**CV Practical No.: 11**

**Aim: Camera Calibration**

New Concept:

i. glob: Finds all the file paths matching a pattern (e.g., all .jpg files in a folder). Often used to load multiple calibration images.

ii. cv2.TERM\_CRITERIA\_EPS: Part of a termination criteria flag used for iterative algorithms. EPS (epsilon) means stop when the accuracy goal is met.

cv2.TERM\_CRITERIA\_MAX\_ITER: Another flag for termination criteria. MAX\_ITER means stop when the maximum number of iterations is reached.

np.zeros: Creates an array filled with zeros. Often used to initialize 3D object points.

np.mgrid: Generates a meshgrid of coordinates. Used to create the (x, y) positions of chessboard corners in real-world space (assuming Z=0).

T.reshape(-1, 2): Reshapes the 2D grid into a list of points. -1 tells NumPy to infer the correct number of rows.

image.shape[:2]: Retrieves the height and width of an image (not channels).

cv2.findChessboardCorners(...): Detects the inner corners of a chessboard pattern.

Parameters:

grayColor: Grayscale version of the image.

CHECKERBOARD: Tuple indicating number of inner corners per row and column, e.g., (9,6).

Flags: Improve corner detection.

cv2.CALIB\_CB\_ADAPTIVE\_THRESH: Adaptive thresholding for better contrast.

cv2.CALIB\_CB\_FAST\_CHECK: Speeds up detection (might fail for poor-quality images).

cv2.CALIB\_CB\_NORMALIZE\_IMAGE: Normalizes lighting.

threedpoints.append(objectp3d): Stores the known 3D world coordinates of the chessboard corners. Used for camera calibration.

cv2.cornerSubPix(...): Refines the corner positions to sub-pixel accuracy.

Parameters:

grayColor: Input image.

corners: Initial rough corner positions.

(11,11): Window size.

(-1,-1): Zero zone (no dead zone around center).

criteria: Termination criteria as discussed above.

twodpoints.append(corners2): Stores the detected 2D image coordinates of the chessboard corners. Used in calibration along with 3D points.

cv2.drawChessboardCorners(...): Draws the detected corners on the image for visualization.

Parameters:

image: Image on which to draw.

CHECKERBOARD: Size of the chessboard pattern.

corners2: Refined corner positions.

ret: Boolean indicating if corners were successfully detected.

ret, matrix, distortion, r\_vecs, t\_vecs = cv2.calibrateCamera(...): This function calculates the intrinsic and extrinsic parameters of the camera.

Inputs:

threedpoints: 3D coordinates in the real world.

twodpoints: 2D coordinates in the image.

grayColor.shape[::-1]: Size of the image (width, height) — note the reversal.

None, None: Initial guess for camera matrix and distortion (can be set to None).

Returns:

ret: Re-projection error (lower is better).

matrix: Camera matrix (intrinsics).

distortion: Distortion coefficients (k1, k2, p1, p2, k3) — used to correct barrel or pincushion distortion.

r\_vecs: Rotation vectors (camera pose relative to the pattern).

t\_vecs: Translation vectors (position of camera relative to the pattern).

Theory:

Camera Calibration

Camera Matrix

Distortion Coefficient

Rotation Vector

**Program:**

# Import required modules

import cv2

import numpy as np

import os

import glob

# Define the dimensions of checkerboard

CHECKERBOARD = (6, 9)

# stop the iteration when specified

# accuracy, epsilon, is reached or

# specified number of iterations are completed.

criteria = (cv2.TERM\_CRITERIA\_EPS +

cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001)

# Vector for 3D points

threedpoints = []

# Vector for 2D points

twodpoints = []

# 3D points real world coordinates

objectp3d = np.zeros((1, CHECKERBOARD[0]

\* CHECKERBOARD[1],

3), np.float32)

objectp3d[0, :, :2] = np.mgrid[0:CHECKERBOARD[0],

0:CHECKERBOARD[1]].T.reshape(-1, 2)

prev\_img\_shape = None

# Extracting path of individual image stored

# in a given directory. Since no path is

# specified, it will take current directory

# jpg files alone

images = glob.glob('\*.jpg')

for filename in images:

image = cv2.imread(filename)

grayColor = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Find the chess board corners

# If desired number of corners are

# found in the image then ret = true

ret, corners = cv2.findChessboardCorners(

grayColor, CHECKERBOARD,

cv2.CALIB\_CB\_ADAPTIVE\_THRESH

+ cv2.CALIB\_CB\_FAST\_CHECK +

cv2.CALIB\_CB\_NORMALIZE\_IMAGE)

# If desired number of corners can be detected then,

# refine the pixel coordinates and display

# them on the images of checker board

if ret == True:

threedpoints.append(objectp3d)

# Refining pixel coordinates

# for given 2d points.

corners2 = cv2.cornerSubPix(

grayColor, corners, (11, 11), (-1, -1), criteria)

twodpoints.append(corners2)

# Draw and display the corners

image = cv2.drawChessboardCorners(image, CHECKERBOARD, corners2, ret)

cv2.imshow('img', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

h, w = image.shape[:2]

# Perform camera calibration by

# passing the value of above found out 3D points (threedpoints)

# and its corresponding pixel coordinates of the

# detected corners (twodpoints)

ret, matrix, distortion, r\_vecs, t\_vecs = cv2.calibrateCamera(

threedpoints, twodpoints, grayColor.shape[::-1], None, None)

# Displaying required output

print(" Camera matrix:")

print(matrix)

print("\n Distortion coefficient:")

print(distortion)

print("\n Rotation Vectors:")

print(r\_vecs)

Output:





