Computer Vision
Assignment-01

Camera Calibration using OpenCV

Deep Prajapati, 21CS01048

S.S.V. Raghava, 21CS01068

Indian Institute of Technology Bhubaneswar

School of Electrical and Computer Science

Overview of Camera Parameters

Intrinsic Parameters

Characteristics specific to camera lens and sensor, including:

- Focal length (f_x and f_y)
- Principle point (c_x and c_y) - the optical centre of the image
- Skew parameter - measures non-orthogonality between an image plane's x-axis and y-axis.

Extrinsic Parameters

Transformation matrices mapping world coordinates to the camera's coordinate system, including:

- Rotation matrix: Indicates how the camera is oriented in the real world.
- Translation matrix: Indicates the position of the camera relative to the world

$$s \underbrace{\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}}_{\substack{\text{2D Image} \\ \text{Coordinates}}} = \underbrace{\begin{bmatrix} f_x & S & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\substack{\text{Intrinsic properties} \\ \text{(Optical Centre, scaling)}}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{\substack{\text{Extrinsic properties} \\ \text{(Camera Rotation} \\ \text{and translation)}}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\substack{\text{3D World} \\ \text{Coordinates}}}$$

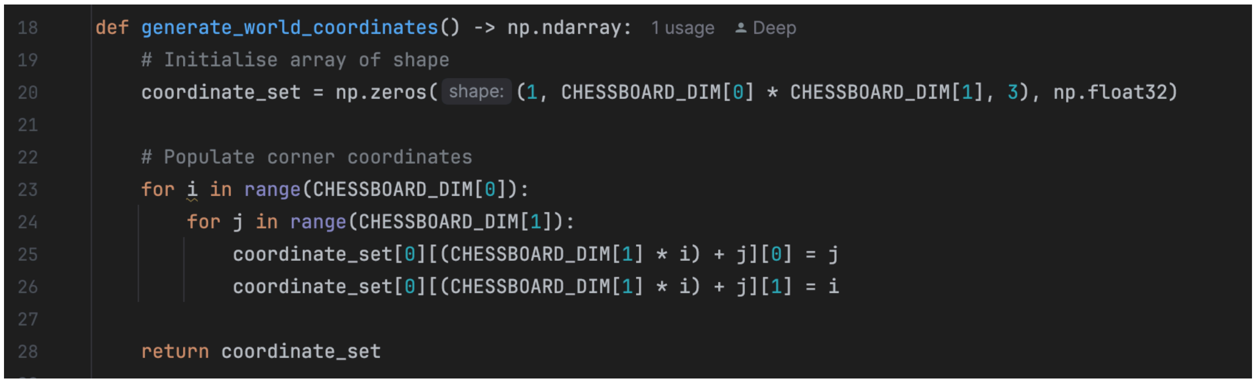
Steps to Estimate Parameters

Step-01: Generate corresponding points

The calibration process relies on the relationship between world coordinates (3D points on the chessboard corners) and image coordinates (2D-pixel location of the same corners as located in the image)

Generating world coordinates:

First, we initialise the 3D grid of points (x, y, z) corresponding to the chessboard's corners in the real world. We assume the chessboard lies on a flat plane (z=0).

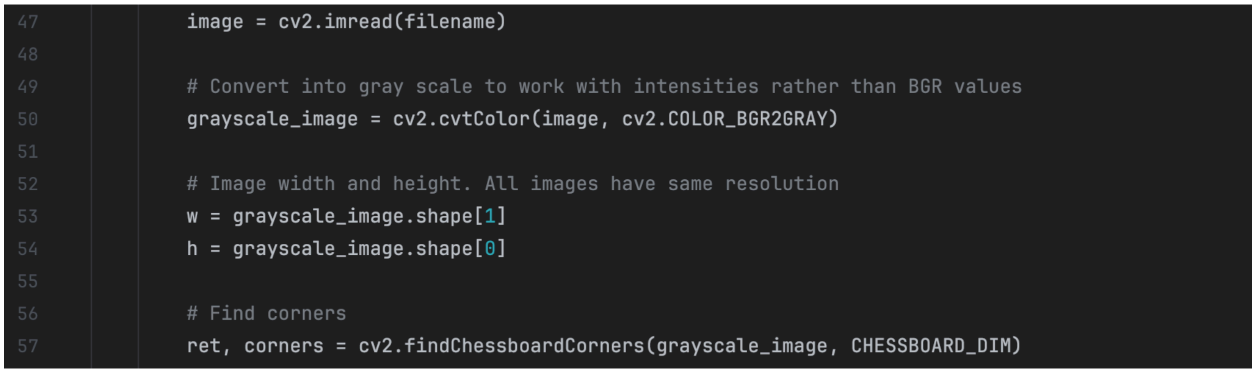


```python
def generate_world_coordinates() -> np.ndarray:
    # Initialise array of shape
    coordinate_set = np.zeros((1, CHESSBOARD_DIM[0] * CHESSBOARD_DIM[1], 3), np.float32)

    # Populate corner coordinates
    for i in range(CHESSBOARD_DIM[0]):
        for j in range(CHESSBOARD_DIM[1]):
            coordinate_set[0][(CHESSBOARD_DIM[1] * i) + j][0] = j
            coordinate_set[0][(CHESSBOARD_DIM[1] * i) + j][1] = i

    return coordinate_set
```

The (x, y) coordinates are incrementally populated for each corner, forming the 3D chessboard pattern in the world coordinate system.

Detecting image coordinates:

The coordinates of the same points as seen by the camera are detected by first converting the image to Grayscale (working on intensities rather than colour) and then detecting corners using *cv2.findChessboardCorners()*.

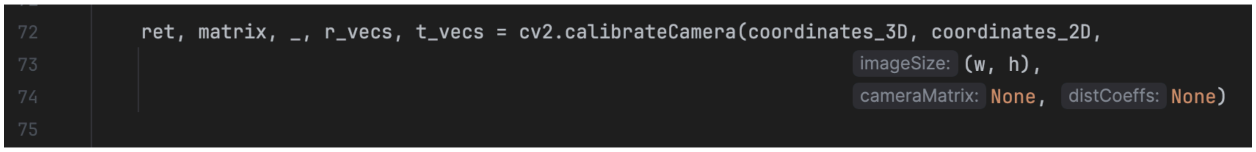


```python
        image = cv2.imread(filename)

        # Convert into gray scale to work with intensities rather than BGR values
        grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Image width and height. All images have same resolution
        w = grayscale_image.shape[1]
        h = grayscale_image.shape[0]

        # Find corners
        ret, corners = cv2.findChessboardCorners(grayscale_image, CHESSBOARD_DIM)
```

If corners are detected, those are further refined to the sub-pixel range using *cv2.cornerSubPix()*.

Step-02: Calibrate the camera

Once the world and image coordinates of all the images are determined, pass these to *cv2.calibrateCamera()*.



```
ret, matrix, _, r_vecs, t_vecs = cv2.calibrateCamera(coordinates_3D, coordinates_2D,
                                                     imageSize: (w, h),
                                                     cameraMatrix: None, distCoeffs: None)
```

Inputs:

1. coordinates_3D: 3D chessboard corner points according to the world coordinate system.
2. coordinates_2D: 2D image coordinates of chessboard corners.

Outputs:

1. matrix: The intrinsic parameter matrix.
2. r_vecs and t_vecs: The extrinsic parameters for each image.

Step-03: Extract results

Intrinsic parameters:

- Focal lengths are extracted from the diagonal elements of the matrix.

f_x = matrix[0, 0]

f_y = matrix[1, 1]

- Principle points are extracted from the third column of the matrix.

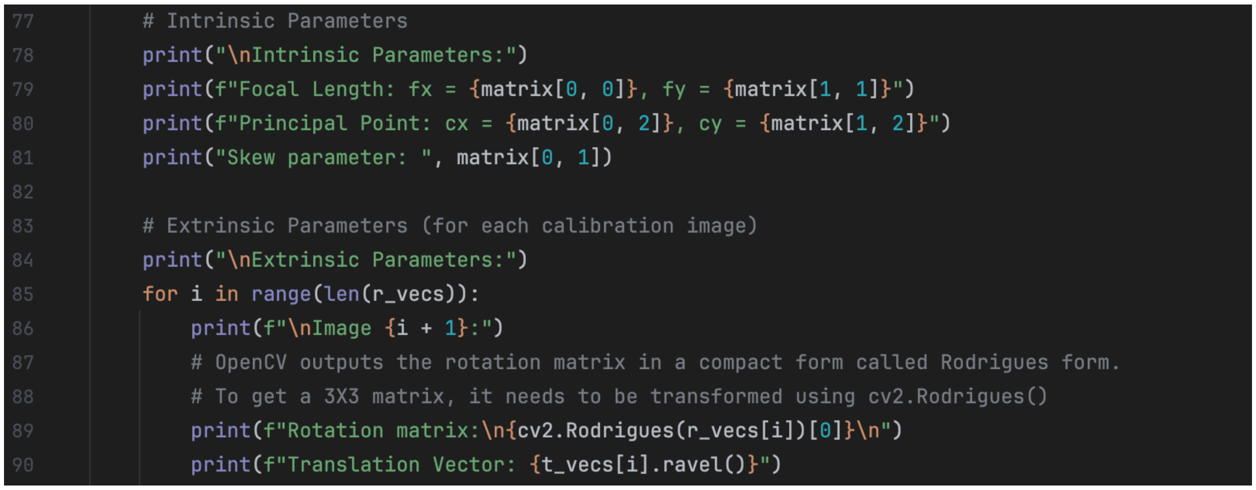
c_x = matrix[0, 2]

c_y = matrix[1, 2]

- Skew parameter, S = matrix[0, 1]

Extrinsic parameters:

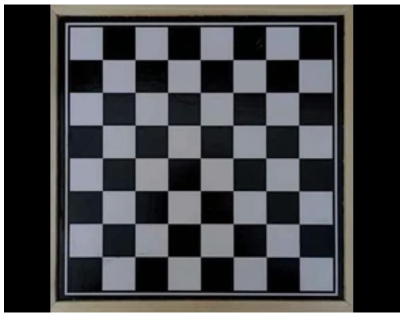
- The rotation vectors (r_vecs) and translation vectors (t_vecs) contain extrinsic parameters for each image.
- By default, the rotation matrix is represented compactly (Rodrigues form). It can be converted into a full matrix using *cv2.Rodrigues()*.

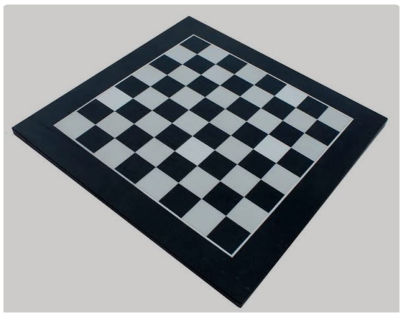


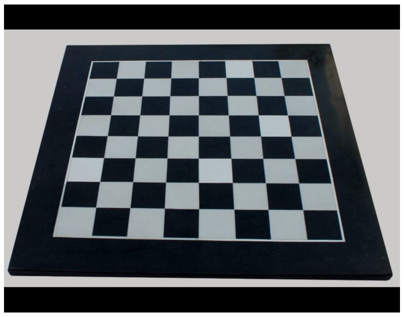
```python
# Intrinsic Parameters
print("\nIntrinsic Parameters:")
print(f"Focal Length: fx = {matrix[0, 0]}, fy = {matrix[1, 1]}")
print(f"Principal Point: cx = {matrix[0, 2]}, cy = {matrix[1, 2]}")
print("Skew parameter: ", matrix[0, 1])

# Extrinsic Parameters (for each calibration image)
print("\nExtrinsic Parameters:")
for i in range(len(r_vecs)):
    print(f"\nImage {i + 1}:")
    # OpenCV outputs the rotation matrix in a compact form called Rodrigues form.
    # To get a 3X3 matrix, it needs to be transformed using cv2.Rodrigues()
    print(f"Rotation matrix:\n{cv2.Rodrigues(r_vecs[i])[0]}\n")
    print(f"Translation Vector: {t_vecs[i].ravel()}")
```

Example:

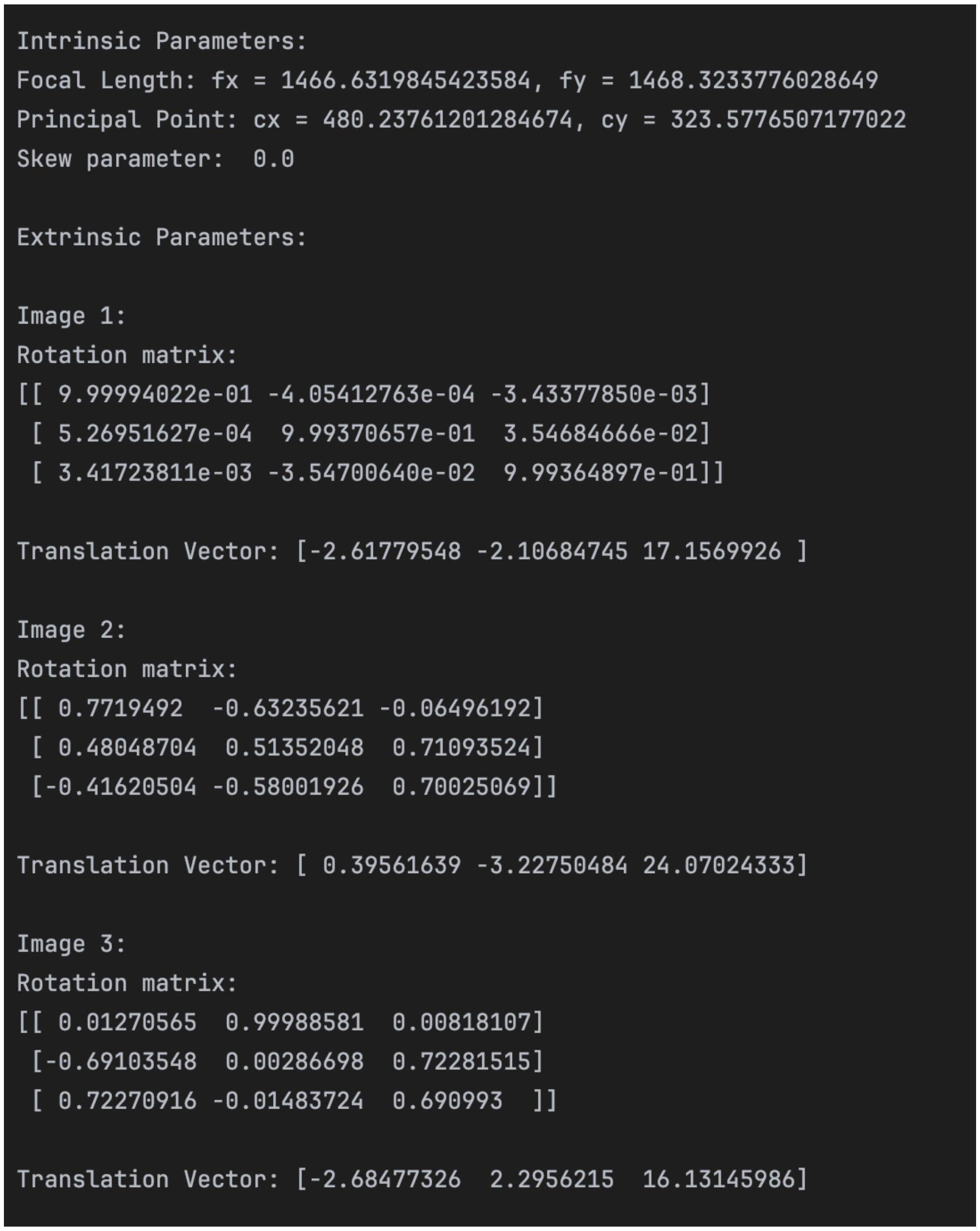
Input:







Output:



```
Intrinsic Parameters:
Focal Length: fx = 1466.6319845423584, fy = 1468.3233776028649
Principal Point: cx = 480.23761201284674, cy = 323.5776507177022
Skew parameter:  0.0

Extrinsic Parameters:

Image 1:
Rotation matrix:
[[ 9.99994022e-01 -4.05412763e-04 -3.43377850e-03]
 [ 5.26951627e-04  9.99370657e-01  3.54684666e-02]
 [ 3.41723811e-03 -3.54700640e-02  9.99364897e-01]]

Translation Vector: [-2.61779548 -2.10684745 17.1569926 ]

Image 2:
Rotation matrix:
[[ 0.7719492  -0.63235621 -0.06496192]
 [ 0.48048704  0.51352048  0.71093524]
 [-0.41620504 -0.58001926  0.70025069]]

Translation Vector: [ 0.39561639 -3.22750484 24.07024333]

Image 3:
Rotation matrix:
[[ 0.01270565  0.99988581  0.00818107]
 [-0.69103548  0.00286698  0.72281515]
 [ 0.72270916 -0.01483724  0.690993  ]]

Translation Vector: [-2.68477326  2.2956215  16.13145986]
```