Année 2024/2025
Master Informatique Visuelle

Série de travaux pratiques n°2
Vision par Artificielle

Tools for OpenCV-Python programming

cv.findChessboardCorners

cv.findChessboardCorners(image, patternSize[, corners[, flags]]) ->retval, corners
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.

Parameters
Image: Source chessboard view. It must be an 8-bit grayscale or color image.
patternSize: Number of inner corners per a chessboard row and column (patternSize = 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.

Figure. Example for (9,6) detected corners

Cv2.Cornersubpix()
cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)

To find more exact corner positions (more exact than integer pixels). Implementation of the work:
S. Garrido-Jurado, R. Munoz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6):2280 – 2292, 2014.

Parameters

image: Input single-channel, 8-bit grayscale or float image
corners: Array that holds the initial approximate location of corners
winSize: Size of the neighborhood where it searches for corners. This is the Half of the side length of the search window. For example, if winSize=Size(5,5) , then a (5*2+1)×(5*2+1)=11×11 search window is used
zeroZone: This is the half of the neighborhood size we want to reject. If you don't want to reject anything pass (-1.-1)
criteria: Termination criteria. You can either stop it after a specified number of iterations or a certain accuracy is achieved, or whichever occurs first.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# max number of iterations=30
#  epsilon = 0.001
```

drawChessboardCorners

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.
cv.drawChessboardCorners(image, patternSize, corners, patternWasFound) -> image

Parameters
Image: Destination image. It must be an 8-bit color image.
patternSize: Number of inner corners per a chessboard row and column (patternSize = cv::Size(points_per_row,points_per_column)).
Corners: Array of detected corners, the output of findChessboardCorners.
patternWasFound: Parameter indicating whether the complete board was found or not. The return value of findChessboardCorners should be passed here.

cv.calibrateCamera

cv.calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs[, rvecs[, tvecs[, flags[, criteria]]]]) ->retval, cameraMatrix, distCoeffs, rvecs, tvecs

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

Parameters
objectPoints: In the new interface it is a vector of vectors of calibration pattern points in the calibration pattern coordinate space (<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 (<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 : internal parameters (fx, fy, Ox, Oy)
distCoeffs: Input/output vector of distortion coefficients (k1,k2,p1,p2,k3) of 5. Or for 4, 8, 12 or 14 elements.
x' = x + x*(K1*r^2 + K2*r^4 + K3*r^6) + P1*(r^2 + 2*x^2) + 2*P2*x*y
y' = y + y*(K1*r^2 + K2*r^4 + K3*r^6) + P2*(r^2 + 2*y^2) + 2*P1*x*y
r is the Euclidean distance between the distorted image point and the distortion center.
The k coefficients are the radial components of the distortion model, while the p coefficients are the tangential components.

Rvecs: Output vector of rotation vectors estimated for each pattern view (e.g. std::vector<cv::Mat>>). That is, each i^{th} rotation vector together with the corresponding i^{th} translation vector (see the next output parameter description) brings the calibration pattern from the object coordinate space (in which object points are specified) to the camera coordinate space. In more technical terms, the tuple of the i^{th}

rotation and translation vector performs a change of basis from object coordinate space to camera coordinate space. Due to its duality, this tuple is equivalent to the position of the calibration pattern with respect to the camera coordinate space.
Tvecs: Output vector of translation vectors estimated for each pattern view, see parameter describtion above.

projectPoints
Projects 3D points to an image plane.

cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
Python:
cv.projectPoints(objectPoints, rvec, tvec, cameraMatrix, distCoeffs[, imagePoints]) -> imagePoints

Parameters
objectPoints: Array of object points expressed wrt. the world coordinate frame, where N is the number of points in the view.
Rvec: The rotation vector that, together with tvec, performs a change of basis from world to camera coordinate system**.**
Tvec: The translation vector, see parameter description above.
cameraMatrix:
distCoeffs: Input vector of distortion coefficients (k1,k2,p1,p2,k3) of 5 elements. If the vector is empty, the zero distortion coefficients are assumed.
imagePoints: Output array of image points

Exercice 1

Appliquer le programme joint à cette série qui prend en entrée une série d'images de grilles obtenues par le déplacement de la caméra. Et qui estime la matrice de calibration (paramètres intrinsèques) et les paramètres extrinsèques.
Discutez les résultats obtenus.

Exercice 2.

Il s'agit de prendre une paire d'images par une caméra avec un mouvement de translation horizontale dont la distance de translation b est connue.
En ayant calibré auparavant la caméra (paramètres intrinsèques), calculer les coordonnées 3D points de la scène après avoir localisé les paires en correspondance en utilisant SIFT.

Exercice 3.

Nous disposons de paires d'images prises par une caméra avec un mouvement de translation axiale. Les images sont données par la dataset middlebury:
https://vision.middlebury.edu/stereo/data/
Calculer pour une paire la carte de disparité en utilisant un algorithme de mise en correspondance.