	Applies weighted median filter to an image.	joint : Joint 8-bit, 1-channel or 3-channel image. src : Source 8-bit or floating-point, 1-channel or 3-channel image. dst : Destination image. r : Radius of filtering kernel, should be a positive integer. sigma : Filter range standard deviation for the joint image. weightType : weightType The type of weight definition, see WMFWeightType mask : A 0-1 mask that has the same size with I. This mask is used to ignore the effect of some pixels. If the pixel value on mask is 0, the pixel will be ignored when maintaining the joint-histogram. This is useful for applications like optical flow occlusion handling.	For more details about this implementation, please see [29]See alsomedianBlur, jointBilateralFilter	7								
ximgproc_EdgeBoxes_getBoundingBoxes	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getBoundingBoxes(cv::ximgproc::EdgeBoxes *obj, cv::_InputArray *edge_map, cv::_InputArray *orientation_map, std::vector<cv::Rect> *boxes) { BEGIN_WRAP obj->getBoundingBoxes(*edge_map, *orientation_map, *boxes); END_WRAP }	cv::ximgproc::EdgeBoxes::getBoundingBoxes()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns array containing proposal boxes.	edge_map : edge image. orientation_map : orientation map. boxes : proposal boxes. scores : of the proposal boxes, provided a vector of float types.		3								
ximgproc_EdgeBoxes_getAlpha	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getAlpha(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getAlpha(); END_WRAP }	cv::ximgproc::EdgeBoxes::getAlpha()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the step size of sliding window search.			1								
ximgproc_EdgeBoxes_setAlpha	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setAlpha(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setAlpha(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setAlpha()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the step size of sliding window search.			1								
ximgproc_EdgeBoxes_getBeta	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getBeta(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getBeta(); END_WRAP }	cv::ximgproc::EdgeBoxes::getBeta()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the nms threshold for object proposals.			1								
ximgproc_EdgeBoxes_setBeta	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setBeta(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setBeta(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setBeta()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the nms threshold for object proposals.			1								
ximgproc_EdgeBoxes_getEta	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getEta(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getEta(); END_WRAP }	cv::ximgproc::EdgeBoxes::getEta()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns adaptation rate for nms threshold.			1								
ximgproc_EdgeBoxes_setEta	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setEta(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setEta(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setEta()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the adaptation rate for nms threshold.			1								
ximgproc_EdgeBoxes_getMinScore	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getMinScore(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getMinScore(); END_WRAP }	cv::ximgproc::EdgeBoxes::getMinScore()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the min score of boxes to detect.			1								
ximgproc_EdgeBoxes_setMinScore	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setMinScore(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setMinScore(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setMinScore()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the min score of boxes to detect.			1								
ximgproc_EdgeBoxes_getMaxBoxes	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getMaxBoxes(cv::ximgproc::EdgeBoxes *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxBoxes(); END_WRAP }	cv::ximgproc::EdgeBoxes::getMaxBoxes()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the max number of boxes to detect.			1								
ximgproc_EdgeBoxes_setMaxBoxes	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setMaxBoxes(cv::ximgproc::EdgeBoxes *obj, int value) { BEGIN_WRAP obj->setMaxBoxes(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setMaxBoxes()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets max number of boxes to detect.			1								
ximgproc_EdgeBoxes_getEdgeMinMag	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getEdgeMinMag(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getEdgeMinMag(); END_WRAP }	cv::ximgproc::EdgeBoxes::getEdgeMinMag()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the edge min magnitude.			1								
ximgproc_EdgeBoxes_setEdgeMinMag	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setEdgeMinMag(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setEdgeMinMag(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setEdgeMinMag()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the edge min magnitude.			1								
ximgproc_EdgeBoxes_getEdgeMergeThr	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getEdgeMergeThr(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getEdgeMergeThr(); END_WRAP }	cv::ximgproc::EdgeBoxes::getEdgeMergeThr()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the edge merge threshold.			1								
ximgproc_EdgeBoxes_setEdgeMergeThr	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setEdgeMergeThr(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setEdgeMergeThr(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setEdgeMergeThr()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the edge merge threshold.			1								
ximgproc_EdgeBoxes_getClusterMinMag	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getClusterMinMag(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getClusterMinMag(); END_WRAP }	cv::ximgproc::EdgeBoxes::getClusterMinMag()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the cluster min magnitude.			1								
ximgproc_EdgeBoxes_setClusterMinMag	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setClusterMinMag(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setClusterMinMag(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setClusterMinMag()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the cluster min magnitude.			1								
ximgproc_EdgeBoxes_getMaxAspectRatio	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getMaxAspectRatio(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getMaxAspectRatio(); END_WRAP }	cv::ximgproc::EdgeBoxes::getMaxAspectRatio()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the max aspect ratio of boxes.			1								
ximgproc_EdgeBoxes_setMaxAspectRatio	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setMaxAspectRatio(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setMaxAspectRatio(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setMaxAspectRatio()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the max aspect ratio of boxes.			1								
ximgproc_EdgeBoxes_getMinBoxArea	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getMinBoxArea(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getMinBoxArea(); END_WRAP }	cv::ximgproc::EdgeBoxes::getMinBoxArea()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the minimum area of boxes.			1								
ximgproc_EdgeBoxes_setMinBoxArea	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setMinBoxArea(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setMinBoxArea(value); END_WRAP }	cv::ximgproc::EdgeBoxes::setMinBoxArea()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the minimum area of boxes.			1								
ximgproc_EdgeBoxes_getGamma	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getGamma(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getGamma(); END_WRAP }	cv::ximgproc::EdgeBoxes::getGamma()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the affinity sensitivity.			1								

ximgproc_EdgeBoxes_setGamma	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setGamma(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setGamma(value); END_WRAP)	cv::ximgproc::EdgeBoxes::setGamma()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the affinity sensitivity.			1									
ximgproc_EdgeBoxes_getKappa	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_getKappa(cv::ximgproc::EdgeBoxes *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getKappa(); END_WRAP)	cv::ximgproc::EdgeBoxes::getKappa()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Returns the scale sensitivity.			1									
ximgproc_EdgeBoxes_setKappa	CVAPI(ExceptionStatus) ximgproc_EdgeBoxes_setKappa(cv::ximgproc::EdgeBoxes *obj, float value) { BEGIN_WRAP obj->setKappa(value); END_WRAP)	cv::ximgproc::EdgeBoxes::setKappa()	classcv_1_1ximgproc_1_1EdgeBoxes.html	Sets the scale sensitivity.			1									
ximgproc_createEdgeBoxes	CVAPI(ExceptionStatus) ximgproc_createEdgeBoxes(float alpha, float beta, float eta, float minScore, int maxBoxes, float edgeMinMag, float edgeMergeThr, float clusterMinMag, float maxAspectRatio, float minBoxArea, float gamma, float kappa, cv::Ptr<cv::ximgproc::EdgeBoxes> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::ximgproc::createEdgeBoxes(alpha, beta, eta, minScore, maxBoxes, edgeMinMag, edgeMergeThr, clusterMinMag, maxAspectRatio, minBoxArea, gamma, kappa)); END_WRAP }	cv::ximgproc::createEdgeBoxes()	group__ximgproc__edgeboxes.html	Creates a Edgeboxes.	alpha : step size of sliding window search. beta : nms threshold for object proposals. eta : adaptation rate for nms threshold. minScore : min score of boxes to detect. maxBoxes : max number of boxes to detect. edgeMinMag : edge min magnitude. Increase to trade off accuracy for speed. edgeMergeThr : edge merge threshold. Increase to trade off accuracy for speed. clusterMinMag : cluster min magnitude. Increase to trade off accuracy for speed. maxAspectRatio : max aspect ratio of boxes. minBoxArea : minimum area of boxes. gamma : affinity sensitivity. kappa : scale sensitivity.		12								
ximgproc_Ptr_EdgeBoxes_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_EdgeBoxes_delete(cv::Ptr<cv::ximgproc::EdgeBoxes> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::EdgeBoxes	classcv_1_1ximgproc_1_1EdgeBoxes.html	Class implementing EdgeBoxes algorithm from [297] :								•				
ximgproc_Ptr_EdgeBoxes_get	CVAPI(ExceptionStatus) ximgproc_Ptr_EdgeBoxes_get(cv::Ptr<cv::ximgproc::EdgeBoxes> *ptr, cv::ximgproc::EdgeBoxes **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::EdgeBoxes	classcv_1_1ximgproc_1_1EdgeBoxes.html	Class implementing EdgeBoxes algorithm from [297] :									•			
ximgproc_Ptr_DTFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_DTFilter_delete(cv::Ptr<cv::ximgproc::DTFilter>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::DTFilter	classcv_1_1ximgproc_1_1DTFilter.html	Interface for realizations of Domain Transform filter. For more details about this filter see [87] .								•				
ximgproc_Ptr_DTFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_DTFilter_get(cv::Ptr<cv::ximgproc::DTFilter>* ptr, cv::ximgproc::DTFilter** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::DTFilter	classcv_1_1ximgproc_1_1DTFilter.html	Interface for realizations of Domain Transform filter. For more details about this filter see [87] .									•			
ximgproc_DTFilter_filter	CVAPI(ExceptionStatus) ximgproc_DTFilter_filter(cv::ximgproc::DTFilter* obj, cv::_InputArray *src, cv::_OutputArray *dst, int dDepth) { BEGIN_WRAP obj->filter(*src, *dst, dDepth); END_WRAP }	cv::ximgproc::DTFilter::filter()	classcv_1_1ximgproc_1_1DTFilter.html	Produce domain transform filtering operation on source image.	src : filtering image with unsigned 8-bit or floating-point 32-bit depth and up to 4 channels. dst : destination image. dDepth : optional depth of the output image. dDepth can be set to -1, which will be equivalent to src.depth().		3									
ximgproc_createDTFilter	CVAPI(ExceptionStatus) ximgproc_createDTFilter(cv::_InputArray *guide, double sigmaSpatial, double sigmaColor, int mode, int numIters, cv::Ptr<cv::ximgproc::DTFilter>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createDTFilter(*guide, sigmaSpatial, sigmaColor, mode, numIters); *returnValue = new cv::Ptr<cv::ximgproc::DTFilter>(ptr); END_WRAP }	cv::ximgproc::createDTFilter()	group__ximgproc__filters.html	Factory method, create instance of DTFilter and produce initialization routines.	guide : guided image (used to build transformed distance, which describes edge structure of guided image). sigmaSpatial : ¥(¥sigma)_H¥) parameter in the original article, it's similar to the sigma in the coordinate space into bilateralFilter. sigmaColor : ¥(¥sigma)_r¥) parameter in the original article, it's similar to the sigma in the color space into bilateralFilter. mode : one form three modes DTF_NC, DTF_RF and DTF_IC which corresponds to three modes for filtering 2D signals in the article. numIters : optional number of iterations used for filtering, 3 is quite enough.	For more details about Domain Transform filter parameters, see the original article [87] and Domain Transform filter homepage.	5								
ximgproc_dtFilter	CVAPI(ExceptionStatus) ximgproc_dtFilter(cv::_InputArray *guide, cv::_InputArray *src, cv::_OutputArray *dst, double sigmaSpatial, double sigmaColor, int mode, int numIters) { BEGIN_WRAP cv::ximgproc::dtFilter(*guide, *src, *dst, sigmaSpatial, sigmaColor, mode, numIters); END_WRAP }	cv::ximgproc::dtFilter()	group__ximgproc__filters.html	Simple one-line Domain Transform filter call. If you have multiple images to filter with the same guided image then use DTFilter interface to avoid extra computations on initialization stage.	guide : guided image (also called as joint image) with unsigned 8-bit or floating-point 32-bit depth and up to 4 channels. src : filtering image with unsigned 8-bit or floating-point 32-bit depth and up to 4 channels. dst : destination image. sigmaSpatial : ¥(¥sigma)_H¥) parameter in the original article, it's similar to the sigma in the coordinate space into bilateralFilter. sigmaColor : ¥(¥sigma)_r¥) parameter in the original article, it's similar to the sigma in the color space into bilateralFilter. mode : one form three modes DTF_NC, DTF_RF and DTF_IC which corresponds to three modes for filtering 2D signals in the article. numIters : optional number of iterations used for filtering, 3 is quite enough.	See alsobilateralFilter, guidedFilter, amFilter	7								
ximgproc_Ptr_GuidedFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_GuidedFilter_delete(cv::Ptr<cv::ximgproc::GuidedFilter>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::GuidedFilter	classcv_1_1ximgproc_1_1GuidedFilter.html	Interface for realizations of Guided Filter. For more details about this filter see [106] .								•				
ximgproc_Ptr_GuidedFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_GuidedFilter_get(cv::Ptr<cv::ximgproc::GuidedFilter>* ptr, cv::ximgproc::GuidedFilter** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::GuidedFilter	classcv_1_1ximgproc_1_1GuidedFilter.html	Interface for realizations of Guided Filter. For more details about this filter see [106] .									•			
ximgproc_GuidedFilter_filter	CVAPI(ExceptionStatus) ximgproc_GuidedFilter_filter(cv::ximgproc::GuidedFilter* obj, cv::_InputArray* src, cv::_OutputArray* dst, int dDepth) { BEGIN_WRAP obj->filter(*src, *dst, dDepth); END_WRAP }	cv::ximgproc::GuidedFilter::filter()	classcv_1_1ximgproc_1_1GuidedFilter.html	Apply Guided Filter to the filtering image.	src : filtering image with any numbers of channels. dst : output image. dDepth : optional depth of the output image. dDepth can be set to -1, which will be equivalent to src.depth().		3									
ximgproc_createGuidedFilter	CVAPI(ExceptionStatus) ximgproc_createGuidedFilter(cv::_InputArray* guide, int radius, double eps, cv::Ptr<cv::ximgproc::GuidedFilter>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createGuidedFilter(*guide, radius, eps); *returnValue = new cv::Ptr<cv::ximgproc::GuidedFilter>(ptr); END_WRAP }	cv::ximgproc::createGuidedFilter()	group__ximgproc__filters.html	Factory method, create instance of GuidedFilter and produce initialization routines.	guide : guided image (or array of images) with up to 3 channels, if it have more then 3 channels then only first 3 channels will be used. radius : radius of Guided Filter. eps : regularization term of Guided Filter. ¥(eps^2¥) is similar to the sigma in the color space into bilateralFilter.	For more details about Guided Filter parameters, see the original article [106]	3								

ximgproc_guidedFilter	CVAPI(ExceptionStatus) ximgproc_guidedFilter(cv::_InputArray *guide, cv::_InputArray *src, cv::_OutputArray *dst, int radius, double eps, int dDepth) { BEGIN_WRAP cv::ximgproc::guidedFilter(*g uide, *src, *dst, radius, eps, dDepth); END_WRAP }	cv::ximgproc::guidedFilter()	group__ximgproc__filters.html	Simple one-line Guided Filter call.	guide : guided image (or array of images) with up to 3 channels, if it have more then 3 channels then only first 3 channels will be used. src : filtering image with any numbers of channels. dst : output image. radius : radius of Guided Filter. eps : regularization term of Guided Filter. ^2 is similar to the sigma in the color space into bilateralFilter. dDepth : optional depth of the output image.	If you have multiple images to filter with the same guided image then use GuidedFilter interface to avoid extra computations on initialization stage.See alsobilateralFilter, dtFilter, amFilter	6								
ximgproc_Ptr_AdaptiveManifoldFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_AdaptiveManifol dFilter_delete(cv::Ptr<cv::ximgproc::Adapti veManifoldFilter>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	Interface for Adaptive Manifold Filter realizations. For more details about this filter see [88] and References_. Below listed optional parameters which may be set up with Algorithm::set function. member double sigma_s = 16.0 Spatial standard deviation. member double sigma_r = 0.2 Color space standard deviation. member int tree_height = -1 Height of the manifold tree (default = -1 : automatically computed). member int num_pca_iterations = 1 Number of iterations to computed the eigenvector. member bool adjust_outliers = false Specify adjust outliers using Eq. 9 or not. member bool use_RNG = true Specify use random number generator to compute eigenvector or not.								•		
ximgproc_Ptr_AdaptiveManifoldFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_AdaptiveManifol dFilter_get(cv::Ptr<cv::ximgproc::Adapti veManifoldFilter>* ptr, cv::ximgproc::AdaptiveManifo ldFilter** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	Interface for Adaptive Manifold Filter realizations. For more details about this filter see [88] and References_. Below listed optional parameters which may be set up with Algorithm::set function. member double sigma_s = 16.0 Spatial standard deviation. member double sigma_r = 0.2 Color space standard deviation. member int tree_height = -1 Height of the manifold tree (default = -1 : automatically computed). member int num_pca_iterations = 1 Number of iterations to computed the eigenvector. member bool adjust_outliers = false Specify adjust outliers using Eq. 9 or not. member bool use_RNG = true Specify use random number generator to compute eigenvector or not.									•	
ximgproc_AdaptiveManifoldFilter_filter	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_filter(cv::ximgproc::AdaptiveManifo ldFilter* obj, cv::_InputArray* src, cv::_OutputArray* dst, cv::_InputArray *joint) { BEGIN_WRAP obj->filter(*src, *dst, ptr0(joint)); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::filter()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	Apply high-dimensional filtering using adaptive manifolds.	src : filtering image with any numbers of channels. dst : output image. joint : optional joint (also called as guided) image with any numbers of channels.		3								
ximgproc_AdaptiveManifoldFilter_collectGarbag	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_collectGarbage(cv::ximgproc::AdaptiveManifo ldFilter* obj) { BEGIN_WRAP obj->collectGarbage(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::collectGarbage()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html				1								
ximgproc_AdaptiveManifoldFilter_getSigmaS	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_getSigmaS(cv::ximgproc:: AdaptiveManifoldFilter* obj, double *returnValue) { BEGIN_WRAP *returnValue = obj- >getSigmaS(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::getSigmaS()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsosetSigmaS			1								
ximgproc_AdaptiveManifoldFilter_setSigmaS	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_setSigmaS(cv::ximgproc:: AdaptiveManifoldFilter* obj, double val) { BEGIN_WRAP obj->setSigmaS(val); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setSigmaS()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetSigmaS			1								
ximgproc_AdaptiveManifoldFilter_getSigmaR	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_getSigmaR(cv::ximgproc:: AdaptiveManifoldFilter* obj, double* returnValue) { BEGIN_WRAP *returnValue = obj- >getSigmaR(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFiltergetSigmaR()													
ximgproc_AdaptiveManifoldFilter_setSigmaR	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_setSigmaR(cv::ximgproc:: AdaptiveManifoldFilter* obj, double val) { BEGIN_WRAP obj->setSigmaR(val); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setSigmaR()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetSigmaR			1								
ximgproc_AdaptiveManifoldFilter_getTreeHeight	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_getTreeHeight(cv::ximgpro c::AdaptiveManifoldFilter* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj- >getTreeHeight(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::getTreeHeight()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsosetTreeHeight			1								
ximgproc_AdaptiveManifoldFilter_setTreeHeight	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_setTreeHeight(cv::ximgpro c::AdaptiveManifoldFilter* obj, int val) { BEGIN_WRAP obj->setTreeHeight(val); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setTreeHeight()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetTreeHeight			1								
ximgproc_AdaptiveManifoldFilter_getPCAIterati	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_getPCAIterations(cv::ximg proc::AdaptiveManifoldFilter* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj- >getPCAIterations(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::getPCAIterations()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsosetPCAIterations			1								
ximgproc_AdaptiveManifoldFilter_setPCAIterati	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_setPCAIterations(cv::ximg proc::AdaptiveManifoldFilter* obj, int val) { BEGIN_WRAP obj- >setPCAIterations(val); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setPCAIterations()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetPCAIterations			1								
ximgproc_AdaptiveManifoldFilter_getAdjustOut	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_getAdjustOutliers(cv::ximg proc::AdaptiveManifoldFilter* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj- >getAdjustOutliers(); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::getAdjustOutliers()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsosetAdjustOutliers			1								
ximgproc_AdaptiveManifoldFilter_setAdjustOut	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilt er_setAdjustOutliers(cv::ximg proc::AdaptiveManifoldFilter* obj, int val) { BEGIN_WRAP obj- >setAdjustOutliers(val); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setAdjustOutliers()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetAdjustOutliers			1								

ximgproc_AdaptiveManifoldFilter_getUseRNG	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilter_getUseRNG(cv::ximgproc::AdaptiveManifoldFilter* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->getUseRNG() ? 1 : 0; END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::getUseRNG()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsosetUseRNG			1									
ximgproc_AdaptiveManifoldFilter_setUseRNG	CVAPI(ExceptionStatus) ximgproc_AdaptiveManifoldFilter_setUseRNG(cv::ximgproc::AdaptiveManifoldFilter* obj, int val) { BEGIN_WRAP obj->setUseRNG(val != 0); END_WRAP }	cv::ximgproc::AdaptiveManifoldFilter::setUseRNG()	classcv_1_1ximgproc_1_1AdaptiveManifoldFilter.html	See alsogetUseRNG			1									
ximgproc_createAMFilter	CVAPI(ExceptionStatus) ximgproc_createAMFilter(double sigma_s, double sigma_r, int adjust_outliers, cv::Ptr<cv::ximgproc::AdaptiveManifoldFilter>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createAMFilter(sigma_s, sigma_r, adjust_outliers != 0); *returnValue = new cv::Ptr<cv::ximgproc::AdaptiveManifoldFilter>(ptr); END_WRAP }	cv::ximgproc::createAMFilter()	group__ximgproc__filters.html	Factory method, create instance of AdaptiveManifoldFilter and produce some initialization routines.	sigma_s : spatial standard deviation. sigma_r : color space standard deviation, it is similar to the sigma in the color space into bilateralFilter. adjust_outliers : optional, specify perform outliers adjust operation or not, (Eq. 9) in the original paper.	For more details about Adaptive Manifold Filter parameters, see the original article [88] .NoteJoint images with CV_8U and CV_16U depth converted to images with CV_32F depth and [0; 1] color range before processing. Hence color space sigma sigma_r must be in [0; 1] range, unlike same sigmas in bilateralFilter and dtFilter functions.	3								
ximgproc_amFilter	CVAPI(ExceptionStatus) ximgproc_amFilter(cv::_InputArray *joint, cv::_InputArray *src, cv::_OutputArray *dst, double sigma_s, double sigma_r, int adjust_outliers) { BEGIN_WRAP cv::ximgproc::amFilter(*joint, *src, *dst, sigma_s, sigma_r, adjust_outliers != 0); END_WRAP }	cv::ximgproc::amFilter()	group__ximgproc__filters.html	Simple one-line Adaptive Manifold Filter call.	joint : joint (also called as guided) image or array of images with any numbers of channels. src : filtering image with any numbers of channels. dst : output image. sigma_s : spatial standard deviation. sigma_r : color space standard deviation, it is similar to the sigma in the color space into bilateralFilter. adjust_outliers : optional, specify perform outliers adjust operation or not, (Eq. 9) in the original paper.	NoteJoint images with CV_8U and CV_16U depth converted to images with CV_32F depth and [0; 1] color range before processing. Hence color space sigma sigma_r must be in [0; 1] range, unlike same sigmas in bilateralFilter and dtFilter functions.See alsobilateralFilter, dtFilter, guidedFilter	6								
ximgproc_jointBilateralFilter	CVAPI(ExceptionStatus) ximgproc_jointBilateralFilter(cv::_InputArray *joint, cv::_InputArray *src, cv::_OutputArray *dst, int d, double sigmaColor, double sigmaSpace, int borderType) { BEGIN_WRAP cv::ximgproc::jointBilateralFilter(*joint, *src, *dst, d, sigmaColor, sigmaSpace, borderType); END_WRAP }	cv::ximgproc::jointBilateralFilter()	group__ximgproc__filters.html	Applies the joint bilateral filter to an image.	joint : Joint 8-bit or floating-point, 1-channel or 3-channel image. src : Source 8-bit or floating-point, 1-channel or 3-channel image with the same depth as joint image. dst : Destination image of the same size and type as src . d : Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace . sigmaColor : Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color. sigmaSpace : Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0 , it specifies the neighborhood size regardless of sigmaSpace . Otherwise, d is proportional to sigmaSpace .	NotebilateralFilter and jointBilateralFilter use L1 norm to compute difference between colors.See alsobilateralFilter, amFilter	7								
ximgproc_bilateralTextureFilter	CVAPI(ExceptionStatus) ximgproc_bilateralTextureFilter(cv::_InputArray *src, cv::_OutputArray *dst, int fr, int numIter, double sigmaAlpha, double sigmaAvg) { BEGIN_WRAP cv::ximgproc::bilateralTextureFilter(*src, *dst, fr, numIter, sigmaAlpha, sigmaAvg); END_WRAP }	cv::ximgproc::bilateralTextureFilter()	group__ximgproc__filters.html	Applies the bilateral texture filter to an image. It performs structure-preserving texture filter. For more details about this filter see [47].	src : Source image whose depth is 8-bit UINT or 32-bit FLOAT dst : Destination image of the same size and type as src . fr : Radius of kernel to be used for filtering. It should be positive integer numIter : Number of iterations of algorithm, It should be positive integer sigmaAlpha : Controls the sharpness of the weight transition from edges to smooth/texture regions, where a bigger value means sharper transition. When the value is negative, it is automatically calculated. sigmaAvg : Range blur parameter for texture blurring. Larger value makes result to be more blurred. When the value is negative, it is automatically calculated as described in the paper.	See alsorollingGuidanceFilter, bilateralFilter	6								
ximgproc_rollingGuidanceFilter	CVAPI(ExceptionStatus) ximgproc_rollingGuidanceFilter(cv::_InputArray *src, cv::_OutputArray *dst, int d, double sigmaColor, double sigmaSpace, int numOfIter, int borderType) { BEGIN_WRAP cv::ximgproc::rollingGuidanceFilter(*src, *dst, d, sigmaColor, sigmaSpace, numOfIter, borderType); END_WRAP }	cv::ximgproc::rollingGuidanceFilter()	group__ximgproc__filters.html	Applies the rolling guidance filter to an image.	src : Source 8-bit or floating-point, 1-channel or 3-channel image. dst : Destination image of the same size and type as src . d : Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace . sigmaColor : Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color. sigmaSpace : Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0 , it specifies the neighborhood size regardless of sigmaSpace . Otherwise, d is proportional to sigmaSpace . numOfIter : Number of iterations of joint edge-preserving filtering applied on the source image.	For more details, please see [29].NoterollingGuidanceFilter uses jointBilateralFilter as the edge-preserving filter.See alsojointBilateralFilter, bilateralFilter, amFilter	7								
ximgproc_Ptr_FastBilateralSolverFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_FastBilateralSolverFilter_delete(cv::Ptr<cv::ximgproc::FastBilateralSolverFilter>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::FastBilateralSolverFilter	classcv_1_1ximgproc_1_1FastBilateralSolverFilter.html	Interface for implementations of Fast Bilateral Solver. For more details about this solver see [15] .								•		
ximgproc_Ptr_FastBilateralSolverFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_FastBilateralSolverFilter_get(cv::Ptr<cv::ximgproc::FastBilateralSolverFilter>* ptr, cv::ximgproc::FastBilateralSolverFilter** returnValue) { BEGIN_WRAP * returnValue = ptr->get(); END_WRAP }	cv::ximgproc::FastBilateralSolverFilter	classcv_1_1ximgproc_1_1FastBilateralSolverFilter.html	Interface for implementations of Fast Bilateral Solver. For more details about this solver see [15] .									•	
ximgproc_FastBilateralSolverFilter_filter	CVAPI(ExceptionStatus) ximgproc_FastBilateralSolverFilter_filter(cv::ximgproc::FastBilateralSolverFilter* obj, cv::_InputArray* src, cv::_InputArray *confidence, cv::_OutputArray* dst) { BEGIN_WRAP obj->filter(*src, *confidence , *dst); END_WRAP }	cv::ximgproc::FastBilateralSolverFilter::filter()	classcv_1_1ximgproc_1_1FastBilateralSolverFilter.html	Apply smoothing operation to the source image.	src : source image for filtering with unsigned 8-bit or signed 16-bit or floating-point 32-bit depth and up to 3 channels. confidence : confidence image with unsigned 8-bit or floating-point 32-bit confidence and 1 channel. dst : destination image.	NoteConfidence images with CV_8U depth are expected to in [0, 255] and CV_32F in [0, 1] range.	3								
ximgproc_createFastBilateralSolverFilter	CVAPI(ExceptionStatus) ximgproc_createFastBilateralSolverFilter(cv::_InputArray *guide, double sigma_spatial, double sigma_luma, double sigma_chroma, double lambda, int num_iter, double max_tol, cv::Ptr<cv::ximgproc::FastBilateralSolverFilter>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createFastBilateralSolverFilter(*guide, sigma_spatial, sigma_luma, sigma_chroma, lambda, num_iter, max_tol); *returnValue = new cv::Ptr<cv::ximgproc::FastBilateralSolverFilter>(ptr); END_WRAP }	cv::ximgproc::createFastBilateralSolverFilter()	group__ximgproc__filters.html	Factory method, create instance of FastBilateralSolverFilter and execute the initialization routines.	guide : image serving as guide for filtering. It should have 8-bit depth and either 1 or 3 channels. sigma_spatial : parameter, that is similar to spatial space sigma (bandwidth) in bilateralFilter. sigma_luma : parameter, that is similar to luma space sigma (bandwidth) in bilateralFilter. sigma_chroma : parameter, that is similar to chroma space sigma (bandwidth) in bilateralFilter. lambda : smoothness strength parameter for solver. num_iter : number of iterations used for solver, 25 is usually enough. max_tol : convergence tolerance used for solver.	For more details about the Fast Bilateral Solver parameters, see the original paper [15].	7								
ximgproc_fastBilateralSolverFilter	CVAPI(ExceptionStatus) ximgproc_fastBilateralSolverFilter(cv::_InputArray *guide, cv::_InputArray *src, cv::_InputArray *confidence, cv::_OutputArray *dst, double sigma_spatial, double sigma_luma, double sigma_chroma, double lambda, int num_iter, double max_tol) { BEGIN_WRAP cv::ximgproc::fastBilateralSolverFilter(*guide, *src, *confidence, *dst, sigma_spatial, sigma_luma, sigma_chroma, lambda, num_iter, max_tol); END_WRAP }	cv::ximgproc::fastBilateralSolverFilter()	group__ximgproc__filters.html	Simple one-line Fast Bilateral Solver filter call. If you have multiple images to filter with the same guide then use FastBilateralSolverFilter interface to avoid extra computations.	guide : image serving as guide for filtering. It should have 8-bit depth and either 1 or 3 channels. src : source image for filtering with unsigned 8-bit or signed 16-bit or floating-point 32-bit depth and up to 4 channels. confidence : confidence image with unsigned 8-bit or floating-point 32-bit confidence and 1 channel. dst : destination image. sigma_spatial : parameter, that is similar to spatial space sigma (bandwidth) in bilateralFilter. sigma_luma : parameter, that is similar to luma space sigma (bandwidth) in bilateralFilter. sigma_chroma : parameter, that is similar to chroma space sigma (bandwidth) in bilateralFilter. lambda : smoothness strength parameter for solver. num_iter : number of iterations used for solver, 25 is usually enough. max_tol : convergence tolerance used for solver.	For more details about the Fast Bilateral Solver parameters, see the original paper [15].NoteConfidence images with CV_8U depth are expected to in [0, 255] and CV_32F in [0, 1] range.	10								

ximgproc_Ptr_FastGlobalSmootherFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_FastGlobalSmootherFilter_delete(cv::Ptr<cv::ximgproc::FastGlobalSmootherFilter>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::FastGlobalSmootherFilter	classcv_1_1ximgproc_1_1FastGlobalSmootherFilter.html	Interface for implementations of Fast Global Smoother filter. For more details about this filter see [172] and [68] .								•			
ximgproc_Ptr_FastGlobalSmootherFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_FastGlobalSmootherFilter_get(cv::Ptr<cv::ximgproc::FastGlobalSmootherFilter>* ptr, cv::ximgproc::FastGlobalSmootherFilter** returnValue) { BEGIN_WRAP * returnValue = ptr->get(); END_WRAP }	cv::ximgproc::FastGlobalSmootherFilter	classcv_1_1ximgproc_1_1FastGlobalSmootherFilter.html	Interface for implementations of Fast Global Smoother filter. For more details about this filter see [172] and [68] .								•			
ximgproc_FastGlobalSmootherFilter_filter	CVAPI(ExceptionStatus) ximgproc_FastGlobalSmootherFilter_filter(cv::ximgproc::FastGlobalSmootherFilter* obj, cv::_InputArray* src, cv::_OutputArray* dst) { BEGIN_WRAP obj->filter(*src, *dst); END_WRAP }	cv::ximgproc::FastGlobalSmootherFilter::filter()	classcv_1_1ximgproc_1_1FastGlobalSmootherFilter.html	Apply smoothing operation to the source image.	src : source image for filtering with unsigned 8-bit or signed 16-bit or floating-point 32-bit depth and up to 4 channels. dst : destination image.		2									
ximgproc_createFastGlobalSmootherFilter	CVAPI(ExceptionStatus) ximgproc_createFastGlobalSmootherFilter(cv::_InputArray *guide, double lambda, double sigma_color, double lambda_attenuation, int num_iter, cv::Ptr<cv::ximgproc::FastGlobalSmootherFilter>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createFastGlobalSmootherFilter(*guide, lambda, sigma_color, lambda_attenuation, num_iter); *returnValue = new cv::Ptr<cv::ximgproc::FastGlobalSmootherFilter>(ptr); END_WRAP }	cv::ximgproc::createFastGlobalSmootherFilter()	group__ximgproc__filters.html	Factory method, create instance of FastGlobalSmootherFilter and execute the initialization routines.	guide : image serving as guide for filtering. It should have 8-bit depth and either 1 or 3 channels. lambda : parameter defining the amount of regularization sigma_color : parameter, that is similar to color space sigma in bilateralFilter. lambda_attenuation : internal parameter, defining how much lambda decreases after each iteration. Normally, it should be 0.25. Setting it to 1.0 may lead to streaking artifacts. num_iter : number of iterations used for filtering, 3 is usually enough.	For more details about Fast Global Smoother parameters, see the original paper [172]. However, please note that there are several differences. Lambda attenuation described in the paper is implemented a bit differently so do not expect the results to be identical to those from the paper; sigma_color values from the paper should be multiplied by 255.0 to achieve the same effect. Also, in case of image filtering where source and guide image are the same, authors propose to dynamically update the guide image after each iteration. To maximize the performance this feature was not implemented here.	5								
ximgproc_fastGlobalSmootherFilter	CVAPI(ExceptionStatus) ximgproc_fastGlobalSmootherFilter(cv::_InputArray *guide, cv::_InputArray *src, cv::_OutputArray *dst, double lambda, double sigma_color, double lambda_attenuation, int num_iter) { BEGIN_WRAP cv::ximgproc::fastGlobalSmootherFilter(*guide, *src, *dst, lambda, sigma_color, lambda_attenuation, num_iter); END_WRAP }	cv::ximgproc::fastGlobalSmootherFilter()	group__ximgproc__filters.html	Simple one-line Fast Global Smoother filter call. If you have multiple images to filter with the same guide then use FastGlobalSmootherFilter interface to avoid extra computations.	guide : image serving as guide for filtering. It should have 8-bit depth and either 1 or 3 channels. src : source image for filtering with unsigned 8-bit or signed 16-bit or floating-point 32-bit depth and up to 4 channels. dst : destination image. lambda : parameter defining the amount of regularization sigma_color : parameter, that is similar to color space sigma in bilateralFilter. lambda_attenuation : internal parameter, defining how much lambda decreases after each iteration. Normally, it should be 0.25. Setting it to 1.0 may lead to streaking artifacts. num_iter : number of iterations used for filtering, 3 is usually enough.		7								
ximgproc_l0Smooth	CVAPI(ExceptionStatus) ximgproc_l0Smooth(cv::_InputArray *src, cv::_OutputArray *dst, double lambda, double kappa) { BEGIN_WRAP cv::ximgproc::l0Smooth(*src, *dst, lambda, kappa); END_WRAP }	cv::ximgproc::l0Smooth()	group__ximgproc__filters.html	Global image smoothing via L0 gradient minimization.	src : source image for filtering with unsigned 8-bit or signed 16-bit or floating-point depth. dst : destination image. lambda : parameter defining the smooth term weight. kappa : parameter defining the increasing factor of the weight of the gradient data term.	For more details about L0 Smoother, see the original paper [280].	4								
ximgproc_Ptr_FastLineDetector_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_FastLineDetector_delete(cv::Ptr<cv::ximgproc::FastLineDetector>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::FastLineDetector	classcv_1_1ximgproc_1_1FastLineDetector.html	Class implementing the FLD (Fast Line Detector) algorithm described in [136] . #include <iostream>#include "opencv2/imgproc.hpp"#include "opencv2/ximgproc.hpp"#include "opencv2/imgcodecs.hpp"#include "opencv2/highgui.hpp"using namespace std;using namespace cv;using namespace cv::ximgproc;int main(int argc, char** argv){ string in; CommandLineParser parser(argc, argv, "{@input|corridor.jpg|input image}{help h||show help message}"); if (parser.has("help")) { parser.printMessage(); return 0; } in = samples::findFile(parser.get<string>("@input")); Mat image = imread(in, IMREAD_GRAYSCALE); if(image.empty()) { return -1; } // Create FLD detector // Param Default value Description // length_threshold 10 - Segments shorter than this will be discarded // distance_threshold 1.41421356 - A point placed from a hypothesis line // segment farther than this will be // regarded as an outlier // canny_th1 50 - First threshold for // hysteresis procedure in Canny() // canny_th2 50 - Secnod threshold for //								•			
ximgproc_Ptr_FastLineDetector_get	CVAPI(ExceptionStatus) ximgproc_Ptr_FastLineDetector_get(cv::Ptr<cv::ximgproc::FastLineDetector>* ptr, cv::ximgproc::FastLineDetector **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::FastLineDetector	classcv_1_1ximgproc_1_1FastLineDetector.html	Class implementing the FLD (Fast Line Detector) algorithm described in [136] . #include <iostream>#include "opencv2/imgproc.hpp"#include "opencv2/ximgproc.hpp"#include "opencv2/imgcodecs.hpp"#include "opencv2/highgui.hpp"using namespace std;using namespace cv;using namespace cv::ximgproc;int main(int argc, char** argv){ string in; CommandLineParser parser(argc, argv, "{@input|corridor.jpg|input image}{help h||show help message}"); if (parser.has("help")) { parser.printMessage(); return 0; } in = samples::findFile(parser.get<string>("@input")); Mat image = imread(in, IMREAD_GRAYSCALE); if(image.empty()) { return -1; } // Create FLD detector // Param Default value Description // length_threshold 10 - Segments shorter than this will be discarded // distance_threshold 1.41421356 - A point placed from a hypothesis line // segment farther than this will be // regarded as an outlier // canny_th1 50 - First threshold for // hysteresis procedure in Canny() // canny_th2 50 - Secnod threshold for //								•			
ximgproc_FastLineDetector_detect_OutputArray	CVAPI(ExceptionStatus) ximgproc_FastLineDetector_detect_OutputArray(cv::ximgproc::FastLineDetector *obj, cv::_InputArray *image, cv::_OutputArray *lines) { BEGIN_WRAP obj->detect(*image, *lines); END_WRAP }	cv::ximgproc::FastLineDetector::detect()	classcv_1_1ximgproc_1_1FastLineDetector.html	Finds lines in the input image. This is the output of the default parameters of the algorithm on the above shown image.	image : A grayscale (CV_8UC1) input image. If only a roi needs to be selected, use: fld_ptr->detect(image(roi), lines, ...); lines += Scalar(roi.x, roi.y, roi.x, roi.y); lines : A vector of Vec4f elements specifying the beginning and ending point of a line. Where Vec4f is (x1, y1, x2, y2), point 1 is the start, point 2 - end. Returned lines are directed so that the brighter side is on their left.	image	2								
ximgproc_FastLineDetector_detect_vector	CVAPI(ExceptionStatus) ximgproc_FastLineDetector_detect_vector(cv::ximgproc::FastLineDetector *obj, cv::_InputArray *image, std::vector<cv::Vec4f> *lines) { BEGIN_WRAP obj->detect(*image, *lines); END_WRAP }	cv::ximgproc::FastLineDetector::detect()	classcv_1_1ximgproc_1_1FastLineDetector.html	Finds lines in the input image. This is the output of the default parameters of the algorithm on the above shown image.	image : A grayscale (CV_8UC1) input image. If only a roi needs to be selected, use: fld_ptr->detect(image(roi), lines, ...); lines += Scalar(roi.x, roi.y, roi.x, roi.y); lines : A vector of Vec4f elements specifying the beginning and ending point of a line. Where Vec4f is (x1, y1, x2, y2), point 1 is the start, point 2 - end. Returned lines are directed so that the brighter side is on their left.	image	2								
ximgproc_FastLineDetector_drawSegments_InputArray	CVAPI(ExceptionStatus) ximgproc_FastLineDetector_drawSegments_InputArray(cv::ximgproc::FastLineDetector *obj, cv::_InputOutputArray *image, cv::_InputArray *lines, int draw_arrow) { BEGIN_WRAP obj->drawSegments(*image, *lines, draw_arrow != 0); END_WRAP }	cv::ximgproc::FastLineDetector::drawSegments()	classcv_1_1ximgproc_1_1FastLineDetector.html	Draws the line segments on a given image.	image : The image, where the lines will be drawn. Should be bigger or equal to the image, where the lines were found. lines : A vector of the lines that needed to be drawn. draw_arrow : If true, arrow heads will be drawn. linecolor : Line color. linethickness : Line thickness.		3								

ximgproc_FastLineDetector_drawSegments_vec	CVAPI(ExceptionStatus) ximgproc_FastLineDetector_drawSegments_vector(cv::ximgproc::FastLineDetector *obj, cv::_InputOutputArray *image, std::vector<cv::Vec4f> *lines, int draw_arrow) { BEGIN_WRAP obj->drawSegments(*image, *lines, draw_arrow != 0); END_WRAP }	cv::ximgproc::FastLineDetector::drawSegments()	classcv_1_1ximgproc_1_1FastLineDetector.html	Draws the line segments on a given image.	image : The image, where the lines will be drawn. Should be bigger or equal to the image, where the lines were found. lines : A vector of the lines that needed to be drawn. draw_arrow : If true, arrow heads will be drawn. linecolor : Line color. linethickness : Line thickness.		3								
ximgproc_createFastLineDetector	CVAPI(ExceptionStatus) ximgproc_createFastLineDetector(int length_threshold, float distance_threshold, double canny_th1, double canny_th2, int canny_aperture_size, int do_merge, cv::Ptr<cv::ximgproc::FastLineDetector> **returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createFastLineDetector(length_threshold, distance_threshold, canny_th1, canny_th2, canny_aperture_size, do_merge != 0); *returnValue = new cv::Ptr<cv::ximgproc::FastLineDetector>(ptr); END_WRAP }	cv::ximgproc::createFastLineDetector()	group__ximgproc__fast__line__detector.html	Creates a smart pointer to a FastLineDetector object and initializes it.	length_threshold : Segment shorter than this will be discarded distance_threshold : A point placed from a hypothesis line segment farther than this will be regarded as an outlier canny_th1 : First threshold for hysteresis procedure in Canny() canny_th2 : Second threshold for hysteresis procedure in Canny() canny_aperture_size : Aperturesize for the sobel operator in Canny(). If zero, Canny() is not applied and the input image is taken as an edge image. do_merge : If true, incremental merging of segments will be performed	Examples: fld_lines.cpp.	6								
ximgproc_RidgeDetectionFilter_create	CVAPI(ExceptionStatus) ximgproc_RidgeDetectionFilter_create(int ddepth, int dx, int dy, int ksize, int out_dtype, double scale, double delta, int borderType, cv::Ptr<cv::ximgproc::RidgeDetectionFilter> **returnValue) { BEGIN_WRAP auto obj = cv::ximgproc::RidgeDetectionFilter::create(ddepth, dx, dy, ksize, out_dtype, scale, delta, borderType); *returnValue = clone(obj); END_WRAP }	cv::ximgproc::RidgeDetectionFilter::create()	classcv_1_1ximgproc_1_1RidgeDetectionFilter.html	Create pointer to the Ridge detection filter.	ddepth : Specifies output image depth. Default is CV_32FC1 dx : Order of derivative x, default is 1 dy : Order of derivative y, default is 1 ksize : Sobel kernel size , default is 3 out_dtype : Converted format for output, default is CV_8UC1 scale : Optional scale value for derivative values, default is 1 delta : Optional bias added to output, default is 0 borderType : Pixel extrapolation method, default is BORDER_DEFAULT	See alsoSobel, threshold, getStructuringElement, morphologyEx.(for additional refinement)	8								
ximgproc_RidgeDetectionFilter_getRidgeFilteredImage	CVAPI(ExceptionStatus) ximgproc_RidgeDetectionFilter_getRidgeFilteredImage(cv::ximgproc::RidgeDetectionFilter *obj, cv::_InputArray *_img, cv::_OutputArray *out) { BEGIN_WRAP obj->getRidgeFilteredImage(*_img, *out); END_WRAP }	cv::ximgproc::RidgeDetectionFilter::getRidgeFilteredImage()	classcv_1_1ximgproc_1_1RidgeDetectionFilter.html	Apply Ridge detection filter on input image.	_img : InputArray as supported by Sobel. img can be 1-Channel or 3-Channels. out : OutputArray of structure as RidgeDetectionFilter::ddepth. Output image with ridges.		2								
ximgproc_Ptr_RidgeDetectionFilter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_RidgeDetectionFilter_delete(cv::Ptr<cv::ximgproc::RidgeDetectionFilter> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::RidgeDetectionFilter	classcv_1_1ximgproc_1_1RidgeDetectionFilter.html	Applies Ridge Detection Filter to an input image. Implements Ridge detection similar to the one in Mathematica using the eigen values from the Hessian Matrix of the input image using Sobel Derivatives. Additional refinement can be done using Skeletonization and Binarization. Adapted from [66] and [165].							•			
ximgproc_Ptr_RidgeDetectionFilter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_RidgeDetectionFilter_get(cv::Ptr<cv::ximgproc::RidgeDetectionFilter> *ptr, cv::ximgproc::RidgeDetectionFilter **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::RidgeDetectionFilter	classcv_1_1ximgproc_1_1RidgeDetectionFilter.html	Applies Ridge Detection Filter to an input image. Implements Ridge detection similar to the one in Mathematica using the eigen values from the Hessian Matrix of the input image using Sobel Derivatives. Additional refinement can be done using Skeletonization and Binarization. Adapted from [66] and [165].								•		
ximgproc_seg_createGraphSegmentation	CVAPI(ExceptionStatus) ximgproc_segmentation_createGraphSegmentation(double sigma, float k, int min_size, cv::Ptr<cv::ximgproc::segmentation::GraphSegmentation> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::ximgproc::segmentation::createGraphSegmentation(sigma, k, min_size)); END_WRAP }	cv::ximgproc::segmentation::createGraphSegmentation()	group__ximgproc__segmentation.html	Creates a graph based segmentor.	sigma : The sigma parameter, used to smooth image k : The k parameter of the algorythm min_size : The minimum size of segments		3								
ximgproc_seg_Ptr_GraphSegmentation_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_GraphSegmentation_delete(cv::Ptr<cv::ximgproc::segmentation::GraphSegmentation> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html	Graph Based Segmentation Algorithm. The class implements the algorithm described in [73] .								•		
ximgproc_seg_Ptr_GraphSegmentation_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_GraphSegmentation_get(cv::Ptr<cv::ximgproc::segmentation::GraphSegmentation> *ptr, cv::ximgproc::segmentation::GraphSegmentation **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html	Graph Based Segmentation Algorithm. The class implements the algorithm described in [73] .									•		
ximgproc_seg_GraphSegmentation_processImage	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_processImage(cv::ximgproc::segmentation::GraphSegmentation *obj, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP obj->processImage(*src, *dst); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::processImage()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html	Segment an image and store output in dst.	src : The input image. Any number of channel (1 (Eg: Gray), 3 (Eg: RGB), 4 (Eg: RGB-D)) can be provided dst : The output segmentation. It's a CV_32SC1 Mat with the same number of cols and rows as input image, with an unique, sequential, id for each pixel.		2								
ximgproc_seg_GraphSegmentation_setSigma	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_setSigma(cv::ximgproc::segmentation::GraphSegmentation *obj, double value) { BEGIN_WRAP obj->setSigma(value); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::setSigma()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1								
ximgproc_seg_GraphSegmentation_getSigma	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_getSigma(cv::ximgproc::segmentation::GraphSegmentation *obj, double *returnValue) { BEGIN_WRAP *returnValue = obj->getSigma(); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::getSigma()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1								
ximgproc_seg_GraphSegmentation_setK	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_setK(cv::ximgproc::segmentation::GraphSegmentation *obj, float value) { BEGIN_WRAP obj->setK(value); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::setK()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1								

ximgproc_seg_GraphSegmentation_getK	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_getK(cv::ximgproc::segmentation::GraphSegmentation *obj, float *returnValue) { BEGIN_WRAP *returnValue = obj->getK(); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::getK()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1									
ximgproc_seg_GraphSegmentation_setMinSize	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_setMinSize(cv::ximgproc::segmentation::GraphSegmentation *obj, int value) { BEGIN_WRAP obj->setMinSize(value); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::setMinSize()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1									
ximgproc_seg_GraphSegmentation_getMinSize	CVAPI(ExceptionStatus) ximgproc_segmentation_GraphSegmentation_getMinSize(cv::ximgproc::segmentation::GraphSegmentation *obj, int *returnValue) { BEGIN_WRAP *returnValue = obj->getMinSize(); END_WRAP }	cv::ximgproc::segmentation::GraphSegmentation::getMinSize()	classcv_1_1ximgproc_1_1segmentation_1_1GraphSegmentation.html				1									
ximgproc_seg_SeleSchSegStrat_setImage	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentationStrategy_setImage(cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy *obj, cv::_InputArray *img, cv::_InputArray *regions, cv::_InputArray *sizes, int image_id) { BEGIN_WRAP obj->setImage(*img, *regions, *sizes, image_id); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy::setImage()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategy.html	Set a initial image, with a segmentation.	img : The input image. Any number of channel can be provided regions : A segmentation of the image. The parameter must be the same size of img. sizes : The sizes of different regions image_id : If not set to -1, try to cache pre-computations. If the same set og (img, regions, size) is used, the image_id need to be the same.		4								
ximgproc_seg_SeleSchSegStrat_get	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentationStrategy_get(cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy *obj, int r1, int r2, float *returnValue) { BEGIN_WRAP *returnValue = obj->get(r1, r2); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy::get()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategy.html	Return the score between two regions (between 0 and 1)	r1 : The first region r2 : The second region		2								
ximgproc_seg_SeleSchSegStrat_merge	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentationStrategy_merge(cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy *obj, int r1, int r2) { BEGIN_WRAP obj->merge(r1, r2); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy::merge()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategy.html	Inform the strategy that two regions will be merged.	r1 : The first region r2 : The second region		2								
ximgproc_seg_createSeleSchSegStratColor	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyColor(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor>(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyColor()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyColor()	group__ximgproc__segmentation.html	Create a new color-based strategy.			1								
ximgproc_seg_createSeleSchSegStratSize	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategySize(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize>(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategySize()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategySize()	group__ximgproc__segmentation.html	Create a new size-based strategy.			1								
ximgproc_seg_createSeleSchSegStratTexture	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyTexture(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture>(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyTexture()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyTexture()	group__ximgproc__segmentation.html	Create a new size-based strategy.			1								
ximgproc_seg_createSeleSchSegStratFill	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyFill(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyFill> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyFill>(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyFill()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyFill()	group__ximgproc__segmentation.html	Create a new fill-based strategy.			1								
ximgproc_seg_Ptr_SeleSchSegStratColor_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyColor_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor::~SelectiveSearchSegmentationStrategyColor()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyColor.html	Color-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].								•		
ximgproc_seg_Ptr_SeleSchSegStratSize_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategySize_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize::~SelectiveSearchSegmentationStrategySize()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategySize.html	Size-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].								•		
ximgproc_seg_Ptr_SeleSchSegStratTexture_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyTexture_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture::~SelectiveSearchSegmentationStrategyTexture()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyTexture.html	Texture-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].								•		

ximgproc_seg_Ptr_SeleSchSegStratFill_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyFill_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyFill> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyFill.html	Fill-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].									•			
ximgproc_seg_Ptr_SeleSchSegStratColor_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyColor_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyColor.html	Color-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].										•		
ximgproc_seg_Ptr_SeleSchSegStratSize_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategySize_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategySize **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategySize.html	Size-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].										•		
ximgproc_seg_Ptr_SeleSchSegStratTexture_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyTexture_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyTexture **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyTexture.html	Texture-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].										•		
ximgproc_seg_Ptr_SeleSchSegStratFill_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyFill_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyFill> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyFill **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyFill.html	Fill-based strategy for the selective search segmentation algorithm The class is implemented from the algorithm described in [252].										•		
ximgproc_seg_SeleSchSegStratMultiple_addStr	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentationStrategyMultiple_addStrategy(cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple *obj, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *g, float weight) { BEGIN_WRAP obj->addStrategy(*g, weight); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Add a new sub-strategy.	g : The strategy weight : The weight of the strategy		2									
ximgproc_seg_SeleSchSegStratMultiple_clearS	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentationStrategyMultiple_clearStrategies(cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple *obj) { BEGIN_WRAP obj->clearStrategies(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Remove all sub-strategies.			1									
ximgproc_seg_createSeleSchSegStratMultipleO	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyMultiple0(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyMultiple()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearc hSegmentationStrategyMultiple()	group__ximgproc__segmentation.html	Create a new multiple strategy and set one subtrategy.	s1 : The first strategy		1									
ximgproc_seg_createSeleSchSegStratMultiple1	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyMultiple1(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s1, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> **returnValue) { BEGIN_WRAP *returnValue = clone(createSelectiveSearchSegmentationStrategyMultiple(*s1)); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm entationStrategyMultiple()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Regroup multiple strategies for the selective search segmentation algorithm.			1									
ximgproc_seg_createSeleSchSegStratMultiple2	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyMultiple2(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s1, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s2, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> **returnValue) { BEGIN_WRAP *returnValue = clone(createSelectiveSearchSegmentationStrategyMultiple(*s1, *s2)); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm entationStrategyMultiple()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Regroup multiple strategies for the selective search segmentation algorithm.			2									

ximgproc_seg_createSeleSchSegStratMultiple3	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyMultiple3(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s1, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s2, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s3, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> **returnValue) { BEGIN_WRAP *returnValue = clone(cv::ximgproc::segmentation::createSelectiveSearchS	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyMultiple()	group__ximgproc__segmentation.html	Create a new multiple strategy and set three subtrategies, with equal weights.	s1 : The first strategy s2 : The second strategy s3 : The third strategy		3								
ximgproc_seg_createSeleSchSegStratMultiple4	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyMultiple4(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s1, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s2, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s3, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s4, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> **returnValue) { BEGIN_WRAP	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyMultiple()	group__ximgproc__segmentation.html	Create a new multiple strategy and set four subtrategies, with equal weights.	s1 : The first strategy s2 : The second strategy s3 : The third strategy s4 : The forth strategy		4								
ximgproc_seg_Ptr_SeleSchSegStratMultiple_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyMultiple_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Regroup multiple strategies for the selective search segmentation algorithm.								•		
ximgproc_seg_Ptr_SeleSchSegStratMultiple_ge	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentationStrategyMultiple_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyMultiple **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegm	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentationStrategyMultiple.html	Regroup multiple strategies for the selective search segmentation algorithm.									•	
ximgproc_seg_SeleSchSeg_setBaseImage	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_setBaseImage(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, cv::_InputArray *img) { BEGIN_WRAP obj->setBaseImage(*img); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::setBaseImage()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Set a image used by switch* functions to initialize the class.	img : The image		1								
ximgproc_seg_SeleSchSeg_switchToSingleStra	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_switchToSingleStrategy(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, int k, float sigma) { BEGIN_WRAP obj->switchToSingleStrategy(k, sigma); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::switchToSingleStrategy()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Initialize the class with the 'Single stragegy' parameters described in [252].	k : The k parameter for the graph segmentation sigma : The sigma parameter for the graph segmentation		2								
ximgproc_seg_SeleSchSeg_switchToSelectiveS	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_switchToSelectiveSearchFast(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, int base_k, int inc_k, float sigma) { BEGIN_WRAP obj->switchToSelectiveSearchFast(base_k, inc_k, sigma); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::switchToSelectiveSearchFast()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Initialize the class with the 'Selective search fast' parameters described in [252].	base_k : The k parameter for the first graph segmentation inc_k : The increment of the k parameter for all graph segmentations sigma : The sigma parameter for the graph segmentation		3								
ximgproc_seg_SeleSchSeg_switchToSelectiveS	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_switchToSelectiveSearchQuality(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, int base_k, int inc_k, float sigma) { BEGIN_WRAP obj->switchToSelectiveSearchQuality(base_k, inc_k, sigma); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::switchToSelectiveSearchQuality()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Initialize the class with the 'Selective search fast' parameters described in [252].	base_k : The k parameter for the first graph segmentation inc_k : The increment of the k parameter for all graph segmentations sigma : The sigma parameter for the graph segmentation		3								
ximgproc_seg_SeleSchSeg_addImage	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_addImage(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, cv::_InputArray *img) { BEGIN_WRAP obj->addImage(*img); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::addImage()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Add a new image in the list of images to process.	img : The image		1								
ximgproc_seg_SeleSchSeg_clearImages	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_clearImages(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj) { BEGIN_WRAP obj->clearImages(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::clearImages()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Clear the list of images to process.			1								
ximgproc_seg_SeleSchSeg_addGraphSegmenta	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_addGraphSegmentation(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, cv::Ptr<cv::ximgproc::segmentation::GraphSegmentation> *g) { BEGIN_WRAP obj->addGraphSegmentation(*g); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::addGraphSegmentation()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Add a new graph segmentation in the list of graph segementations to process.	g : The graph segmentation		1								
ximgproc_seg_SeleSchSeg_clearGraphSegmen	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_clearGraphSegmentations(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj) { BEGIN_WRAP obj->clearGraphSegmentations(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::clearGraphSegmentations()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Clear the list of graph segmentations to process;.			1								

ximgproc_seg_SeleSchSeg_addStrategy	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_addStrategy(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategy> *s) { BEGIN_WRAP obj->addStrategy(*s); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::addStrategy()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Add a new strategy in the list of strategy to process.	s : The strategy		1								
ximgproc_seg_SeleSchSeg_clearStrategies	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_clearStrategies(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj) { BEGIN_WRAP obj->clearStrategies(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::clearStrategies()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Clear the list of strategy to process;.			1								
ximgproc_seg_SeleSchSeg_process	CVAPI(ExceptionStatus) ximgproc_segmentation_SelectiveSearchSegmentation_process(cv::ximgproc::segmentation::SelectiveSearchSegmentation *obj, std::vector<cv::Rect> *rects) { BEGIN_WRAP obj->process(*rects); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation::process()	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Based on all images, graph segmentations and stragies, computes all possible rects and return them.	rects : The list of rects. The first ones are more relevents than the lasts ones.		1								
ximgproc_seg_createSeleSchSeg	CVAPI(ExceptionStatus) ximgproc_segmentation_createSelectiveSearchSegmentationStrategyColor(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentationStrategyColor>(cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyColor()); END_WRAP }	cv::ximgproc::segmentation::createSelectiveSearchSegmentationStrategyColor()	group__ximgproc__segmentation.html	Create a new color-based strategy.			1								
ximgproc_seg_Ptr_SeleSchSeg_delete	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentation_delete(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentation> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Selective search segmentation algorithm The class implements the algorithm described in [252].								•			
ximgproc_seg_Ptr_SeleSchSeg_get	CVAPI(ExceptionStatus) ximgproc_segmentation_Ptr_SelectiveSearchSegmentation_get(cv::Ptr<cv::ximgproc::segmentation::SelectiveSearchSegmentation> *ptr, cv::ximgproc::segmentation::SelectiveSearchSegmentation **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::segmentation::SelectiveSearchSegmentation	classcv_1_1ximgproc_1_1segmentation_1_1SelectiveSearchSegmentation.html	Selective search segmentation algorithm The class implements the algorithm described in [252].									•		
ximgproc_createRFFeatureGetter	CVAPI(ExceptionStatus) ximgproc_createRFFeatureGetter(cv::Ptr<cv::ximgproc::RFFeatureGetter> **returnValue) { BEGIN_WRAP *returnValue = new cv::Ptr<cv::ximgproc::RFFeatureGetter>(cv::ximgproc::createRFFeatureGetter()); END_WRAP }	cv::ximgproc::createRFFeatureGetter()	group__ximgproc__edge.html				1								
ximgproc_Ptr_RFFeatureGetter_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_RFFeatureGetter_delete(cv::Ptr<cv::ximgproc::RFFeatureGetter> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::RFFeatureGetter	classcv_1_1ximgproc_1_1RFFeatureGetter.html	Helper class for training part of [P. Dollar and C. L. Zitnick. Structured Forests for Fast Edge Detection, 2013].								•			
ximgproc_Ptr_RFFeatureGetter_get	CVAPI(ExceptionStatus) ximgproc_Ptr_RFFeatureGetter_get(cv::Ptr<cv::ximgproc::RFFeatureGetter> *ptr, cv::ximgproc::RFFeatureGetter **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::RFFeatureGetter	classcv_1_1ximgproc_1_1RFFeatureGetter.html	Helper class for training part of [P. Dollar and C. L. Zitnick. Structured Forests for Fast Edge Detection, 2013].									•		
ximgproc_RFFeatureGetter_getFeatures	CVAPI(ExceptionStatus) ximgproc_RFFeatureGetter_getFeatures(cv::ximgproc::RFFeatureGetter *obj, cv::Mat *src, cv::Mat *features, const int gnrmRad, const int gsmthRad, const int shrink, const int outNum, const int gradNum) { BEGIN_WRAP obj->getFeatures(*src, *features, gnrmRad, gsmthRad, shrink, outNum, gradNum); END_WRAP }	cv::ximgproc::RFFeatureGetter::getFeatures()	classcv_1_1ximgproc_1_1RFFeatureGetter.html	This functions extracts feature channels from src. Than StructureEdgeDetection uses this feature space to detect edges.	src : : source image to extract features features : : output n-channel floating point feature matrix. gnrmRad : : __rf.options.gradientNormalizationRadius gsmthRad : : __rf.options.gradientSmoothingRadius shrink : : __rf.options.shrinkNumber outNum : : __rf.options.numberOfOutputChannels gradNum : : __rf.options.numberOfGradientOrientations		7								
ximgproc_createStructuredEdgeDetection	CVAPI(ExceptionStatus) ximgproc_createStructuredEdgeDetection(const char *model, cv::Ptr<cv::ximgproc::RFFeatureGetter> *howToGetFeatures, cv::Ptr<cv::ximgproc::StructuredEdgeDetection> **returnValue) { BEGIN_WRAP if (howToGetFeatures == nullptr) *returnValue = new cv::Ptr<cv::ximgproc::StructuredEdgeDetection>(cv::ximgproc::createStructuredEdgeDetection(model)); else *returnValue = new cv::Ptr<cv::ximgproc::StructuredEdgeDetection>(cv::ximgproc::createStructuredEdgeDetection(model, *howToGetFeatures)); END_WRAP }	cv::ximgproc::createStructuredEdgeDetection()	group__ximgproc__edge.html	The only constructor	model : : name of the file where the model is stored howToGetFeatures : : optional object inheriting from RFFeatureGetter. You need it only if you would like to train your own forest, pass NULL otherwise		2								
ximgproc_Ptr_StructuredEdgeDetection_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_StructuredEdgeDetection_delete(cv::Ptr<cv::ximgproc::StructuredEdgeDetection> *obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::StructuredEdgeDetection	classcv_1_1ximgproc_1_1StructuredEdgeDetection.html	Class implementing edge detection algorithm from [57] :								•			
ximgproc_Ptr_StructuredEdgeDetection_get	CVAPI(ExceptionStatus) ximgproc_Ptr_StructuredEdgeDetection_get(cv::Ptr<cv::ximgproc::StructuredEdgeDetection> *ptr, cv::ximgproc::StructuredEdgeDetection **returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::StructuredEdgeDetection	classcv_1_1ximgproc_1_1StructuredEdgeDetection.html	Class implementing edge detection algorithm from [57] :									•		
ximgproc_StructuredEdgeDetection_detectEdges	CVAPI(ExceptionStatus) ximgproc_StructuredEdgeDetection_detectEdges(cv::ximgproc::StructuredEdgeDetection *obj, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP obj->detectEdges(*src, *dst); END_WRAP }	cv::ximgproc::StructuredEdgeDetection::detectEdges()	classcv_1_1ximgproc_1_1StructuredEdgeDetection.html	The function detects edges in src and draw them to dst.	_src : source image (RGB, float, in [0;1]) to detect edges _dst : destination image (grayscale, float, in [0;1]) where edges are drawn	The algorithm underlies this function is much more robust to texture presence, than common approaches, e.g. SobelSee alsoSobel, Canny	2								

ximgproc_StructuredEdgeDetection_computeO	CVAPI(ExceptionStatus) ximgproc_StructuredEdgeDetection_computeOrientation(cv::ximgproc::StructuredEdgeDetection *obj, cv::_InputArray *src, cv::_OutputArray *dst) { BEGIN_WRAP obj->computeOrientation(*src, *dst); END_WRAP }	cv::ximgproc::StructuredEdgeDetection::computeOrientation()	classcv_1_1ximgproc_1_1StructuredEdgeDetection.html	The function computes orientation from edge image.	_src : edge image. _dst : orientation image.		2									
ximgproc_StructuredEdgeDetection_edgesNms	CVAPI(ExceptionStatus) ximgproc_StructuredEdgeDetection_edgesNms(cv::ximgproc::StructuredEdgeDetection *obj, cv::_InputArray *edge_image, cv::_InputArray *orientation_image, cv::_OutputArray *dst, int r, int s, float m, int isParallel) { BEGIN_WRAP obj->edgesNms(*edge_image, *orientation_image, *dst, r, s, m, isParallel != 0); END_WRAP }	cv::ximgproc::StructuredEdgeDetection::edgesNms()	classcv_1_1ximgproc_1_1StructuredEdgeDetection.html	The function edgenms in edge image and suppress edges where edge is stronger in orthogonal direction.	edge_image : edge image from detectEdges function. orientation_image : orientation image from computeOrientation function. _dst : suppressed image (grayscale, float, in [0;1]) r : radius for NMS suppression. s : radius for boundary suppression. m : multiplier for conservative suppression. isParallel : enables/disables parallel computing.		7									
ximgproc_Ptr_SuperpixelLSC_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelLSC_delete(cv::Ptr<cv::ximgproc::SuperpixelLSC>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::SuperpixelLSC	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Class implementing the LSC (Linear Spectral Clustering) superpixels algorithm described in [141]. LSC (Linear Spectral Clustering) produces compact and uniform superpixels with low computational costs. Basically, a normalized cuts formulation of the superpixel segmentation is adopted based on a similarity metric that measures the color similarity and space proximity between image pixels. LSC is of linear computational complexity and high memory efficiency and is able to preserve global properties of images								•		
ximgproc_Ptr_SuperpixelLSC_get	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelLSC_get(cv::Ptr<cv::ximgproc::SuperpixelLSC>* ptr, cv::ximgproc::SuperpixelLSC** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::SuperpixelLSC	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Class implementing the LSC (Linear Spectral Clustering) superpixels algorithm described in [141]. LSC (Linear Spectral Clustering) produces compact and uniform superpixels with low computational costs. Basically, a normalized cuts formulation of the superpixel segmentation is adopted based on a similarity metric that measures the color similarity and space proximity between image pixels. LSC is of linear computational complexity and high memory efficiency and is able to preserve global properties of images										•	
ximgproc_SuperpixelLSC_getNumberOfSuperpi	CVAPI(ExceptionStatus) ximgproc_SuperpixelLSC_getNumberOfSuperpixels(cv::ximgproc::SuperpixelLSC* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->getNumberOfSuperpixels(); END_WRAP }	cv::ximgproc::SuperpixelLSC::getNumberOfSuperpixels()	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Calculates the actual amount of superpixels on a given segmentation computed and stored in SuperpixelLSC object.			1									
ximgproc_SuperpixelLSC_iterate	CVAPI(ExceptionStatus) ximgproc_SuperpixelLSC_iterate(cv::ximgproc::SuperpixelLSC* obj, int num_iterations) { BEGIN_WRAP obj->iterate(num_iterations); END_WRAP }	cv::ximgproc::SuperpixelLSC::iterate()	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Calculates the superpixel segmentation on a given image with the initialized parameters in the SuperpixelLSC object.	num_iterations : Number of iterations. Higher number improves the result.	This function can be called again without the need of initializing the algorithm with createSuperpixelLSC(). This save the computational cost of allocating memory for all the structures of the algorithm.The function computes the superpixels segmentation of an image with the parameters initialized with the function createSuperpixelLSC(). The algorithms starts from a grid of superpixels and then refines the boundaries by proposing updates of edges boundaries.	1									
ximgproc_SuperpixelLSC_getLabels	CVAPI(ExceptionStatus) ximgproc_SuperpixelLSC_getLabels(cv::ximgproc::SuperpixelLSC* obj, cv::_OutputArray *labels_out) { BEGIN_WRAP obj->getLabels(*labels_out); END_WRAP }	cv::ximgproc::SuperpixelLSC::getLabels()	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Returns the segmentation labeling of the image.	labels_out : Return: A CV_32SC1 integer array containing the labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	Each label represents a superpixel, and each pixel is assigned to one superpixel label.The function returns an image with the labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	1									
ximgproc_SuperpixelLSC_getLabelContourMask	CVAPI(ExceptionStatus) ximgproc_SuperpixelLSC_getLabelContourMask(cv::ximgproc::SuperpixelLSC* obj, cv::_OutputArray *image, int thick_line) { BEGIN_WRAP obj->getLabelContourMask(*image, thick_line != 0); END_WRAP }	cv::ximgproc::SuperpixelLSC::getLabelContourMask()	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Returns the mask of the superpixel segmentation stored in SuperpixelLSC object.	image : Return: CV_8U1 image mask where -1 indicates that the pixel is a superpixel border, and 0 otherwise. thick_line : If false, the border is only one pixel wide, otherwise all pixels at the border are masked.	The function return the boundaries of the superpixel segmentation.	2									
ximgproc_SuperpixelLSC_enforceLabelConnect	CVAPI(ExceptionStatus) ximgproc_SuperpixelLSC_enforceLabelConnectivity(cv::ximgproc::SuperpixelLSC* obj, int min_element_size) { BEGIN_WRAP obj->enforceLabelConnectivity(min_element_size); END_WRAP }	cv::ximgproc::SuperpixelLSC::enforceLabelConnectivity()	classcv_1_1ximgproc_1_1SuperpixelLSC.html	Enforce label connectivity.	min_element_size : The minimum element size in percents that should be absorbed into a bigger superpixel. Given resulted average superpixel size valid value should be in 0-100 range, 25 means that less then a quarter sized superpixel should be absorbed, this is default.	The function merge component that is too small, assigning the previously found adjacent label to this component. Calling this function may change the final number of superpixels.	1									
ximgproc_createSuperpixelLSC	CVAPI(ExceptionStatus) ximgproc_createSuperpixelLSC(cv::_InputArray *image, int region_size, float ratio, cv::Ptr<cv::ximgproc::SuperpixelLSC>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createSuperpixelLSC(*image, region_size, ratio); *returnValue = new cv::Ptr<cv::ximgproc::SuperpixelLSC>(ptr); END_WRAP }	cv::ximgproc::createSuperpixelLSC()	group__ximgproc__superpixel.html	Class implementing the LSC (Linear Spectral Clustering) superpixels.	image : Image to segment region_size : Chooses an average superpixel size measured in pixels ratio : Chooses the enforcement of superpixel compactness factor of superpixel	The function initializes a SuperpixelLSC object for the input image. It sets the parameters of superpixel algorithm, which are: region_size and ruler. It preallocate some buffers for future computing iterations over the given image. An example of LSC is illustrated in the following picture. For enanched results it is recommended for color images to preprocess image with little gaussian blur with a small 3 x 3 kernel and additional conversion into CieLAB color space.image	3									
ximgproc_Ptr_SuperpixelSEEDS_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelSEEDS_delete(cv::Ptr<cv::ximgproc::SuperpixelSEEDS>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::SuperpixelSEEDS	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Class implementing the SEEDS (Superpixels Extracted via Energy-Driven Sampling) superpixels algorithm described in [257] . The algorithm uses an efficient hill-climbing algorithm to optimize the superpixels' energy function that is based on color histograms and a boundary term, which is optional. The energy function encourages superpixels to be of the same color, and if the boundary term is activated, the superpixels have smooth boundaries and are of similar shape. In practice it starts from a regular grid of superpixels and moves the pixels or blocks of pixels at the boundaries to refine the solution. The algorithm runs in real-time using a single CPU.										•	
ximgproc_Ptr_SuperpixelSEEDS_get	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelSEEDS_get(cv::Ptr<cv::ximgproc::SuperpixelSEEDS>* ptr, cv::ximgproc::SuperpixelSEEDS** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::SuperpixelSEEDS	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Class implementing the SEEDS (Superpixels Extracted via Energy-Driven Sampling) superpixels algorithm described in [257] . The algorithm uses an efficient hill-climbing algorithm to optimize the superpixels' energy function that is based on color histograms and a boundary term, which is optional. The energy function encourages superpixels to be of the same color, and if the boundary term is activated, the superpixels have smooth boundaries and are of similar shape. In practice it starts from a regular grid of superpixels and moves the pixels or blocks of pixels at the boundaries to refine the solution. The algorithm runs in real-time using a single CPU.											•	
ximgproc_SuperpixelSEEDS_getNumberOfSupe	CVAPI(ExceptionStatus) ximgproc_SuperpixelSEEDS_getNumberOfSuperpixels(cv::ximgproc::SuperpixelSEEDS* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->getNumberOfSuperpixels(); END_WRAP }	cv::ximgproc::SuperpixelSEEDS::getNumberOfSuperpixels()	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Calculates the superpixel segmentation on a given image stored in SuperpixelSEEDS object.		The function computes the superpixels segmentation of an image with the parameters initialized with the function createSuperpixelSEEDS().	1									

ximgproc_SuperpixelSEEDS_iterate	CVAPI(ExceptionStatus) ximgproc_SuperpixelSEEDS_iterate(cv::ximgproc::SuperpixelSEEDS* obj, cv::_InputArray *img, int num_iterations) { BEGIN_WRAP obj->iterate(*img, num_iterations); END_WRAP }	cv::ximgproc::SuperpixelSEEDS::iterate()	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Calculates the superpixel segmentation on a given image with the initialized parameters in the SuperpixelSEEDS object.	img : Input image. Supported formats: CV_8U, CV_16U, CV_32F. Image size & number of channels must match with the initialized image size & channels with the function createSuperpixelSEEDS(). It should be in HSV or Lab color space. Lab is a bit better, but also slower. num_iterations : Number of pixel level iterations. Higher number improves the result.	This function can be called again for other images without the need of initializing the algorithm with createSuperpixelSEEDS(). This save the computational cost of allocating memory for all the structures of the algorithm.The function computes the superpixels segmentation of an image with the parameters initialized with the function createSuperpixelSEEDS(). The algorithms starts from a grid of superpixels and then refines the boundaries by proposing updates of blocks of pixels that lie at the boundaries from large to smaller size, finalizing with proposing pixel updates. An illustrative example can be seen below.image	2								
ximgproc_SuperpixelSEEDS_getLabels	CVAPI(ExceptionStatus) ximgproc_SuperpixelSEEDS_getLabels(cv::ximgproc::SuperpixelSEEDS* obj, cv::_OutputArray* labels_out) { BEGIN_WRAP obj->getLabels(*labels_out); END_WRAP }	cv::ximgproc::SuperpixelSEEDS::getLabels()	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Returns the segmentation labeling of the image.	labels_out : Return: A CV_32UC1 integer array containing the labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	Each label represents a superpixel, and each pixel is assigned to one superpixel label.The function returns an image with sethe labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	1								
ximgproc_SuperpixelSEEDS_getLabelContourMask	CVAPI(ExceptionStatus) ximgproc_SuperpixelSEEDS_getLabelContourMask(cv::ximgproc::SuperpixelSEEDS* obj, cv::_OutputArray* image, int thick_line) { BEGIN_WRAP obj->getLabelContourMask(*image, thick_line != 0); END_WRAP }	cv::ximgproc::SuperpixelSEEDS::getLabelContourMask()	classcv_1_1ximgproc_1_1SuperpixelSEEDS.html	Returns the mask of the superpixel segmentation stored in SuperpixelSEEDS object.	image : Return: CV_8UC1 image mask where -1 indicates that the pixel is a superpixel border, and 0 otherwise. thick_line : If false, the border is only one pixel wide, otherwise all pixels at the border are masked.	The function return the boundaries of the superpixel segmentation.Note (Python) A demo on how to generate superpixels in images from the webcam can be found at opencv_source_code/samples/python2/seeds.py (cpp) A demo on how to generate superpixels in images from the webcam can be found at opencv_source_code/modules/ximgproc/samples/seeds.cpp. By adding a file image as a command line argument, the static image will be used instead of the webcam. It will show a window with the video from the webcam with the superpixel boundaries marked in red (see below). Use Space to switch between different output modes. At the top of the window there are 4 sliders, from which the user can change on-the-fly the number of superpixels, the number of block levels, the strength of the boundary prior term to modify the shape, and the number of iterations at pixel level. This is useful to play with the parameters and set them to	2								
ximgproc_createSuperpixelSEEDS	CVAPI(ExceptionStatus) ximgproc_createSuperpixelSEEDS(int image_width, int image_height, int image_channels, int num_superpixels, int num_levels, int prior, int histogram_bins, int double_step, cv::Ptr<cv::ximgproc::SuperpixelSEEDS>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createSuperpixelSEEDS(image_width, image_height, image_channels, num_superpixels, num_levels, prior, histogram_bins, double_step); *returnValue = new cv::Ptr<cv::ximgproc::SuperpixelSEEDS>(ptr); END_WRAP }	cv::ximgproc::createSuperpixelSEEDS()	group__ximgproc__superpixel.html	Initializes a SuperpixelSEEDS object.	image_width : Image width. image_height : Image height. image_channels : Number of channels of the image. num_superpixels : Desired number of superpixels. Note that the actual number may be smaller due to restrictions (depending on the image size and num_levels). Use getNumberOfSuperpixels() to get the actual number. num_levels : Number of block levels. The more levels, the more accurate is the segmentation, but needs more memory and CPU time. prior : enable 3x3 shape smoothing term if >0. A larger value leads to smoother shapes. prior must be in the range [0, 5]. histogram_bins : Number of histogram bins. double_step : If true, iterate each block level twice for higher accuracy.	The function initializes a SuperpixelSEEDS object for the input image. It stores the parameters of the image: image_width, image_height and image_channels. It also sets the parameters of the SEEDS superpixel algorithm, which are: num_superpixels, num_levels, use_prior, histogram_bins and double_step.The number of levels in num_levels defines the amount of block levels that the algorithm use in the optimization. The initialization is a grid, in which the superpixels are equally distributed through the width and the height of the image. The larger blocks correspond to the superpixel size, and the levels with smaller blocks are formed by dividing the larger blocks into 2 x 2 blocks of pixels, recursively until the smaller block level. An example of initialization of 4 block levels is illustrated in the following figure.image	8								
ximgproc_Ptr_SuperpixelSLIC_delete	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelSLIC_delete(cv::Ptr<cv::ximgproc::SuperpixelSLIC>* obj) { BEGIN_WRAP delete obj; END_WRAP }	cv::ximgproc::SuperpixelSLIC	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Class implementing the SLIC (Simple Linear Iterative Clustering) superpixels algorithm described in [1]. SLIC (Simple Linear Iterative Clustering) clusters pixels using pixel channels and image plane space to efficiently generate compact, nearly uniform superpixels. The simplicity of approach makes it extremely easy to use a lone parameter specifies the number of superpixels and the efficiency of the algorithm makes it very practical. Several optimizations are available for SLIC class: SLICO stands for "Zero parameter SLIC" and it is an optimization of baseline SLIC described in [1]. MSLIC stands for "Manifold SLIC" and it is an optimization of baseline SLIC described in [150].								•		
ximgproc_Ptr_SuperpixelSLIC_get	CVAPI(ExceptionStatus) ximgproc_Ptr_SuperpixelSLIC_get(cv::Ptr<cv::ximgproc::SuperpixelSLIC>* ptr, cv::ximgproc::SuperpixelSLIC** returnValue) { BEGIN_WRAP *returnValue = ptr->get(); END_WRAP }	cv::ximgproc::SuperpixelSLIC	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Class implementing the SLIC (Simple Linear Iterative Clustering) superpixels algorithm described in [1]. SLIC (Simple Linear Iterative Clustering) clusters pixels using pixel channels and image plane space to efficiently generate compact, nearly uniform superpixels. The simplicity of approach makes it extremely easy to use a lone parameter specifies the number of superpixels and the efficiency of the algorithm makes it very practical. Several optimizations are available for SLIC class: SLICO stands for "Zero parameter SLIC" and it is an optimization of baseline SLIC described in [1]. MSLIC stands for "Manifold SLIC" and it is an optimization of baseline SLIC described in [150].								•		
ximgproc_SuperpixelSLIC_getNumberOfSuperpixels	CVAPI(ExceptionStatus) ximgproc_SuperpixelSLIC_getNumberOfSuperpixels(cv::ximgproc::SuperpixelSLIC* obj, int* returnValue) { BEGIN_WRAP *returnValue = obj->getNumberOfSuperpixels(); END_WRAP }	cv::ximgproc::SuperpixelSLIC::getNumberOfSuperpixels()	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Calculates the actual amount of superpixels on a given segmentation computed and stored in SuperpixelSLIC object.			1								
ximgproc_SuperpixelSLIC_iterate	CVAPI(ExceptionStatus) ximgproc_SuperpixelSLIC_iterate(cv::ximgproc::SuperpixelSLIC* obj, int num_iterations) { BEGIN_WRAP obj->iterate(num_iterations); END_WRAP }	cv::ximgproc::SuperpixelSLIC::iterate()	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Calculates the superpixel segmentation on a given image with the initialized parameters in the SuperpixelSLIC object.	num_iterations : Number of iterations. Higher number improves the result.	This function can be called again without the need of initializing the algorithm with createSuperpixelSLIC(). This save the computational cost of allocating memory for all the structures of the algorithm.The function computes the superpixels segmentation of an image with the parameters initialized with the function createSuperpixelSLIC(). The algorithms starts from a grid of superpixels and then refines the boundaries by proposing updates of edges boundaries.	1								
ximgproc_SuperpixelSLIC_getLabels	CVAPI(ExceptionStatus) ximgproc_SuperpixelSLIC_getLabels(cv::ximgproc::SuperpixelSLIC* obj, cv::_OutputArray* labels_out) { BEGIN_WRAP obj->getLabels(*labels_out); END_WRAP }	cv::ximgproc::SuperpixelSLIC::getLabels()	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Returns the segmentation labeling of the image.	labels_out : Return: A CV_32SC1 integer array containing the labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	Each label represents a superpixel, and each pixel is assigned to one superpixel label.The function returns an image with the labels of the superpixel segmentation. The labels are in the range [0, getNumberOfSuperpixels()].	1								
ximgproc_SuperpixelSLIC_getLabelContourMask	CVAPI(ExceptionStatus) ximgproc_SuperpixelSLIC_getLabelContourMask(cv::ximgproc::SuperpixelSLIC* obj, cv::_OutputArray* image, int thick_line) { BEGIN_WRAP obj->getLabelContourMask(*image, thick_line != 0); END_WRAP }	cv::ximgproc::SuperpixelSLIC::getLabelContourMask()	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Returns the mask of the superpixel segmentation stored in SuperpixelSLIC object.	image : Return: CV_8U1 image mask where -1 indicates that the pixel is a superpixel border, and 0 otherwise. thick_line : If false, the border is only one pixel wide, otherwise all pixels at the border are masked.	The function return the boundaries of the superpixel segmentation.	2								
ximgproc_SuperpixelSLIC_enforceLabelConnect	CVAPI(ExceptionStatus) ximgproc_SuperpixelSLIC_enforceLabelConnectivity(cv::ximgproc::SuperpixelSLIC* obj, int min_element_size) { BEGIN_WRAP obj->enforceLabelConnectivity(min_element_size); END_WRAP }	cv::ximgproc::SuperpixelSLIC::enforceLabelConnectivity()	classcv_1_1ximgproc_1_1SuperpixelSLIC.html	Enforce label connectivity.	min_element_size : The minimum element size in percents that should be absorbed into a bigger superpixel. Given resulted average superpixel size valid value should be in 0-100 range, 25 means that less then a quarter sized superpixel should be absorbed, this is default.	The function merge component that is too small, assigning the previously found adjacent label to this component. Calling this function may change the final number of superpixels.	1								
ximgproc_createSuperpixelSLIC	CVAPI(ExceptionStatus) ximgproc_createSuperpixelSLIC(cv::_InputArray *image, int algorithm, int region_size, float ruler, cv::Ptr<cv::ximgproc::SuperpixelSLIC>** returnValue) { BEGIN_WRAP const auto ptr = cv::ximgproc::createSuperpixelSLIC(*image, algorithm, region_size, ruler); *returnValue = new cv::Ptr<cv::ximgproc::SuperpixelSLIC>(ptr); END_WRAP }	cv::ximgproc::createSuperpixelSLIC()	group__ximgproc__superpixel.html	Initialize a SuperpixelSLIC object.	image : Image to segment algorithm : Chooses the algorithm variant to use: SLIC segments image using a desired region_size, and in addition SLICO will optimize using adaptive compactness factor, while MSLIC will optimize using manifold methods resulting in more content-sensitive superpixels. region_size : Chooses an average superpixel size measured in pixels ruler : Chooses the enforcement of superpixel smoothness factor of superpixel	The function initializes a SuperpixelSLIC object for the input image. It sets the parameters of choosed superpixel algorithm, which are: region_size and ruler. It preallocate some buffers for future computing iterations over the given image. For enanched results it is recommended for color images to preprocess image with little gaussian blur using a small 3 x 3 kernel and additional conversion into CieLAB color space. An example of SLIC versus SLICO and MSLIC is illustrated in the following picture.image	4								