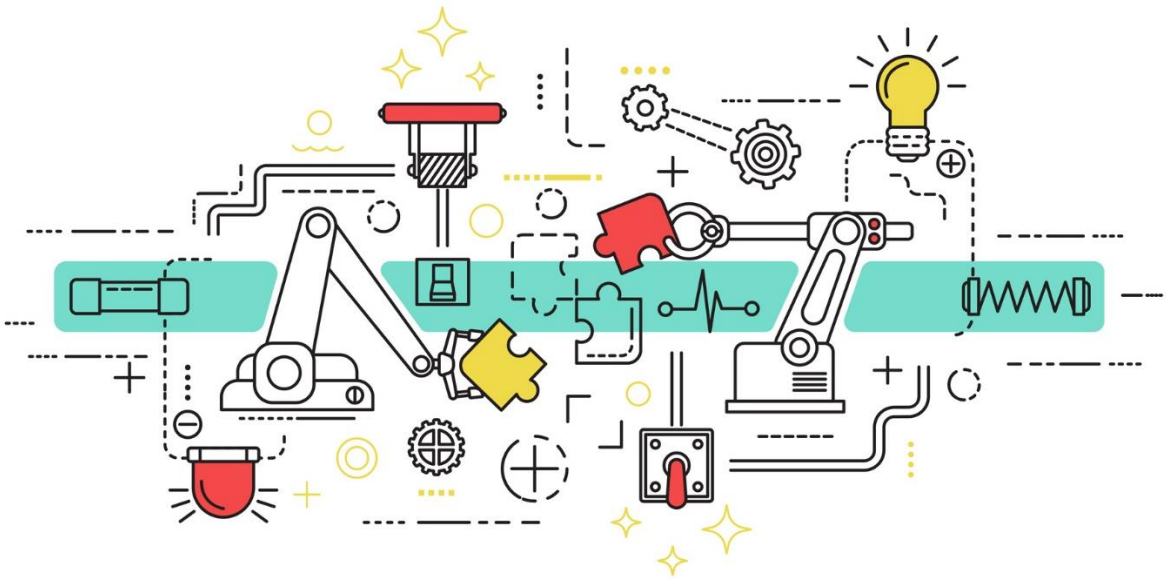
ENPM 673

Perception for Autonomous Robots



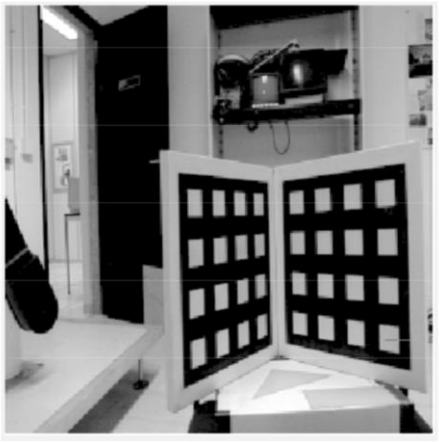
Project - 3

2023

Datta Lohith Gannavarapu
119455395
gdatta@umd.edu

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.

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

Answer:

Theory behind the solution to the above problem:

Camera calibration is the process of determining the intrinsic and extrinsic parameters of a camera. The intrinsic parameters of a camera are its internal characteristics such as focal length, principal point, and distortion coefficients, which are specific to each camera. The extrinsic parameters, on the other hand, define the position and orientation of the camera in 3D space. In this report, we will discuss the mathematics behind camera calibration, the different methods used, and its applications.

The process of camera calibration involves finding the relationship between the 3D world coordinates and their corresponding 2D image coordinates on the camera sensor. This relationship can be described using the pinhole camera model. Using the Tsai grid, we calibrate the camera.

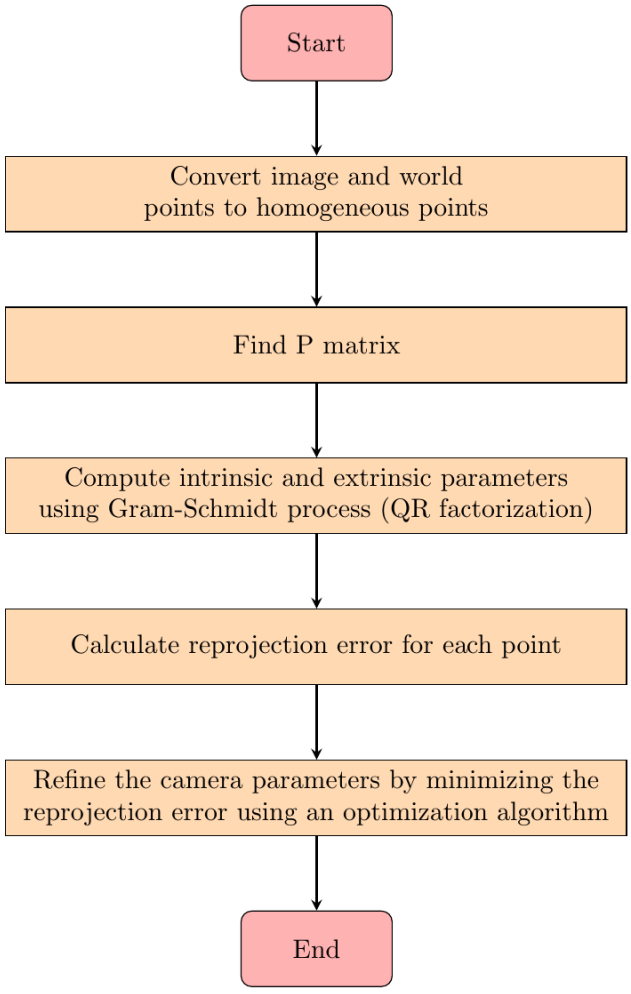


Figure 1: Pipeline to calibrate the camera using the image provided

To Mathematically Solve the Camera Calibration Problem, a minimum of 6 points are required.

Mathematics to solve the problem:

The intrinsic parameters of a camera can be calculated using the camera matrix. The camera matrix relates the 3D world coordinates to their corresponding 2D image coordinates on the camera sensor. The Image coordinates and the world coordinates can be represented as follows w.r.t P matrix.

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

(u, v) are the image coordinates
(X, Y, Z) are the world coordinates.

The P matrix can be represented as

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}$$

We can get the P – matrix from image and world points by the following equation

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -u_2X_2 & -u_2Y_2 & -u_2Z_2 & -u_2 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

It can also be represented w.r.t intrinsic and Extrinsic parameters as

$$P = \begin{bmatrix} \alpha_x & 0 & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where $K = \begin{bmatrix} \alpha_x & 0 & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ and $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ $T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$

The K,R,T matrices can be decomposed from the P matrix by using Gram-Schmidt method.

Decomposing $A = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$ gives Rotation matrix and Intrinsic parameters matrix.

Gram-Schmidt method –

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{a}_2 - \operatorname{proj}_{\mathbf{u}_1} \mathbf{a}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{a}_3 - \operatorname{proj}_{\mathbf{u}_1} \mathbf{a}_3 - \operatorname{proj}_{\mathbf{u}_2} \mathbf{a}_3, & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
&\vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{a}_k - \sum_{j=1}^{k-1} \operatorname{proj}_{\mathbf{u}_j} \mathbf{a}_k, & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}
\end{aligned}$$

a_i can be expressed as

$$\begin{aligned}
\mathbf{a}_1 &= \langle \mathbf{e}_1, \mathbf{a}_1 \rangle \, \mathbf{e}_1 \\
\mathbf{a}_2 &= \langle \mathbf{e}_1, \mathbf{a}_2 \rangle \, \mathbf{e}_1 + \langle \mathbf{e}_2, \mathbf{a}_2 \rangle \, \mathbf{e}_2 \\
\mathbf{a}_3 &= \langle \mathbf{e}_1, \mathbf{a}_3 \rangle \, \mathbf{e}_1 + \langle \mathbf{e}_2, \mathbf{a}_3 \rangle \, \mathbf{e}_2 + \langle \mathbf{e}_3, \mathbf{a}_3 \rangle \, \mathbf{e}_3 \\
&\vdots \\
\mathbf{a}_k &= \sum_{j=1}^{k} \langle \mathbf{e}_j, \mathbf{a}_k \rangle \, \mathbf{e}_j
\end{aligned}$$

where $\langle \mathbf{e}_i, \mathbf{a}_i \rangle = \|\mathbf{u}_i\|$. This can be written in matrix form:

$$A = QR$$

where:

$$Q = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_n \end{bmatrix}$$

and

$$R = \begin{bmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \langle \mathbf{e}_1, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_1, \mathbf{a}_n \rangle \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \langle \mathbf{e}_2, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_2, \mathbf{a}_n \rangle \\ 0 & 0 & \langle \mathbf{e}_3, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_3, \mathbf{a}_n \rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \langle \mathbf{e}_n, \mathbf{a}_n \rangle \end{bmatrix}.$$

The Upper triangular matrix after decomposition will be Intrinsic parameters matrix and the other matrix would be the Rotation Matrix.

Now, for the translation matrix, the last column of the P matrix can be represented as

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = K \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

From which we get

$$T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$

Problems Encountered for Problem 1 -

- **Decomposing the P matrix –** Proper guide was found on Wikipedia and that helped to implement the Gram-Schmidt method for decomposition of P matrix.
- **Validating the Obtained Intrinsic and Extrinsic matrices –** Compared the results with the inbuilt function and validated the results.

Execution –

The code provides a solution to estimate the camera calibration parameters from a set of corresponding image and world points using the mathematics explained above.

Steps followed –

- Importing numpy library
- Creating arrays to store image and world points
- Converting the points to homogeneous coordinates
- Using the function **find_P_matrix** to find P matrix and the reprojection error of each point from the given points
- Using the function **decompose_P** to get the R,K,T matrices

Python Code –

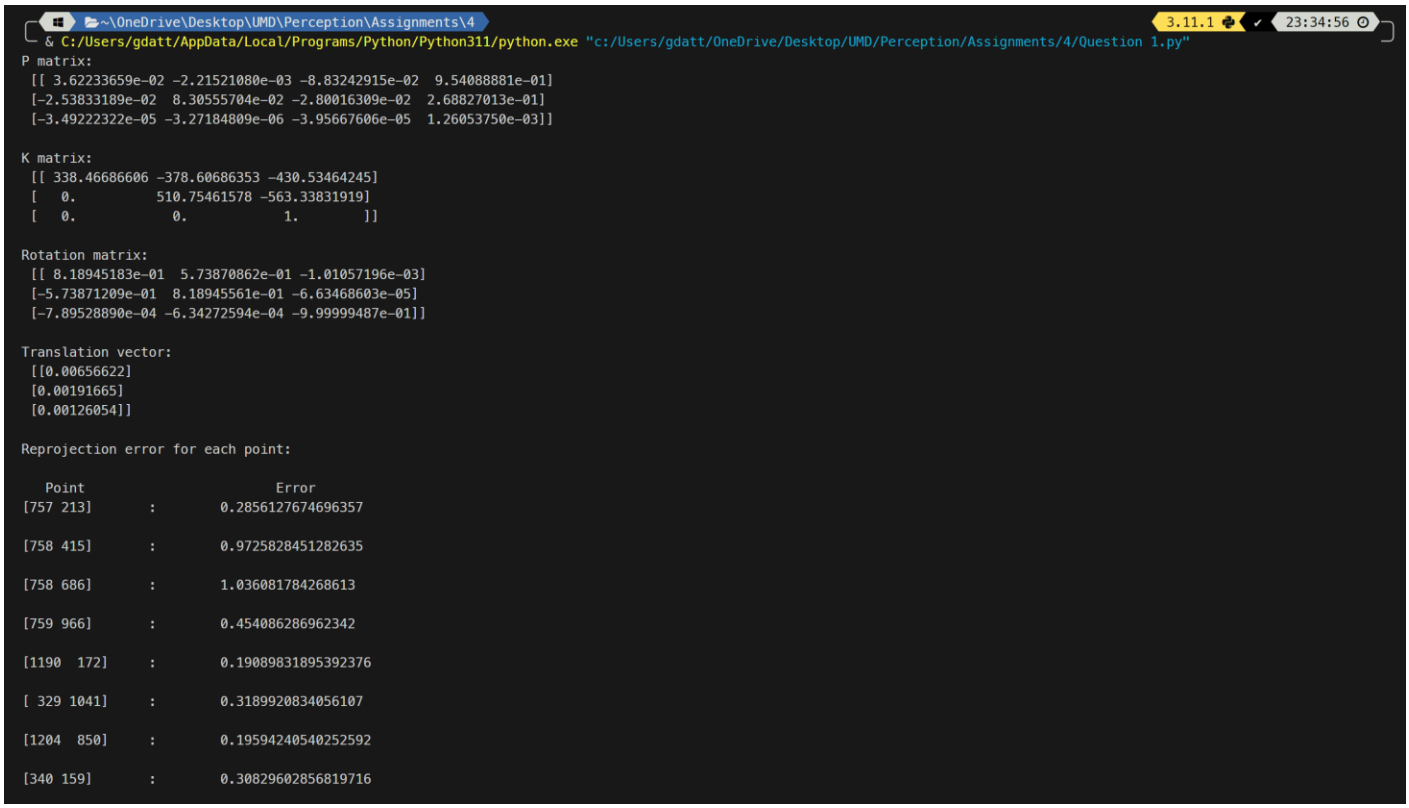
```
import numpy as np

# Image points
image_points = np.array([[757, 213], [758, 415], [758, 686], [759, 966],
                         [1190, 172], [329, 1041], [1204, 850], [340, 159]])

# World points
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]])

def decompose_P(P):
    # Implementing Gram-Schmidt process to decompose P matrix for Intrinsic and Extrinsic
parameters
    P_k = P[:, :3].copy()

    # Number of rows and columns
    n, m = P_k.shape

    # Empty Matrices for Rotation(R) and Intrinsic parameters(K) matrices
    R = np.zeros((n, m))
    K = np.zeros((m, m))

    # Gram-Schmidt process
    v= P_k[:, 0]
    K[0, 0] = np.linalg.norm(v)
    R[:, 0] = v/K[0, 0]

    for j in range(1, m):
        v = P_k[:, j]
        for i in range(j):
            K[i, j] = np.dot(R[:, i], P_k[:, j])
            proj = K[i,j]/np.dot(R[:, i], R[:, i])
            v -= proj * R[:, i]
        K[j, j] = np.linalg.norm(v)
        R[:, j] = v / K[j, j]

    # Normalizing K matrix in last column
    K = K/K[2,2]

    # Decomposing P matrix for Translation vector(T)
    T= (np.linalg.inv(K) @ P[:,3])
    T.shape = (3,1)
    return R,K,T

def find_P_matrix(image_points, world_points):

    # Convert to homogeneous coordinates
    image_points_hom = np.hstack((image_points, np.ones((8, 1))))
```

```
    world_points_hom = np.hstack((world_points, np.ones((8, 1))))

    # Define A matrix
    A=[]
    for i in range(len(image_points_hom)):
        A.append(np.hstack((world_points_hom[i], np.zeros(4), -image_points_hom[i, 0] *
world_points_hom[i])))
        A.append(np.hstack((np.zeros(4), world_points_hom[i], -image_points_hom[i, 1] *
world_points_hom[i])))

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

    # Verify that the projection is accurate
    image_points_pred = np.dot(world_points_hom, P.T)
    image_points_pred /= image_points_pred[:, 2][:, np.newaxis]
    errors = image_points_pred - image_points_hom

    return P,errors


if __name__ == "__main__":
    # Find P matrix
    P,E = find_P_matrix(image_points, world_points)
    print("P matrix:\n", P)

    # Decompose P matrix for Intrinsic and Extrinsic parameters
    R,K,T = decompose_P(P)
    print("\nK matrix:\n", K)
    print("\nRotation matrix:\n", R)
    print("\nTranslation vector:\n", T)

    print("\nReprojection error for each point:\n\n   Point\t\t\tError")
    # Reprojection error for each point
    for i in range(len(E)):
        print(f"{image_points[i]} \t:\t {np.linalg.norm(E[i])}\n")
```

Functions used –

- **find_P_matrix –** This function takes in the image and world points as input and returns the projection matrix P derived using SVD. It also calculates the reprojection error for each point and returns them.
- **decompose_P –** This function decomposes the projection matrix P into intrinsic and extrinsic parameters using the Gram-Schmidt process. It returns the rotation matrix R, intrinsic matrix K and the translation vector T.

Terminal Output –



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 21.5mm. 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 anyin-builtfunction.

- 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?

Answer:

Theory behind the solution to the above problem:

Camera calibration is the process of determining the intrinsic and extrinsic parameters of a camera. The intrinsic parameters of a camera are its internal characteristics such as focal length, principal point, and distortion coefficients, which are specific to each camera. The extrinsic parameters, on the other hand, define the position and orientation of the camera in 3D space. In this report, we will discuss the mathematics behind camera calibration, the different methods used, and its applications.

The process of camera calibration involves finding the relationship between the 3D world coordinates and their corresponding 2D image coordinates on the camera sensor. This relationship can be described using the pinhole camera mode. A chessboard picture is given and 13 images are provides which were taken from a camera.

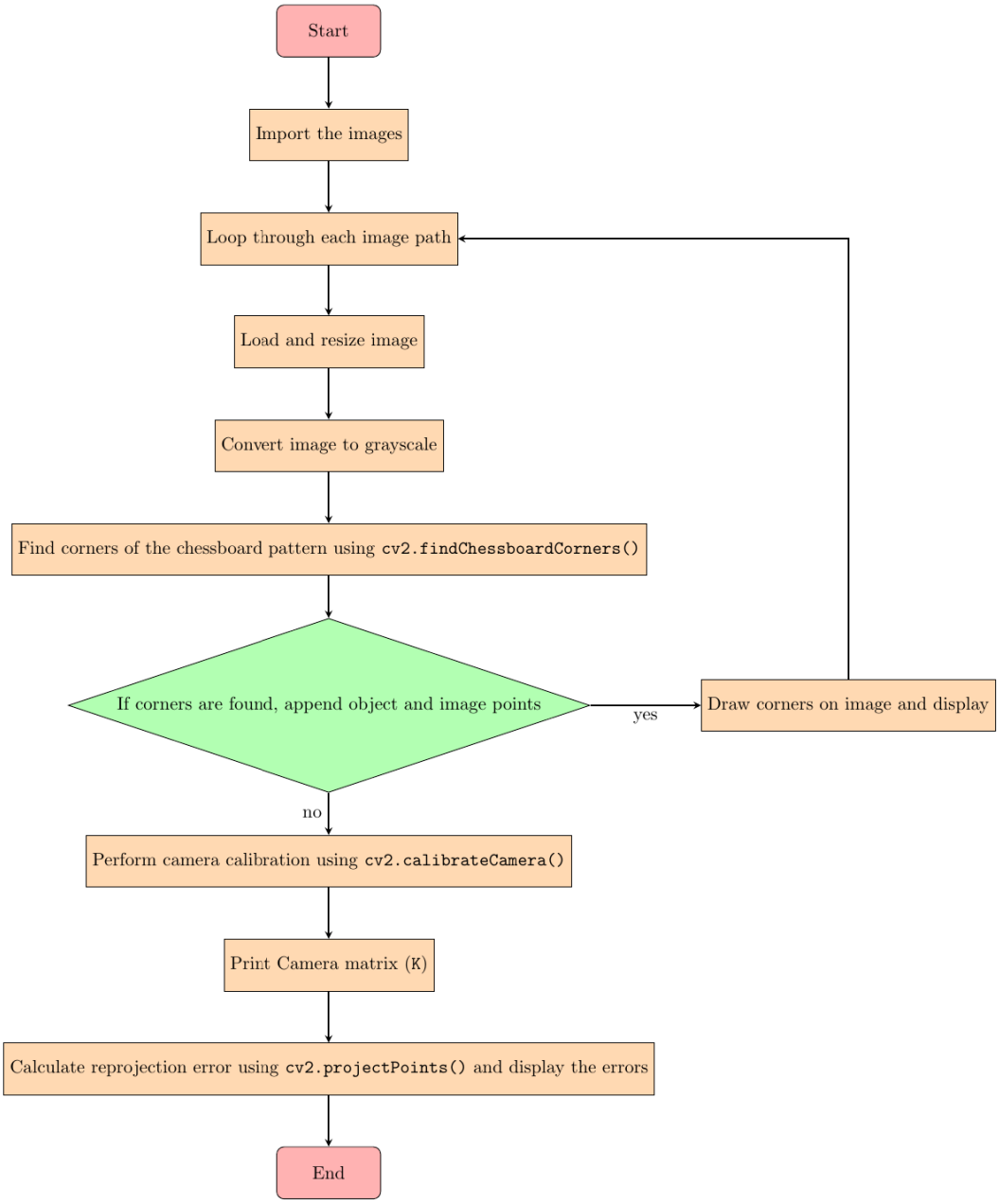


Figure 2: Pipeline for Problem 2

Problems Encountered for Problem 2 -

- **Visualizing the images –** Resized the image preserving the aspect ratio for better visualization and computation.
- **Disparity in results –** Eliminated the disparity in results due to resizing images by scaling up the results to the original image size.

Execution –

This code is for camera calibration using OpenCV library. It calibrates a camera using a chessboard calibration target with known square size, and outputs the intrinsic camera matrix and distortion coefficients.

Steps followed –

- Import the necessary libraries (numpy, cv2, glob, and os).
- Define the scale, objpoints, and imgpoints variables as global variables.
- Define the load_resize function to load and resize the image.(The images are reloaded for better visualization preserving the aspect ratio)
- Define the K_matrix function to scale the camera matrix to the original image size.
- Set the path to the calibration images and define the parameters for the calibration target.
- Prepare object points for the calibration target using the pattern size and square size.
- Get the paths of the calibration images using glob.
- Loop through each image path:
    - Load and resize the image.
    - Convert the image to grayscale.
    - Find the corners of the chessboard pattern using cv2.findChessboardCorners()
    - If the corners are found, append the object points and image points to their respective lists.
    - Draw the corners on the image using cv2.drawChessboardCorners() and display the image.
- Perform camera calibration using cv2.calibrateCamera() to get the camera matrix (mtx), distortion coefficients (dist), and rotation and translation vectors (rvecs and tvecs).
- Print the scale and camera matrix (K) scaled to the original image size.
- Loop through each image and calculate the reprojection error using cv2.projectPoints().
- Scale the reprojection error to the original image size and print it for each image.

Python code –

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

scale = 1                                                    # Global variable to
store the scaling factor
objpoints = []                                               # Real world 3D points
imgpoints = []                                               # Image points

# Function to load and resize the image
def load_resize(img_path, ratio=0.3):
    global scale
```

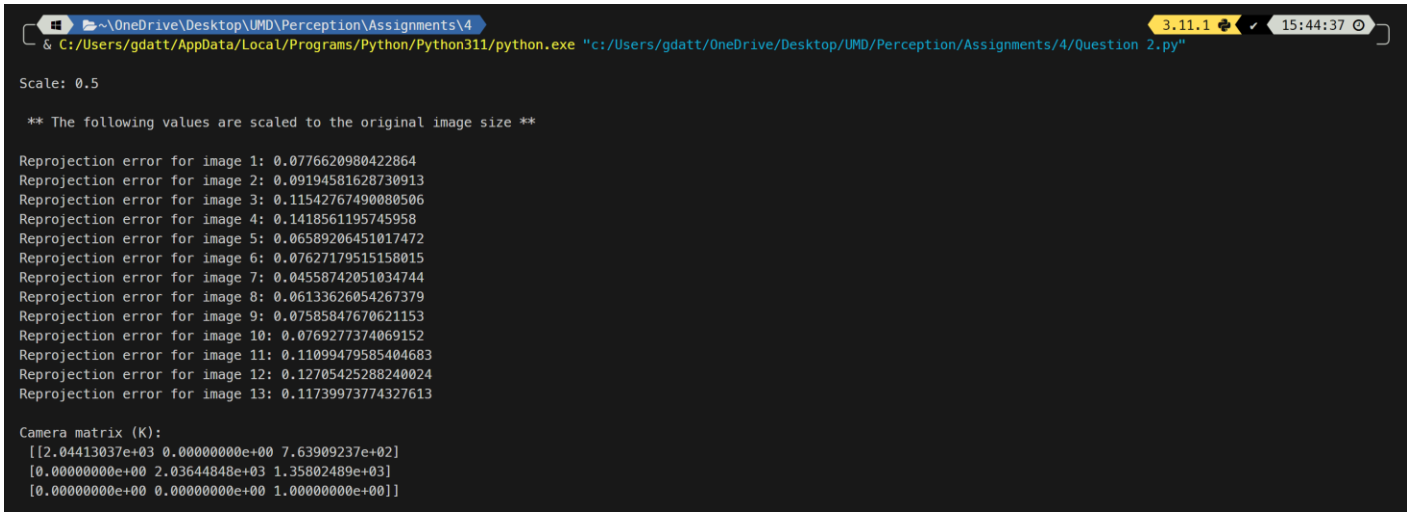
```
    img = cv.imread(img_path)
    h, w = img.shape[:2]
    scale = ratio
    new_h = int(ratio * h)
    new_w = int(ratio * w)
    return cv.resize(img, (new_w, new_h))

# Function to scale the K matrix to the original image size
def K_matrix(mtx):
    scale_matrix = np.array([[1/scale, 0, 0], [0, 1/scale, 0], [0, 0, 1]])
    K= np.dot(scale_matrix, mtx)
    return K

# Path to the calibration images
CURRENT_DIR = os.path.dirname(__file__)
images_path = os.path.join(CURRENT_DIR,'Calibration_Imgs/*.jpg')

# Parameters for the calibration target
square_size = 21.5
pattern_size = (9, 6)

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

# Getting the paths of Calibration Images
images = glob.glob(images_path)

for image in images:
    # Getting the Resized Image and converting it to grayscale
    img = load_resize(image,0.5)                                    # Change the scaling
factor here(default: 0.3)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chessboard corners
    ret, corners = cv.findChessboardCorners(gray, pattern_size, None)

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

        # Draw the corners on the image and display it
        cv.drawChessboardCorners(img, pattern_size, corners, ret)
        cv.imshow('img', img)
        cv.waitKey(200)

# Performing Camera Calibration
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],
None, None)
```

```
print(f"\nScale: {scale}\n\n ** The following values are scaled to the original image size
**\n")

# Calculating the Reprojection error for each image
for i in range(len(images)):
    rvec, tvec = rvecs[i], tvecs[i]
    projected_points, _ = cv.projectPoints(objpoints[i], rvec, tvec, mtx, dist)
    reprojection_error = cv.norm(imgpoints[i], projected_points,
cv.NORM_L2)/len(projected_points)
    reprojection_error = reprojection_error / scale                # Scaling the error to
the original image size
    print(f'Reprojection error for image {i+1}: {reprojection_error}')

K = K_matrix(mtx)                                                  # Scaling the K matrix
to the original image size
print(f"\nCamera matrix (K):\n {K} \n")
```

Functions used –

- **load_resize -** This function loads and resizes an image. The input parameters are the image path and the scaling factor (default is 0.3). It returns the resized image.
- **K_matrix -** This function scales the input camera matrix mtx to the original image size based on the global variable scale. It returns the scaled camera matrix K.

Terminal Output –



Ways to improve K-Matrix –

The K matrix, also known as the calibration matrix or camera matrix, is a fundamental parameter in computer vision and image processing. It describes the relationship between the 3D world coordinates and the 2D image coordinates. Improving the accuracy of the K matrix can enhance the overall performance of computer vision algorithms such as object detection, tracking, and 3D reconstruction. Some of the ways to improve the accuracy of the K matrix for better performance in Computer Vision algorithms are as follows:

- **Use high-quality camera equipment:** Using high-quality cameras with high-resolution sensors and low lens distortion can improve the accuracy of the K matrix.
- **Use a reliable calibration method:** There are various calibration methods available for computing the K matrix, such as the Zhang method, the Bouguet method, and the Tsai method. Each method has its advantages and disadvantages, and choosing a reliable and accurate method is crucial.
- **Increase the number and variety of calibration images:** The accuracy of the K matrix depends on the number and variety of calibration images used. Increasing the number of images and capturing images of different scenes and lighting conditions can improve the accuracy of the K matrix.
- **Ensure accurate image point correspondences:** Accurate image point correspondences between the 3D world coordinates and the 2D image coordinates are crucial for accurate calibration. Using feature detection and matching techniques can improve the accuracy of the correspondences.
- **Perform multiple calibrations and average the results:** Performing multiple calibrations and averaging the results can improve the accuracy of the K matrix. This is because each calibration may have slight errors, and averaging can reduce the overall error.
- **Regularly recalibrate the camera:** Camera calibration is not a one-time process, and regular recalibration is necessary to account for changes in the camera's internal parameters due to wear and tear or changes in the environment.