جامعة الأمير مقرن بن عبد العزيز
Prince Mugrin University

Lab 5

Camera calibration with OpenCV

Learning Objectives:

- *To use OpenCV for camera calibration*
- *Detect chessboard corners for calibration.*
- *Compute intrinsic parameters and distortion coefficients.*
- *Apply image undistortion to remove lens distortion.*
- *Use calibration for real-world applications like AR and robotics.*

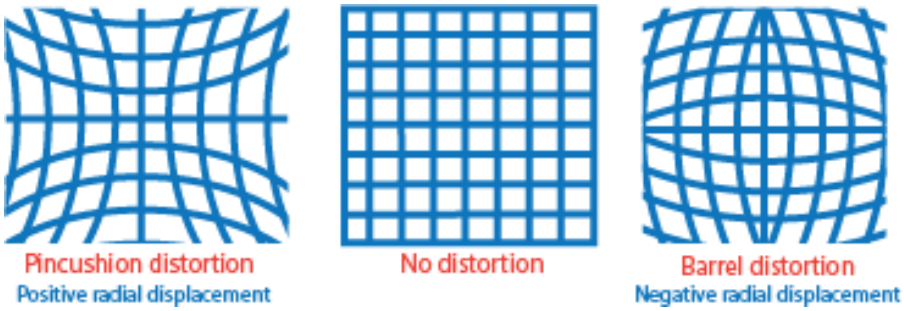
Introduction:

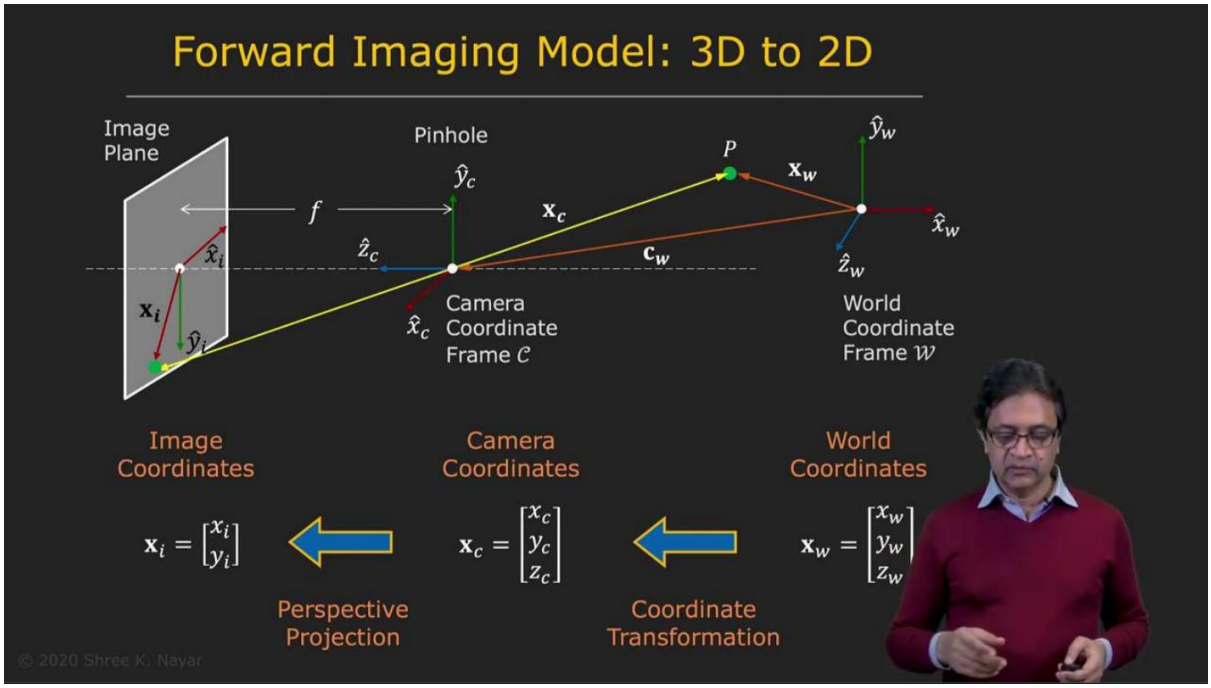
Camera calibration is an essential process in computer vision that allows us to correct distortions caused by a camera lens and accurately map real-world points onto an image. Calibration estimates the intrinsic and extrinsic parameters that define how a camera perceives the environment. This process is crucial for applications such as augmented reality (AR), 3D reconstruction, autonomous navigation, and object tracking.

Generally when we have a scene and we want to project it into an image 3D to 2D we need to find the projection matrix which include the intrinsic and extrinsic parameters , finding this matrix is camera calibration

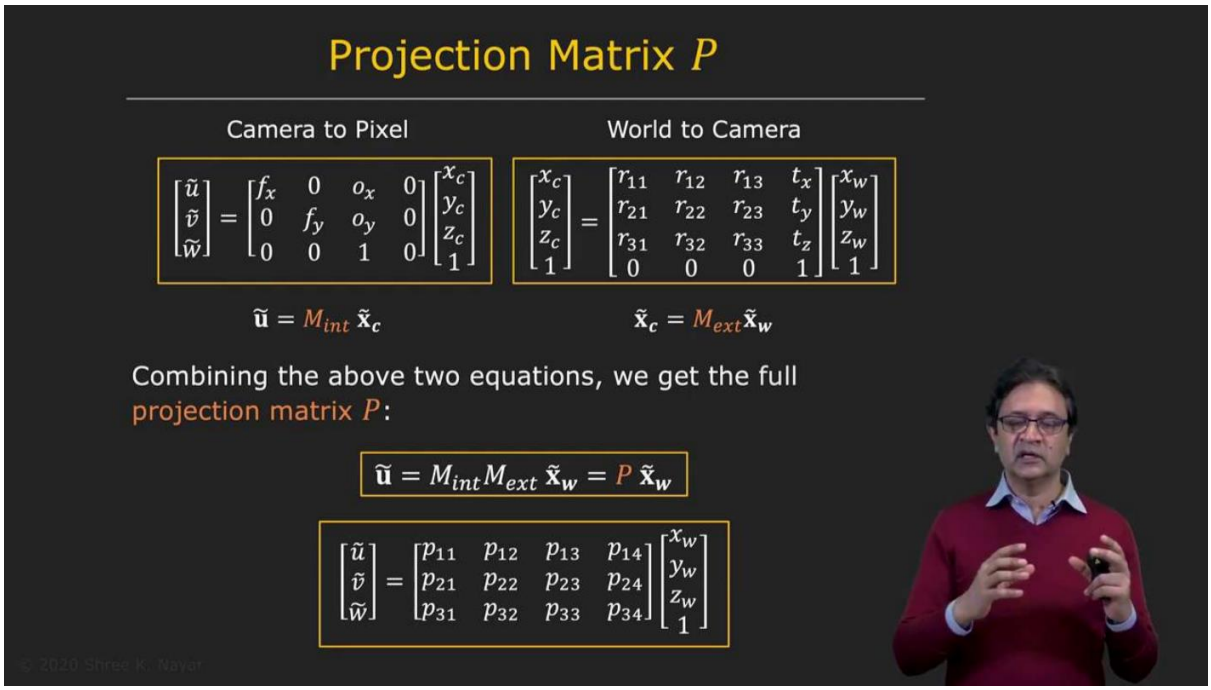
Why Do We Need Camera Calibration:

When an image is captured using a camera, distortions cause straight lines to appear curved, affecting depth estimation and object localization. Calibration corrects these distortions, ensuring **accurate mapping of 3D world coordinates onto a 2D image plane**.





How 3D scene become a 2D image



The final form of the projection Matrix

Camera Model: Intrinsic and Extrinsic Parameters:

A camera model consists of:

1. **Intrinsic Parameters** (Internal camera properties):
    - **Focal length (fx, fy)**: Defines how much the camera magnifies the scene.
    - **Principal point (cx, cy)**: The image center where the optical axis meets the image plane.
    - **Distortion coefficients (k1, k2, p1, p2, k3)**: Corrects radial and tangential distortions.
2. **Extrinsic Parameters** (Defines camera position and orientation):
    - **Rotation matrix (R)**: Describes how the camera is rotated.
    - **Translation vector (T)**: Defines the camera's position in space.

Mathematical Model: The relationship between 3D world coordinates and 2D image points is:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where K is the intrinsic matrix, and [R | T] represents the extrinsic parameters.

Steps of Camera Calibration:

📌 Step 1: Capture Chessboard Images

- **Why?** A checkerboard pattern provides well-defined corners, which help estimate distortion parameters.
- The images used for calibration must be taken from different angles to ensure accurate parameter estimation.

📌 Step 2: Detect Chessboard Corners

- **Why?** Detecting corner points provides a mapping between the real-world 3D structure and the 2D image.
- This function detects the corners required for calibration.

```
ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)
```

📌 Step 3: Store Object Points and Image Points

- **Why?** Object points represent real-world 3D coordinates, and image points represent their corresponding 2D projections**.**

```
objpoints.append(objp)
imgpoints.append(corners)
```

- **These lists store real-world and detected points for calibration.**

📌 Step 4: Compute Camera Calibration Parameters

- **Why?** Computes intrinsic parameters (camera matrix) and distortion coefficients.

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, gray.shape[::-1], None, None)
```

- mtx: Intrinsic matrix (focal length, principal point).
- dist: Distortion coefficients (radial and tangential distortion).
- rvecs:
- tvecs:

📌 Step 5: Apply Calibration to Remove Distortion

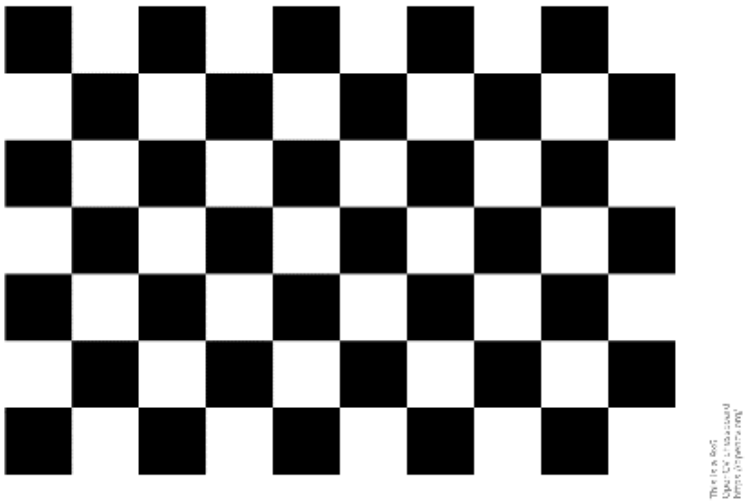
- **Why?** Corrects lens distortion and produces an undistorted image.

```
undistorted_img = cv2.undistort(img, mtx, dist, None, new_camera_mtx)
```

- cv2.undistort() removes distortions using the calibration parameters.

Lets do step 1 and 2

Task: Find Chessboard Corners for Calibration



```python
import cv2
import numpy as np

# Load a checkerboard image
img = cv2.imread('checkerboard.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Define chessboard size (columns, rows)
checkerboard_size = (6, 9)

# Find chessboard corners
ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

# Draw detected corners
if ret:
    img = cv2.drawChessboardCorners(img, checkerboard_size, corners, ret)
    cv2.imshow('Detected Corners', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

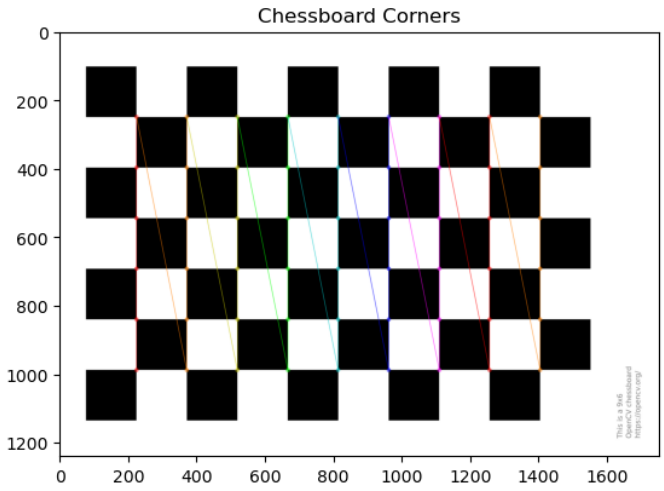
Lets do step 3 and 4

Task: **Compute Camera Parameters**

- **Step 1:** Different viewpoints of check-board image is captured.



- **Step 2:** *Find Chessboard Corners for each image*



- **Step 3:** Store Object Points and Image Points
- **Step 4:** given all previous information we can calculate the camera parameters using *calibrateCamera() function*
- **Step 5:** print the extrinsic and intrinsic parameters

Code Example:

```
import cv2
import numpy as np
import glob
import os

# Define the chessboard size (columns, rows)
checkerboard_size = (6, 9)

# Prepare object points
objp = np.zeros((checkerboard_size[0] * checkerboard_size[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:checkerboard_size[0],
0:checkerboard_size[1]].T.reshape(-1, 2)

# Store object points and image points
objpoints = []  # Real world 3D points
imgpoints = []  # 2D points in image plane

# Define the path to the folder containing calibration images
image_folder = r'C:\Users\m.nasif\Desktop\CV\Lab 5\Images'


# Get all image file paths in the folder
image_files = glob.glob(os.path.join(image_folder, '*.jpg'))  # Change '*.jpg'
to '*.png' if needed
# Process each image
for fname in image_files:
    img = cv2.imread(fname)

    if img is None:
        print(f"Warning: Could not read {fname}")
        continue

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

    if ret:
        objpoints.append(objp)
        imgpoints.append(corners)


# Perform camera calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
# Print the camera matrix and distortion coefficients
print("Camera Matrix:\n", mtx)
print("Distortion Coefficients:\n", dist)
```

Exercise 1: Read image pattern from camera or folder and detect the pattern corners.

Exercise 2: Perform camera calibration using webcam and print the calibration parameters.

Bonus Task: Augmented Reality (AR) with Calibration

🔷 **Objective:**

Use camera calibration data to overlay a **virtual object** in a real-world scene.

🛠️ **Task:**

1. **Detect a chessboard pattern in real-time video.**
2. **Correct distortions** using the camera matrix.
3. **Overlay a 3D virtual object (cube or marker).**

Reference:

- Open CV camera calibration
- Video Tutorials for the theoretical part