What is Camera Calibration in Computer Vision?

Camera calibration is a fundamental task in computer vision crucial in various applications such as 3D reconstruction, object tracking, augmented reality, and image analysis. Accurate calibration ensures precise measurements and reliable analysis by correcting distortions and estimating intrinsic and extrinsic camera parameters. This comprehensive guide delves into the principles, techniques, and algorithms of camera calibration. We explore obtaining intrinsic and extrinsic camera parameters, understanding distortion models, conducting calibration patterns, and utilizing calibration software. Whether you are a beginner or an experienced computer vision practitioner, this guide will equip you with the knowledge and skills to perform accurate camera calibration and unlock the full potential of your vision-based applications.

What is Camera Calibration?

A camera is a device that converts the 3D world into a 2D image. A camera plays a very important role in capturing three-dimensional images and storing them in two-dimensional images. To know the mathematics behind it is extremely fascinating. The following equation can represent the camera.

x=PX

Here x denotes 2-D image point, P denotes camera matrix and X denotes 3-D world point.

$$\underbrace{\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}}_{\substack{\text{homogeneous} \\ \text{image} \\ 3 \times 1}} = \underbrace{\begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix}}_{\substack{\text{Camera} \\ \text{matrix} \\ 3 \times 4}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\substack{\text{homogeneous} \\ \text{world point} \\ 4 \times 1}}$$

Figure1 vector representation of x=PX [1]

Camera calibration is frequently used word in image processing or computer vision field. The camera calibration method is intended to identify the geometric characteristics of the image creation process. This is a vital step to perform in many computer vision applications, especially when metric information on the scene is needed. The camera is often categorized on the basis of a set of intrinsic parameters such as skew of the axis, focal length, and main point in these applications, and its orientation is expressed by extrinsic parameters such as rotation and translation. Linear or nonlinear algorithms are used to estimate intrinsic and extrinsic parameters utilizing known points in real-time and their projections in the picture plane.

Types of Camera Calibration

Camera calibration is the process of determining specific camera parameters in order to complete operations with specified performance measurements.

Camera calibration can be defined as the technique of estimating the characteristics of a camera. It means that we have all of the camera's information like parameters or coefficients which are needed to determine an accurate relationship between a 3D point in the real world and its corresponding 2D projection in the image acquired by that calibrated camera.

In most cases, this entails recovering two types of parameters.

1. Intrinsic or Internal Parameters

It allows mapping between pixel coordinates and camera coordinates in the image frame. E.g. optical center, focal length, and radial distortion coefficients of the lens.

2. Extrinsic or External Parameters

It describes the orientation and location of the camera. This refers to the rotation and translation of the camera with respect to some world coordinate system.

Camera Calibration Method

In this section, we will look at a simple calibrating procedure. The important part is to get the right focal length because most of the parameters can be set using simple assumptions like square straight pixels, optical center at the middle of the image. A flat rectangular calibration object for example a book will suffice, measuring tape or a ruler, and a flat surface are required for this calibration method. The following steps can be performed to have camera calibration.

- Take a measurement of the length and width of your rectangle calibration object. Let us refer to these as dX and dY.
- Place the camera and calibration object on a flat surface with the camera back and calibration object parallel and the object roughly in the center of the camera's vision. To acquire a good alignment, you may need to lift the camera or object.
- Calculate the distance between the camera and the calibration object. Let's call it dZ.
- Take a picture to ensure that the setup is straight, which means that the sides of the calibration object align with the image's rows and columns.
- In pixels, determine the width and height of the object. Let us refer to these as dx and dy.

Focal Length

$$f_x = \frac{dx}{dX} dZ, \qquad f_y = \frac{dy}{dY} dZ.$$

Now let us see the following Figure-3 for set up.

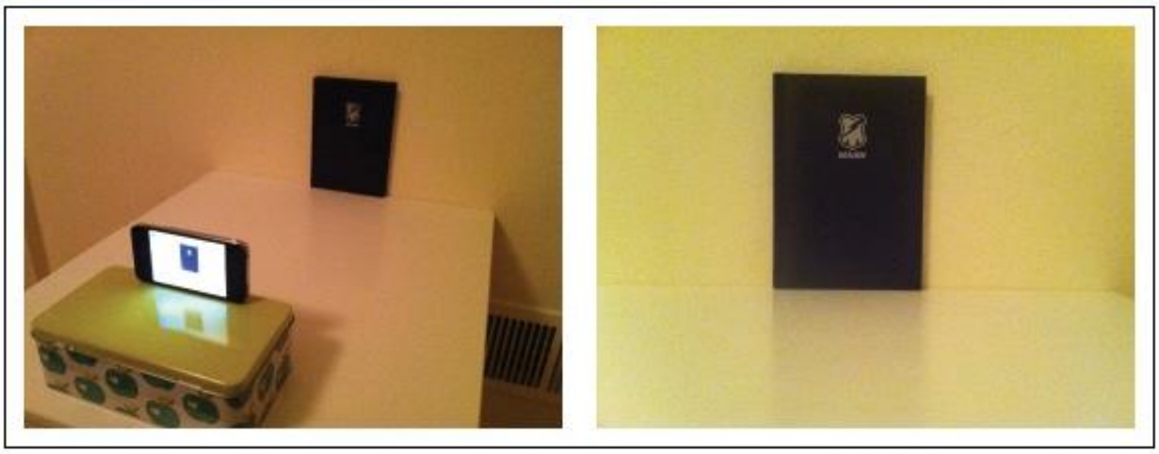


Figure 3: Normal camera calibration setup. The left image shows an image of the setup; the right side image displays calibration. The focal length can be determined by measuring the width and height of the calibration object in the image, as well as the physical measurements of the setup.

For the exact setup in Figure 4-3, the object was measured to be 130 by 185 mm, hence dX = 130 and dY = 185. The camera's distance from the object was 460 mm, hence dZ = 460. Only the ratios of the measurements matter; any unit of measurement can be utilized. After using ginput() to choose four points in the image, the width and height in pixels were 722 and 1040, respectively. In other words, dx = 722 and dy = 1040. When these numbers are used in the aforementioned relationship, the result is

fx equals 2555, and fy equals 2586.

It is critical to note that this is only applicable to a specific image resolution. In this situation, the image was 2592 1936 pixels in size. It should be noted that focal length and optical center are mainly measured in pixels and scale with image resolution. If you choose a different image resolution, the values will change (for example, a thumbnail image). It's a good idea to add your camera's variables to a helper function like this:

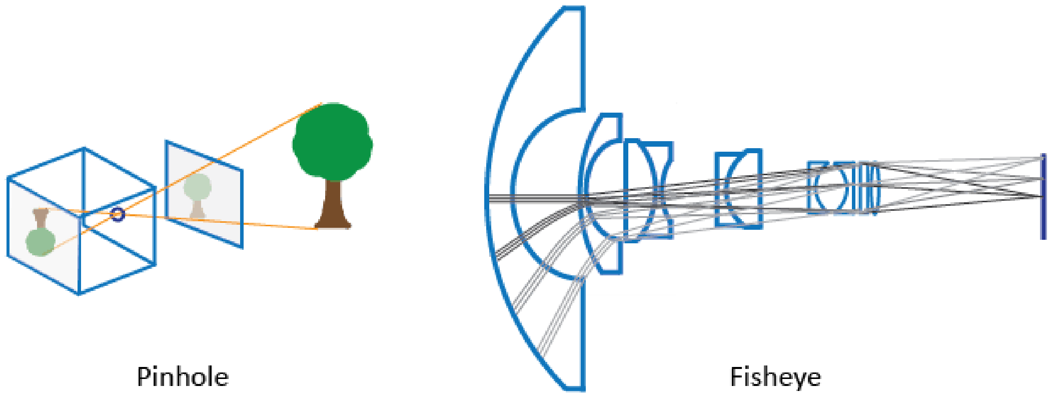
```
def my_calibration(sz):
  row,col = sz
  fx = 2555*col/2592
  fy = 2586*row/1936
  K = diag([fx,fy,1])
```

```
K[0,2] = 0.5*col
K[1,2] = 0.5*row
return K
```

After that, this function takes a size tuple and returns the calibration matrix. The optical center is assumed to be the image's center in this case. Replace the focal lengths with their mean if you like; for most consumer cameras, this is fine. It should be noted that the calibration is only for photographs in landscape orientation.

Camera Calibration Models

Calibration techniques for the pinhole camera model and the fisheye camera model are included in the Computer Vision ToolboxTM. The fisheye variant is compatible with cameras with a field of vision (FOV) of up to 195 degrees.

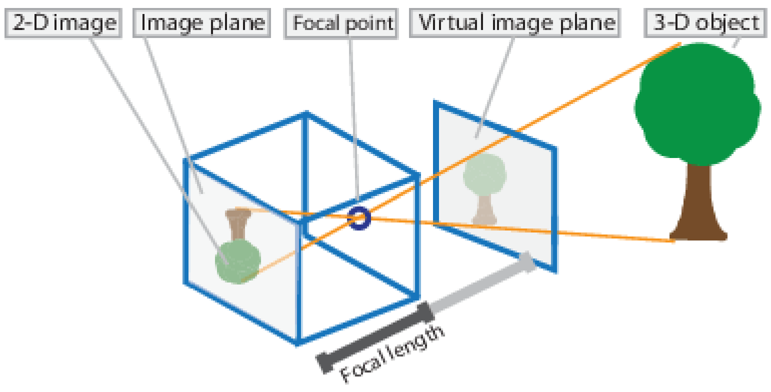


The pinhole calibration algorithm is based on Jean-Yves Bouguet's [3] model. The pinhole camera model and lens distortion are included in the model. Because an ideal pinhole camera does not have a lens, the pinhole camera model does not account for lens distortion. To accurately simulate a genuine camera, the algorithm's whole camera model incorporates radial and tangential lens distortion.

The pinhole model cannot model a fisheye camera due to the high distortion produced by a fisheye lens.

Pinhole Camera Model

A pinhole camera is a basic camera model without a lens. Light rays pass through the aperture and project an inverted image on the opposite side of the camera. Visualize the virtual image plane in front of the camera and assume that it is containing the upright image of the scene.

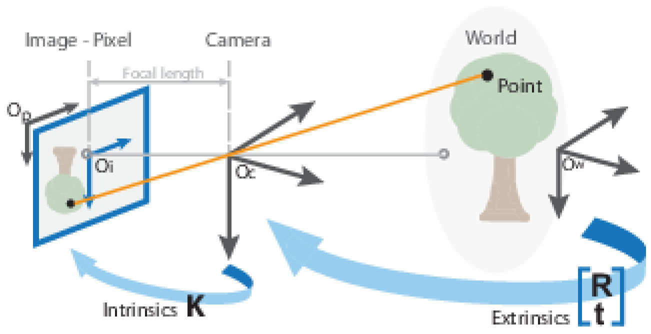


The camera matrix is a 4-by-3 matrix that represents the pinhole camera specifications. The image plane is mapped into the image plane by this matrix, which maps the 3-D world scene. Using the extrinsic and intrinsic parameters, the calibration algorithm computes the camera matrix. The extrinsic parameters represent the camera's position in the 3-D scene. The intrinsic characteristics represent the camera's optical center and focal length.

$$\underset{\text{Scale factor}}{w}\,[\underbrace{x\ y\ 1}_{\text{Image points}}] = [\underbrace{X\ Y\ Z\ 1}_{\text{World points}}]\,P$$

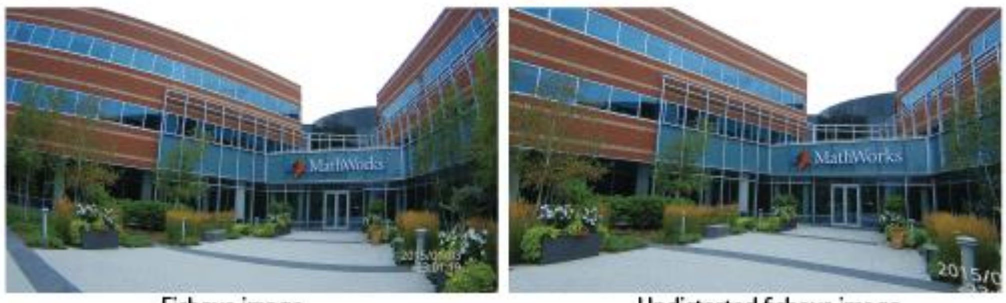
$$\underset{\text{Camera matrix}}{P} = \underset{\substack{\text{Extrinsics}\\ \text{Rotation and translation}}}{\begin{bmatrix} R \\ t \end{bmatrix}}\,\underset{\text{Intrinsic matrix}}{K}$$

The world points are transformed to camera coordinates using the extrinsic parameters. Intrinsic parameters are used to map the camera coordinates into the image plane.



Fisheye Camera Model

Camera calibration is the process of calculating the extrinsic and intrinsic properties of a camera. After calibrating a camera, the picture information can be utilized to extract 3-D information from 2-D photographs. Images taken with a fisheye camera can also be distortion-free. In Matlab, there is the Computer Vision Toolbox which includes calibration procedures for the pinhole camera model and the fisheye camera model. The fisheye variation works with cameras that have a field of view (FOV) of up to 195 degrees.

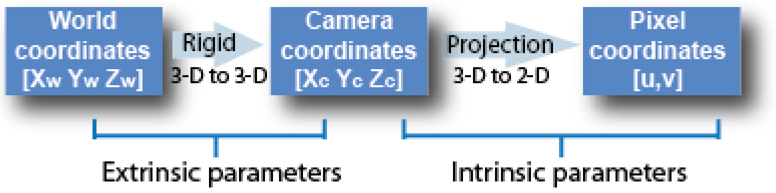


Fisheye image Undistorted fisheye image

As a fisheye lens produces extreme distortion, the pinhole model cannot model a fisheye camera.

Fisheye cameras are employed in odometry as well as to visually solve simultaneous localization and mapping (SLAM) difficulties. Surveillance systems, GoPro, virtual reality (VR) to capture a 360-degree field of view (fov), and stitching algorithms are examples of other applications. These cameras employ a complicated array of lenses to increase the camera's field of view, allowing it to capture wide panoramic or hemispherical images. The lenses achieve this extraordinarily wide-angle view, however, by distorting the perspective lines in the image.

Scaramuzza's fisheye camera model is used by the Computer Vision Toolbox calibration algorithm. An omnidirectional camera model is used in the model. The imaging system is treated as a compact system in the procedure. You must collect the camera's extrinsic and intrinsic parameters in order to link a 3-D world point to a 2-D image. The extrinsic parameters are used to translate world points to camera coordinates. The intrinsic parameters are used to transfer the camera coordinates onto the picture plane.



Types of distortion effects and their cause

We obtain better photos when we use a lens, yet the lens produces some distortion effects. Distortion effects are classified into two types:

Radial Distortion

This sort of distortion is caused by unequal light bending. The rays bend more at the lens's borders than they do at the lens's center. Straight lines in the actual world appear curved in the image due to radial distortion. Before hitting the image sensor, the light ray is shifted radially inward or outward from its optimal point. The radial distortion effect is classified into two types.

1. Effect of barrel distortion, which corresponds to negative radial displacement
2. The pincushion distortion effect results in a positive radial displacement.

Tangential Distortion

It occurs when the picture screen or sensor is at an angle with respect to the lens. As a result, the image appears to be slanted and stretched.

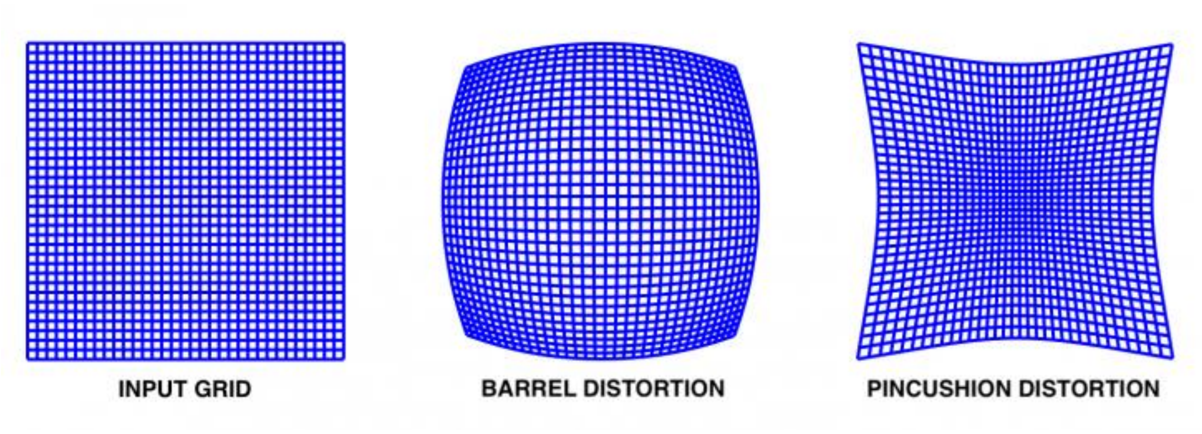


Figure: Example of the effect of barrel and pincushion distortion on a square grid.

There are three types of distortion depending on the source: radial distortion, decentering distortion, and thin prism distortion. Decentering and narrow prism distortion both cause radial and tangential distortion.

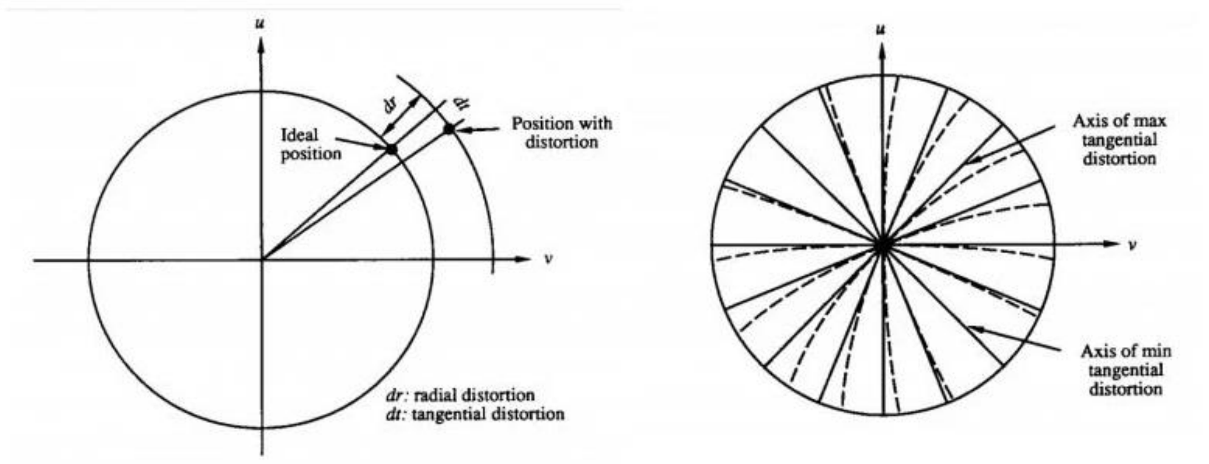

u
v
dr
dt
Ideal position
Position with distortion
dr: radial distortion
dt: tangential distortion
u
v
Axis of max tangential distortion
Axis of min tangential distortion

Figure: Diagrams illustrating the effect of tangential distortion, with solid lines representing no distortion and dotted lines representing tangential distortion (right) and (left) how tangential and radial distortion shifts a pixel from its ideal position.

We now have a better understanding of the different sorts of distortion effects generated by lenses, but what does a distorted image look like? Is it necessary to be concerned about the lens's distortion? If so, why? How are we going to deal with it?

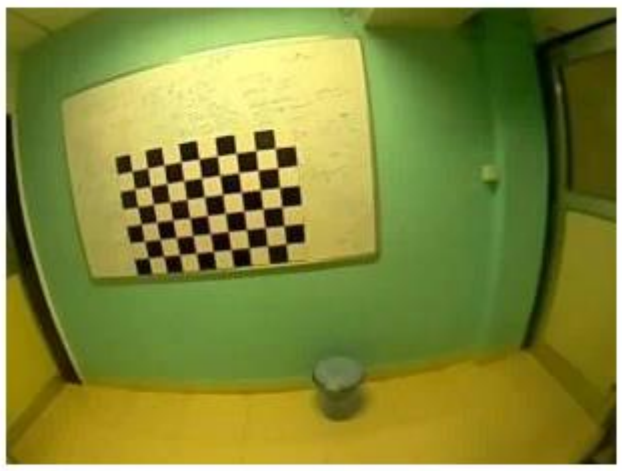


Figure: Illustration of the distortion effect. Take note of how the margins of the wall and doors are curled as a result of distortion.

The image above is an example of the distortion effect that a lens can produce. The figure corresponds to figure 1 and is a barrel distortion effect, which is a form of the radial distortion effect. Which two points would you consider if you were asked to find the correct door height? Things get considerably more challenging when executing SLAM or developing an augmented reality application using cameras that have a large distortion effect in the image.

Mathematically Representing Lens Distortion

When attempting to estimate the 3D points of the real world from an image, we must account for distortion effects.

Based on the lens parameters, we mathematically analyze the distortion effect and combine it with the pinhole camera model described in the previous piece in this series. As additional intrinsic parameters, we have distortion coefficients (which quantitatively indicate lens distortion), in addition to the intrinsic and extrinsic characteristics discussed in the preceding post.

To account for these distortions in our camera model, we make the following changes to the pinhole camera model:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [R|T] \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \tag{1}$$

$$x' = \frac{X_c}{Z_c} \tag{2}$$

$$y' = y_c/Z_c \tag{3}$$

$$\gamma = (1+ \mathrm{K}_1 r^2 + K_2 r^4 + K_3 r^6)/(1 + K_4 r^2 + K_5 r^4 + K_6 r^6) \tag{4}$$

$$r^2 = x'^2 + y'^2 \tag{5}$$

$$x" = x'(\gamma) + 2\mathrm{P}_1 x' y' + P_2(r^2 + 2x'^2) \tag{6}$$

$$y" = y'(\gamma) + \mathrm{P}_1(r^2 + 2y'^2) + 2P_2 x' y' \tag{7}$$

$$s * \begin{bmatrix} x_{img} \\ y_{img} \\ 1 \end{bmatrix} = \begin{bmatrix} -1/m_x & 0 & c_x \\ 0 & -1/m_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x" \\ y" \\ 1 \end{bmatrix} \tag{8}$$

The calibrateCamera method returns the distCoeffs matrix, which contains values for K1 through K6, which represent radial distortion, and P1 and P2, which represent tangential distortion. Because the following mathematical model of lens distortion covers all sorts of distortions, radial distortion, decentering distortion, and thin prism distortion, the coefficients K1 through K6 represent net radial distortion while P1 and P2 represent net tangential distortion.

Removing Distortion

So what do we do after the calibration step? We got the camera matrix and distortion coefficients in the previous post on camera calibration but how do we use the values?

One application is to use the derived distortion coefficients to un-distort the image. The images shown below depict the effect of lens distortion and how it can be removed from the coefficients obtained from camera calibration.

How to Reduce Lens Distortion?

To reduce lens distortion, three fundamental actions must be taken.

1. Calibrate the camera and obtain the intrinsic camera parameters. This is exactly what we accomplished in the previous installment of this series. The camera distortion characteristics are also included in the intrinsic parameters.
2. Control the percentage of undesired pixels in the undistorted image by fine-tuning the camera matrix.
3. Using the revised camera matrix to remove distortion from the image.

The getOptimalNewCameraMatrix() method is used in the second phase. What exactly does this refined matrix imply, and why do we require it? Refer to the photographs below; in the right image, we can see several black pixels at the boundaries. These are caused by the image's undistortion. These dark pixels are sometimes undesirable in the final undistorted image. Thus, the getOptimalNewCameraMatrix() method gives a refined camera matrix as well as the ROI (region of interest), which may be used to crop the image to exclude all black pixels. The percentage of undesirable pixels to be removed is determined by the alpha parameter, which is supplied as an input to the getOptimalNewCameraMatrix() method.

Python Code for Camera Calibration

```
import cv2
 import numpy as np
 import os
 import glob
 # Defining the dimensions of checkerboard
 CHECKERBOARD = (6,9)
 criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
 # Creating vector to store vectors of 3D points for each checkerboard image
 objpoints = []
 # Creating vector to store vectors of 2D points for each checkerboard image
 imgpoints = []
 # Defining the world coordinates for 3D points
 objp = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.float32)
 objp[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
 images = glob.glob('./images/*.jpg')
 for fname in images:
 img = cv2.imread(fname)
 gray = cv2.cvtColor(img,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(gray, CHECKERBOARD,
 cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK +
cv2.CALIB_CB_NORMALIZE_IMAGE)
 """
 If desired number of corner are detected,
 we refine the pixel coordinates and display
 them on the images of checker board
 """
 if ret == True:
 objpoints.append(objp)
 # refining pixel coordinates for given 2d points.
 corners2 = cv2.cornerSubPix(gray, corners, (11,11),(-1,-1), criteria)
 imgpoints.append(corners2)
 # Draw and display the corners
 img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2, ret)
 cv2.imshow('img',img)
 cv2.waitKey(0)
 cv2.destroyAllWindows()
 h,w = img.shape[:2]
 """
 Performing camera calibration by
 passing the value of known 3D points (objpoints)
 and corresponding pixel coordinates of the
 detected corners (imgpoints)
 """
 ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],
None, None)
 print("Camera matrix : n")
 print(mtx)
 print("dist : n")
 print(dist)
 print("rvecs : n")
 print(rvecs)
 print("tvecs : n")
 print(tvecs)
```

Reference

1. http://www.cs.cmu.edu/~16385/s17/Slides/11.1_Camera_matrix.pdf
2.https://www.sciencedirect.com/topics/computer-science/camera-calibration
3.https://learnopencv.com/understanding-lens-distortion
4.https://in.mathworks.com/help/vision/ug/camera-calibration.html