Project 3

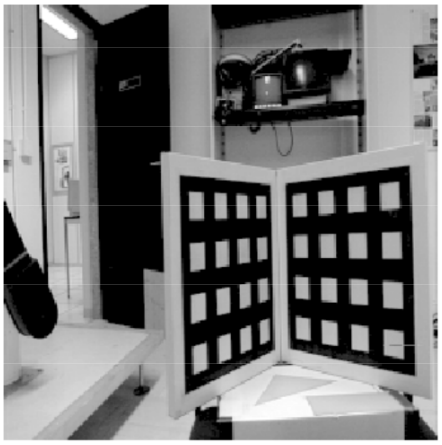
ENPM673

Vishnu Mandala
119452608

Problem 1:

Calibrate the camera (Find the intrinsic matrix K)

For this question, you are NOT allowed to use any in-built function that solves the question for you.



1. **What is the minimum number matching points to solve this mathematically?**
2. **What is the pipeline or the block diagram that needs to be done in order to calibrate this camera given the image above.**
3. **First write down the mathematical formation for your answer including steps that need to be done to find the intrinsic matrix K.**
4. **Find the P matrix.**
5. **Decompose the P matrix into the Translation, Rotation and Intrinsic matrices using the Gram–Schmidt process and compute the reprojection error for each point.**

Note: You are only allowed to use numpy for this question. No marks will be given if you use any other library/tool.

Image Points		World Points		
x	y	X	Y	Z
757	213	0	0	0
758	415	0	3	0
758	686	0	7	0
759	966	0	11	0
1190	172	7	1	0
329	1041	0	11	7
1204	850	7	9	0
340	159	0	1	7

Solution:

Camera calibration is the process of estimating the intrinsic and extrinsic parameters of a camera, which are necessary for mapping 3D world coordinates to 2D image coordinates. In other words, camera calibration helps us to establish a relationship between the world and image coordinate systems.

The intrinsic parameters of a camera define its internal geometry, such as the focal length, image center, and lens distortion characteristics. These parameters are typically constant for a given camera and are often modeled using an intrinsic matrix (K). The extrinsic parameters of a camera define its position and orientation in the world coordinate system and include the camera's translation vector and rotation matrix.

To perform camera calibration, a set of calibration images are captured with known 3D world points. The images are analyzed to extract the image coordinates of the world points, and these image coordinates are then used to estimate the intrinsic and extrinsic parameters of the camera.

There are different methods for camera calibration, but most rely on the use of a calibration object with known geometry. The calibration object is typically a planar grid or a 3D object with well-defined features. By analyzing the image of the calibration object and comparing it to the known geometry, the intrinsic and extrinsic parameters of the camera can be estimated.

Once the camera is calibrated, the intrinsic and extrinsic parameters can be used to project 3D world coordinates onto the 2D image plane. This process is important for a variety of computer vision applications, such as object detection, 3D reconstruction, and augmented reality.

A projection matrix is a mathematical representation of the transformation from 3D world coordinates to 2D image coordinates in a camera. It is a 3x4 matrix that combines the intrinsic and extrinsic parameters of the camera. The projection matrix is denoted as P.
The intrinsic matrix K is a 3x3 matrix that represents the internal geometry of the camera. It is defined with parameters: fx and fy are the focal lengths in the x and y directions, cx and cy are the image centers in the x and y directions and s is the skew coefficient that accounts for any non-orthogonality between the image axes

The rotation matrix R is a 3x3 matrix that represents the orientation of the camera in the world coordinate system. It defines the transformation from the camera coordinate system to the world coordinate system.

The translation vector t is a 3x1 vector that represents the position of the camera in the world coordinate system. It defines the location of the camera center in the world coordinate system.

To project a 3D world point X = [X,Y,Z] onto the 2D image plane, we can use the projection matrix P and the homogeneous coordinate representation of X:

$$X = [X\,Y\,Z\,1]$$

The projection of X onto the image plane is given by:

$$[x\,y\,w] = P[X\,Y\,Z\,1]$$

where, x and y are the image coordinates of the point X
w is a scaling factor that ensures that the image coordinates are in the same units as the intrinsic matrix K.

In summary, the projection matrix combines the intrinsic and extrinsic parameters of a camera into a single matrix that can be used to project 3D world points onto the 2D image plane. The intrinsic matrix K defines the internal geometry of the camera, the rotation matrix R defines the orientation of the camera in the world coordinate system, and the translation vector t defines the position of the camera in the world coordinate system.

1. The minimum number of matching points required to solve for the intrinsic matrix K is 6. The reason why the minimum number of matching points required to solve for the intrinsic matrix K is 6 is due to the degrees of freedom in the camera calibration problem. The intrinsic matrix K has 5 independent parameters (fx, fy, cx, cy, s), plus any additional distortion parameters (such as radial distortion coefficients and tangential distortion coefficients) that may be present.
To solve for the intrinsic matrix K, we need to have at least 5 equations that relate the image coordinates to the world coordinates of the Tsai grid points. Each equation provides one constraint on the values of the intrinsic parameters.

However, there are additional constraints that arise from the geometry of the Tsai grid pattern. Specifically, for each point in the image, we can use the known geometry of the grid lines to obtain a 6th equation that relates the image coordinates to the world coordinates. This 6th equation provides an additional constraint on the intrinsic parameters.

Therefore, the minimum number of matching points required to solve for the intrinsic matrix K is 6. With 6 correspondences between image points and world points, we have 6 equations and 6 unknowns, which can be solved using standard linear algebra techniques.

2. Pipeline:

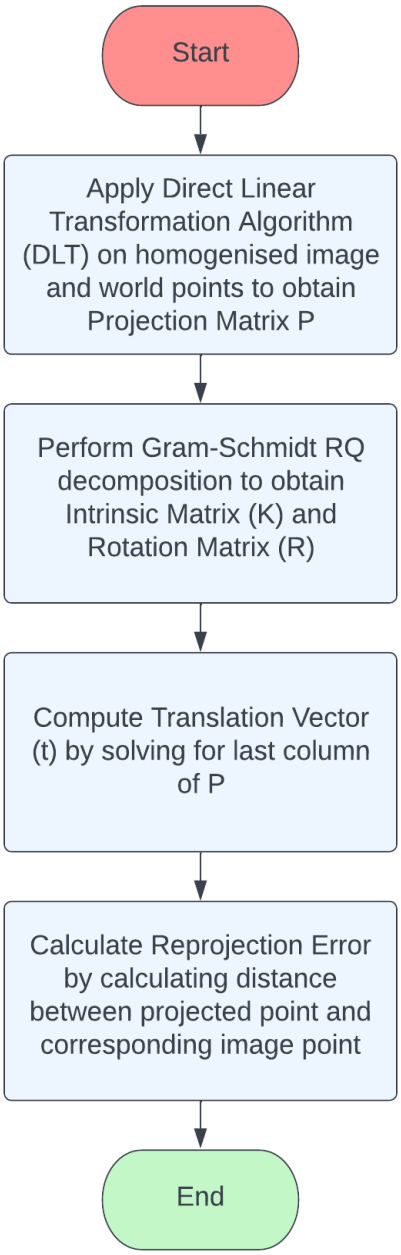


Figure 1 Block Diagram for computing P, K, R and t

I. Direct Linear Transformation (DLT) algorithm
   i. Define the image points and world points
   ii. Implement the DLT algorithm to compute the projection matrix (P) using the input image and world points
   iii. Return the projection matrix (P)

II. Decompose P matrix using Gram-Schmidt process
   i. Extract the 3x3 matrix M from P
   ii. Use the Gram-Schmidt process to compute the rotation matrix (R) and the intrinsic matrix (K)
   iii. Compute the translation vector (t) by solving for the last column of P
   iv. Normalize the intrinsic matrix K
   v. Return the intrinsic matrix (K), rotation matrix (R), and translation vector (t)

III. Compute reprojection error for each point
   i. For each point in the input world points, compute its projection onto the image plane using the projection matrix (P)
   ii. Normalize the projection
   iii. Compute the distance between the projected point and its corresponding image point
   iv. Print the reprojection error for each point

3. Direct Linear Transformation (DLT) Algorithm:
The DLT algorithm is used to estimate the projection matrix P. Given a set of n 3D world points and their corresponding 2D image points, the goal is to find the 3x4 projection matrix P. To achieve this, the algorithm first constructs a matrix A of size 2n x 12, where each row of A corresponds to a pair of image and world points, and the columns correspond to the elements of the projection matrix P. The algorithm then solves for P using Singular Value Decomposition (SVD) of A, where the last row of the V matrix corresponds to the elements of P.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This can be written as $[x\ y\ 1]^T = P\ [X\ Y\ Z\ 1]^T$
where, $[x\ y\ 1]^T$ are the homogeneous image coordinates of the 3D point [X Y Z]
P is the 3x4 projection matrix

$$A = \begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{bmatrix}$$

Here, X, Y, Z are the coordinates of the ith world point, and x, y are the coordinates of the ith image point. The first four columns of each row correspond to the homogeneous coordinates of the corresponding world point, while the last four columns correspond to the homogeneous coordinates of the corresponding image point. The remaining four columns are computed by multiplying each of the last four columns by the corresponding homogeneous coordinate of the world point and then negating the result.

The Projection Matrix P can be expressed as:

$$P = K[R|t]$$

where, K is the intrinsic matrix
R is the rotation matrix
t is the translation vector
The brackets indicate concatenation of the matrices.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

where, f_x and f_y are the focal lengths along the x and y axes
c_x and c_y are the coordinates of the principal point
s is the skew parameter
r_{ij} are the elements of the rotation matrix R
t_1, t_2, and t_3 are the components of the translation vector t in x y and z directions

Decompose P matrix using Gram-Schmidt process:
The Gram-Schmidt process is used to decompose the projection matrix P into the intrinsic matrix K and the extrinsic parameters R and t. The intrinsic matrix K contains information about the camera's focal length, principal point, and aspect ratio, while the extrinsic parameters R and t describe the camera's position and orientation with respect to the world coordinate system. The process involves computing the 3x3 rotation matrix R using the first three columns of P, and then using the inverse of R to compute the intrinsic matrix K and the translation vector t.

For Left 3x3 submatrix of P

$$M = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

Let,

$$\begin{aligned} a_1 &= \begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} \\ a_2 &= \begin{bmatrix} p_{21} & p_{22} & p_{23} \end{bmatrix} \\ a_3 &= \begin{bmatrix} p_{31} & p_{32} & p_{33} \end{bmatrix} \end{aligned}$$

Then,

$$e_3 = \frac{a_3}{||a_3||}$$

$$p_2 = (a_2 . e_3)e_3$$

$$e_2 = \frac{a_2 - p_2}{||a_2 - p_2||}$$

$$p_1 = (a_1 . e_2)e_2 + (a_1 . e_3)e_3$$

$$e_1 = \frac{a_1 - p_1}{||a_1 - p_1||}$$

$$R = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}$$

where, R is the rotation matrix

The intrinsic matrix K can be calculated by multiplying M with the inverse of R

$$\begin{aligned} M &= KR \\ MR^{-1} &= KRR^{-1} \\ K &= MR^{-1} \end{aligned}$$

where, K is an upper triangular matrix and R is an orthonormal matrix

The translation vector t can be found by extracting the final column of the V matrix obtained from the SVD of P

$$\begin{aligned} P &= U\,D\,V^T \\ t &= last\ column\ of\ V \end{aligned}$$

Compute reprojection error for each point:
The reprojection error is computed for each point by projecting the 3D world coordinates onto the 2D image plane using the estimated projection matrix P, and then computing the distance between the projected point and the corresponding image point. The reprojection error is a measure of the accuracy of the camera calibration and is an important metric for evaluating the performance of the algorithm.

Note: The code is given below and the matrices were validated using the inbuilt function scipy.linalg.rq and decomposeProjectionMatrix

Full Code:

```
 import numpy as np

# Define the image points and world points
image_points = np.array([[757, 213], [758, 415], [758, 686], [759, 966], [1190, 172], [329, 1041], [1204,
850], [340, 159]])
world_points = np.array([[0, 0, 0], [0, 3, 0], [0, 7, 0], [0, 11, 0], [7, 1, 0], [0, 11, 7], [7, 9, 0], [0,
1, 7]])

# Step 1: Direct Linear Transformation (DLT) algorithm
def dlt(image_points, world_points):
    A = []                                                  # Initialize an empty list to hold the
A matrix
    for i, (X, Y, Z) in enumerate(world_points):
        x, y = image_points[i]
        A.append([X, Y, Z, 1, 0, 0, 0, 0, -x*X, -x*Y, -x*Z, -x])    # Append the first row of the ith pair
of points inclusive of the homogenous coordinate
        A.append([0, 0, 0, 0, X, Y, Z, 1, -y*X, -y*Y, -y*Z, -y])    # Append the second row of the ith pair
of points inclusive of the homogenous coordinate
    A = np.asarray(A)                                       # Convert A to a numpy array

    # Solve for P using SVD
    _, _, V = np.linalg.svd(A)
    P = V[-1, :].reshape((3, 4))

    return P

# Step 2: Decompose P matrix using Gram-Schmidt process
def gram_schmidt(P):
    M = P[:3, :3].copy()                                    # Extract the 3x3 matrix M from P

    a1, a2, a3 = M[0,:], M[1,:], M[2,:]                     # Extract the first, second, and third rows of
M
    e3 = a3 / np.linalg.norm(a3)                            # Compute the third row of the rotation
matrix R
    p2 = np.dot(e3, a2) * e3                                # Compute the second row of the rotation
matrix R
    e2 = (a2 - p2) / np.linalg.norm(a2 - p2)
    p1 = np.dot(e2, a1) * e2 + np.dot(e3, a1) * e3          # Compute the first row of the rotation
matrix R
    e1 = (a1 - p1) / np.linalg.norm(a1 - p1)
    R = np.row_stack((e1, e2, e3))                          # Compute the rotation matrix R using the
Gram-Schmidt process

    K = M @ np.linalg.inv(R)                                # Compute the intrinsic matrix K
    K /= K[-1, -1]                                          # Normalize the intrinsic matrix K

    return K, R

# Step 3: Compute reprojection error for each point
def reprojection_error(image_points, world_points, P):
    for i, (X, Y, Z) in enumerate(world_points):
        p = P @ np.array([X, Y, Z, 1])                      # Compute the projection of the ith point
in world coordinates
        p /= p[-1]                                          # Normalize the projection
        error = np.linalg.norm(image_points[i] - p[:2])     # Compute the reprojection error
        print(f"Reprojection Error for Point {i+1} is: {error}")

P = dlt(image_points, world_points)
_, _, V = np.linalg.svd(P)                                  # Compute the SVD of P
t = V.T[:,-1].reshape(4,1)                                  # Extract the translation vector t from the
SVD of P
K, R = gram_schmidt(P)

np.set_printoptions(suppress=True, precision=6)             # Set the print options to suppress
scientific notation and print 6 decimal places
print("\nProjection Matrix (P):\n", P, "\n\nIntrinsic Matrix (K):\n", K, "\n\nRotation Matrix (R):\n", R,
"\n\nTranslation Vector (t):\n", t, "\n")
reprojection_error(image_points, world_points, P)
```

Terminal Output:

```
PS C:\Users\manda> & "C:/Program Files/Python311/python.exe" "c:/Users/manda/OneDrive - University of
Maryland/Perception For Autonmous Robots/Project 3/problem1.py"

Projection Matrix (P):
 [[ 0.036223 -0.002215 -0.088324  0.954089]
 [-0.025383  0.083056 -0.028002  0.268827]
 [-0.000035 -0.000003 -0.00004   0.001261]]

Intrinsic Matrix (K):
 [[1619.01802    -1.89271   800.113193]
 [   0.       1612.025941  616.150419]
 [   0.         -0.          1.      ]]

Rotation Matrix (R):
 [[ 0.749486  0.00587  -0.661994]
 [-0.045356  0.998066 -0.0425  ]
 [-0.660464 -0.061879 -0.748303]]

Translation Vector (t):
 [[-0.648624]
 [-0.301832]
 [-0.697519]
 [-0.040647]]

Reprojection Error for Point 1 is: 0.28561276727805496
Reprojection Error for Point 2 is: 0.9725828452229532
Reprojection Error for Point 3 is: 1.036081784374865
Reprojection Error for Point 4 is: 0.45408628677326207
Reprojection Error for Point 5 is: 0.19089831889735914
Reprojection Error for Point 6 is: 0.3189920832714891
Reprojection Error for Point 7 is: 0.19594240534327106
Reprojection Error for Point 8 is: 0.30829602844222703
```

Problem 2:

In this problem, you will perform camera calibration using the concepts you have learned in class. Assuming a pinhole camera model and ignoring radial distortion, we will be relying on a calibration target (checkerboard in our case) to estimate the camera parameters. The calibration target used can be found here.

This was printed on an A4 paper and the size of each square is 21.5 mm. Note that the Y axis has an odd number of squares and X axis has an even number of squares. It is a general practice to neglect the outer squares (extreme squares on each side and in both directions).

Thirteen images taken from a Google Pixel XL phone with focus locked can be downloaded from here which you will use to calibrate.
For this question, you are allowed to use any in-built function.

- **Find the checkerboard corners using any corner detection method (inbuilt OpenCV functions such as findChessboardCorners are allowed) and display them for each image.**
- **Compute the Reprojection Error for each image using built-in functions in OpenCV**
- **Compute the K matrix**
- **How can we improve the accuracy of the K matrix?**

Solution:

Camera calibration is the process of estimating the intrinsic and extrinsic parameters of a camera. Intrinsic parameters include the focal length, principal point, and lens distortion, while extrinsic parameters refer to the position and orientation of the camera in 3D space. In this problem, we will assume a pinhole camera model, where light passes through a single point (the focal point) and is projected onto an image plane. This model can be used to estimate the intrinsic parameters of the camera.

To calibrate the camera, we will use a calibration target, which is a planar object with a known geometry. In this case, we will use a checkerboard pattern, which has a regular grid of black and white squares. By capturing multiple images of the checkerboard pattern from different angles, we can use the correspondence between the 3D points on the checkerboard and their 2D projections in the image to estimate the intrinsic and extrinsic parameters of the camera.

The first step in camera calibration is to detect the corners of the checkerboard in each image. This can be done using any corner detection method, such as the Harris corner detector or the Shi-Tomasi corner detector. OpenCV provides an inbuilt function called findChessboardCorners that can be used to detect the corners of a checkerboard pattern in an image. Once the corners are detected, we can draw them on the image using the drawChessboardCorners function.

The next step is to compute the reprojection error for each image. The reprojection error measures the distance between the 2D projections of the 3D points on the checkerboard and their corresponding actual 2D image coordinates. The lower the reprojection error, the more accurate the calibration is. OpenCV provides a function called projectPoints that can be used to project the 3D points onto the image plane using the estimated camera parameters. We can then compute the reprojection error as the distance between the projected 2D points and the actual 2D image coordinates using the norm function in NumPy.

Once we have computed the reprojection error for each image, we can use these errors to compute the intrinsic parameters of the camera. The intrinsic parameters include the focal length and the principal point, which can be represented by the K matrix. OpenCV provides a function called calibrateCamera that can be used to estimate the K matrix and the distortion coefficients of the camera using the detected checkerboard corners and their 3D coordinates.

Functions Used:

i. **findChessboardCorners**: The findChessboardCorners function is an OpenCV function that can be used to detect the corners of a chessboard pattern in an image. The function takes as input the image, the size of the chessboard (number of inner corners), and a flag that specifies whether to use a more accurate corner detection method or not. The function returns a Boolean value indicating whether the corners were successfully detected, as well as the detected corners themselves. The function works by searching for a pattern of black and white squares in the image that matches the size of the chessboard. It then applies a corner detection algorithm to the intersection points of the black and white squares to detect the corners.

ii. **drawChessboardCorners**: The drawChessboardCorners function is another OpenCV function that can be used to draw the detected corners of a chessboard pattern on an image. The function takes as input the image, the size of the chessboard, and the detected corners. It then draws circles at the location of the detected corners, making it easy to visualize the accuracy of the corner detection algorithm.

iii. **projectPoints**: The projectPoints function is an OpenCV function that can be used to project 3D points onto a 2D image plane using the camera parameters. The function takes as input the 3D points, the rotation and translation vectors that define the pose of the camera, and the intrinsic parameters of the camera (the K matrix and distortion coefficients). It then applies the perspective projection equation to each 3D point to compute its 2D projection onto the image plane.

iv. **norm**: The norm function is a NumPy function that can be used to compute the Euclidean distance between two vectors. In the context of camera calibration, the function is used to compute the distance between the projected 2D points and the actual 2D image coordinates. The function takes as input the two vectors and returns the Euclidean distance between them.

v. **calibrateCamera**: The calibrateCamera function is an OpenCV function that can be used to estimate the intrinsic and extrinsic parameters of a camera using a set of calibration images. The function takes as input the 3D coordinates of the calibration target (usually a checkerboard pattern), the 2D coordinates of the detected corners in each image, and the size of the calibration target. It then uses an iterative optimization algorithm to estimate the intrinsic and extrinsic parameters of the camera, including the K matrix and the distortion coefficients.

Full Code:

```
import cv2
import numpy as np
import glob
import matplotlib.pyplot as plt

square_size = 21.5  # Size of each square in millimeters
pattern_size = (9, 6)   # Number of inner corners along the X and Y axis

obj = np.zeros((pattern_size[0]*pattern_size[1], 3), np.float32)   # Prepare object points for the
calibration target
obj[:, :2] = np.mgrid[0:pattern_size[0], 0:pattern_size[1]].T.reshape(-1, 2)   # Create a grid of points
obj *= square_size    # Scale the points

# Loop over all images and find checkerboard corners
obj_points, img_points = [], [] # Arrays to store object points and image points from all the images
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 9))  # Create a figure with 3 rows and 5 columns of
subplots
for i, image_path in enumerate(glob.glob('./Calibration_Imgs/*.jpg')):  # Loop over all images in the
folder
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)    # Convert the image to grayscale
    ret, corners = cv2.findChessboardCorners(gray, pattern_size, None)  # Find the chessboard corners
    if ret:
        obj_points.append(obj)
        img_points.append(corners)
        cv2.drawChessboardCorners(img, pattern_size, corners, ret)  # Draw the corners on the image
```

```
        # Calculate reprojection error for current image
        _, K, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, gray.shape[::-1], None,
None)
        img_points2, _ = cv2.projectPoints(obj, rvecs[-1], tvecs[-1], K, dist)
        error = cv2.norm(corners, img_points2, cv2.NORM_L2) / len(img_points2)

        # Plot the image with the reprojection error
        row = i // 5
        col = i % 5
        ax = axes[row, col]
        ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax.set_title(f"Image {i+1}\nReprojection Error: {error:.4f}")
        ax.axis('off')

# Hide remaining subplots
for j in range(i+1, 15):
    row = j // 5
    col = j % 5
    axes[row, col].set_visible(False)
plt.show()

# Print the camera matrix
np.set_printoptions(suppress=True, precision=6)              # Set the print options to suppress
scientific notation and print 6 decimal places
print(f"\nIntrinsic Matrix: \n{K}\n")
```

Output:

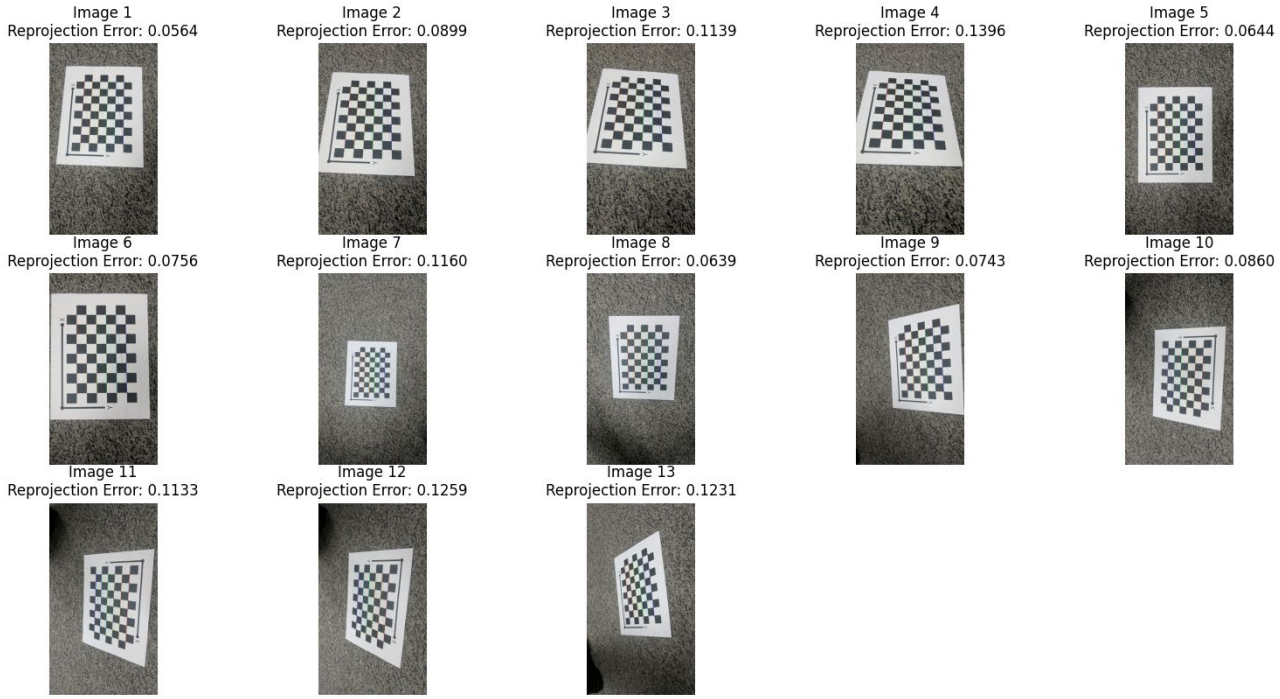


Figure 2 Plot with all the images and their corners

Terminal Output:

```
PS C:\Users\manda\OneDrive - University of Maryland\Perception For Autonmous Robots\Project 3> &
"C:/Program Files/Python311/python.exe" "c:/Users/manda/OneDrive - University of Maryland/Perception For
Autonmous Robots/Project 3/problem2.py"

Intrinsic Matrix:
[[2040.394711    0.        764.587598]
 [   0.       2032.174674 1359.294911]
 [   0.          0.          1.      ]]
```

Note: Use the zoom option displayed in the Matplotlib window to clearly see the drawn corners on the images.

Pipeline:

1. First, the necessary libraries are imported:
    a. **cv2** is OpenCV, which is used for computer vision tasks.
    b. **numpy** is a numerical library for Python, which is used for array operations.
    c. **glob** is used for finding files in a specified directory.
    d. **matplotlib** is a library for creating visualizations in Python.
2. The size of each square on the calibration target and the number of inner corners along the X and Y axis are defined. These values are used to generate the object points for the calibration target.
3. An array of zeros is created with the dimensions of the calibration target. The first two columns are populated with the X and Y coordinates of each point on the target, and the last column is set to zero.
4. The code loops over all images in a specified folder, using **glob.glob().** For each image:
    a. The image is read using **cv2.imread().**
    b. The image is converted to grayscale using **cv2.cvtColor().**
5. The OpenCV function **cv2.findChessboardCorners()** is used to find the chessboard corners in the grayscale image. This function takes three arguments: the grayscale image, the size of the calibration target, and a flag that determines whether to use a more accurate but slower algorithm for finding the corners.
6. If the corners are found in the image, the object points and image points are appended to separate arrays. The object points are the same for every image, but the image points vary depending on the position of the calibration target in the image.
7. The OpenCV function **cv2.drawChessboardCorners()** is used to draw the corners on the original color image. This function takes four arguments: the color image, the size of the calibration target, the image points, and a flag that indicates whether the corners were found successfully.
8. The OpenCV function **cv2.calibrateCamera()** is used to calculate the camera matrix, distortion coefficients, and rotation and translation vectors for the calibration target. This function takes five arguments: the object points, the image points, the shape of the grayscale image, the initial camera matrix (which is set to None in this case), and the initial distortion coefficients (also set to None). The camera matrix contains information about the camera's focal length, principal point, and distortion coefficients, and is used to undistort images captured by the camera.
9. The OpenCV function **cv2.projectPoints()** is used to project the object points onto the image using the rotation and translation vectors and the camera matrix. This function takes five arguments: the object points, the rotation vector for the current image, the translation vector for the current image, the camera matrix, and the distortion coefficients.
10. The image is plotted using **matplotlib's imshow()** function, with the corners drawn on top. The title of each subplot includes the image number and the reprojection error for that image. The reprojection error is calculated by dividing the Euclidean distance between the detected corners and the projected object points by the number of corners. A lower reprojection error indicates a better calibration.
11. Finally, the camera matrix is printed using NumPy's print() function, with options to suppress scientific notation and display 6 decimal places. The camera matrix is a 3x3 matrix that describes the intrinsic parameters of the camera, including the focal length, principal point, and distortion coefficients.

Methods to improve the accuracy of Intrinsic Matrix (K):

The K matrix, also known as the camera matrix or intrinsic matrix, is a 3x3 matrix that represents the intrinsic parameters of the camera. These parameters include the focal length, principal point, and skew coefficient. The accuracy of the K matrix is crucial in computer vision applications, as it affects the accuracy of the 3D reconstruction, object tracking, and camera pose estimation.

1. **Increase the number of images**: Using more images for calibration can increase the accuracy of the K matrix. Ideally, we should use at least 10-15 images of the calibration target to ensure that we capture the full range of camera poses and lighting conditions.
2. **Improve the quality of the calibration target**: The calibration target should be printed on high-quality paper with a high-contrast pattern. The size of the squares should be accurate, and the target should be free

from any distortion or warping. Any defects or distortions in the calibration target can affect the accuracy of the K matrix.

3. **Use a larger calibration target**: A larger calibration target provides more information about the camera's intrinsic and extrinsic parameters, resulting in a more accurate K matrix. A larger calibration target also allows us to capture more accurate 3D coordinates of the calibration points.
4. **Use a more accurate corner detection method**: The accuracy of the corner detection method can affect the accuracy of the K matrix. In some cases, a more accurate corner detection method, such as the SIFT or SURF feature detector, may be required. These feature detectors are more robust to changes in lighting and orientation and can provide more accurate corner detections.
5. **Consider radial and tangential distortion**: Radial and tangential distortions are types of lens distortion that can affect the accuracy of the K matrix. To improve the accuracy of the K matrix, we can consider the effects of radial distortion and tangential distortion and use a distortion model, such as the Brown-Conrady model, to correct for it.
6. **Ensure sufficient camera coverage**: When capturing images for calibration, it's important to ensure that the camera covers a wide range of angles, depths, and distances from the calibration target. This will help capture a more accurate representation of the camera's intrinsic parameters.
7. **Use a better optimization algorithm**: The accuracy of the K matrix can be further improved by using a more advanced optimization algorithm to solve for the camera parameters. For example, the Levenberg-Marquardt algorithm is a popular choice for optimizing camera parameters in computer vision applications.
8. **Account for lens distortion using more advanced model**s: While radial and tangential distortion models can help correct for some lens distortions, more advanced models such as fisheye and polynomial models can provide even better correction. Using a more accurate lens distortion model can help improve the accuracy of the K matrix.
9. **Use a high-quality lens**: The quality of the camera lens can greatly affect the accuracy of the K matrix. A high-quality lens with low distortion and chromatic aberration can provide more accurate images, resulting in a more accurate K matrix.
10. **Consider the effects of sensor noise**: The sensor noise in the camera can also affect the accuracy of the K matrix. To minimize the effects of sensor noise, it's important to capture images at the camera's native ISO, use proper exposure settings, and avoid overexposure or underexposure.

By considering these additional factors, we can further improve the accuracy of the K matrix and ensure that our camera calibration is as accurate as possible.