Assignment 1 - Camera Calibration
Computer Vision

Name: Anantha Lakshmi
Roll No: 2020101103

Question 1

1.1. Internal corners detection

I started with importing some libraries including scipy, numpy, random, math, cv2, and matplotlib.pyplot. This combination of libraries suggests you might be working on some image processing or computer vision task, especially with the inclusion of cv2 (OpenCV) and matplotlib.pyplot for visualization.

Procedure:

- **Loading the Image:** The code starts by loading an image named 'calib-object.jpg' using OpenCV's cv2.imread() function.

- **Splitting the Image:** The loaded image is then split into two halves using NumPy slicing. This is done to process each half separately.

- **Finding Chessboard Corners:** The find_chessboard_corners() function is defined to detect the corners of a chessboard pattern in an image. It takes an image as input and performs the following steps:
- Convert the image to grayscale using cv2.cvtColor().
- Use cv2.findChessboardCorners() to find the corners of a chessboard pattern. This function returns a flag indicating whether the corners are found (ret) and the coordinates of the corners (corners).
- If the corners are found (ret is True), refine the corner positions using cv2.cornerSubPix().
- Return the refined corner positions.

- **Finding Chessboard Corners for Each Half:** The find_chessboard_corners() function is applied separately to the left and right halves of the image.

- **Concatenating Corner Arrays:** If corners are found in both the left and right halves (left_corners and right_corners are not None), the x-coordinates of the corners in the right half are shifted to match the original image's coordinates. Then, the corner arrays are concatenated along the rows to form all_corners.

- **Drawing Corners on the Original Image:** The original image is copied, and red circles representing the detected corners are drawn on it using cv2.circle().

- **Displaying the Image with Detected Corners:** The modified image with detected corners is displayed using matplotlib.pyplot.imshow()

Results:

Challenges:

- Image loading and splitting for consistency across varied images.
- Accurate chessboard corner detection despite lighting and noise.
- Precise drawing of corners on the original image.
- Graceful error handling for cases where corners aren't detected.

Learnings:

- Hands-on experience with OpenCV for image processing.
- Optimization of corner detection algorithms for accuracy and efficiency.
- Effective visualization of detected features on images.
- Improved debugging skills for image processing tasks.

1.2 Camera calibration

Procedure:

The provided code defines functions to compute a projection matrix from 3D world points to 2D image points and to find the image points given the projection matrix and 3D world points. Additionally, it calculates errors for each point and extracts intrinsic and rotation matrices from the projection matrix.
Here's a brief summary of the procedure and algorithm:

1. **Projection Matrix Computation (projection_matrix function)**:
   - This function takes two lists of corresponding 2D image points (X2) and 3D world points (X3).
   - It constructs a matrix M by iterating through each point and forming two columns representing the transformation equations from world coordinates to image coordinates.
   - Singular Value Decomposition (SVD) is then applied to M to get the right singular vector corresponding to the smallest singular value.
   - The last row of V (right singular vectors) normalized by its last element yields the projection matrix P.
2. **Image Point Computation (Imgpoint function):**

- This function computes the image point coordinates from the projection matrix and 3D world coordinates.
- It appends 1 to the world point coordinates to make them homogeneous.
- Multiplying the projection matrix by the homogeneous world coordinates and normalizing by the last component gives the image point coordinates.

3. **Error Calculation:**
   - For each world point, the Imgpoint function is used to compute the corresponding image point.
   - The Euclidean distance between the computed image point and the actual image point is calculated to measure the error.
4. **Extraction of Intrinsic and Rotation Matrices:**
   - The QR decomposition is applied to the leftmost 3x3 submatrix of the projection matrix to obtain the intrinsic (K) and rotation (R) matrices.
   - The projection matrix is split into the intrinsic and rotation matrices.
5. **Projection Center Calculation:**
   - The Imgpoint function is used to find the image coordinates corresponding to the world origin (0, 0, 0), representing the projection center.

Results:

```
Projection matrix:
 [[ 2.41878308e+00 -5.97330511e-01 -6.19376244e+00  1.55722673e+03]
 [-1.22746048e+00 -6.70810144e+00 -1.19573784e+00  1.60202663e+03]
 [-1.13872849e-03 -3.72516859e-04 -1.09699874e-03  1.00000000e+00]]
Intrinsic matrix:
 [[-2.71241076e+00 -2.50298258e+00  4.98215308e+00]
 [ 0.00000000e+00  6.25224015e+00  3.86918867e+00]
 [ 0.00000000e+00  0.00000000e+00 -3.60837247e-03]]
Rotation matrix:
 [[-8.91746604e-01 -4.52534877e-01  4.23468808e-04]
 [ 4.52534880e-01 -8.91746698e-01 -9.32406387e-05]
 [ 4.19821552e-04  1.08487383e-04  9.99999906e-01]]
Projection center Coordinates: [1557.2267335  1602.02663112]
```

Challenges:

- **Numerical Stability:** Matrix computations, such as SVD and matrix multiplication, can suffer from numerical instability, especially when dealing with large or ill-conditioned matrices. Ensuring numerical stability is crucial for accurate results.

- **Error Analysis:** Evaluating the accuracy of the computed projection matrix by comparing the projected image points with the actual image points involves error analysis. Different sources of error, including noise in the data or inaccuracies in the camera model, need to be considered.

Learnings:

- **Understanding Camera Geometry**: Implementing the projection matrix computation provides a deeper understanding of the geometric relationship between 3D world coordinates and 2D image coordinates in a pinhole camera model.

1.3

Procedure

Data Preparation:

- Define a list of points containing 3D coordinates representing points in space.These points likely correspond to vertices of a wireframe object in a 3D space.
- Convert these 3D points into 2D image coordinates, possibly using a projection matrix (P) and a transformation function (Imgpoint). Append these transformed 2D points to a list wireframe_points.
- **Plotting Wireframe (plot_wireframe function):**The function takes projections (2D image coordinates) and image (the image on which the wireframe will be plotted) as inputs.

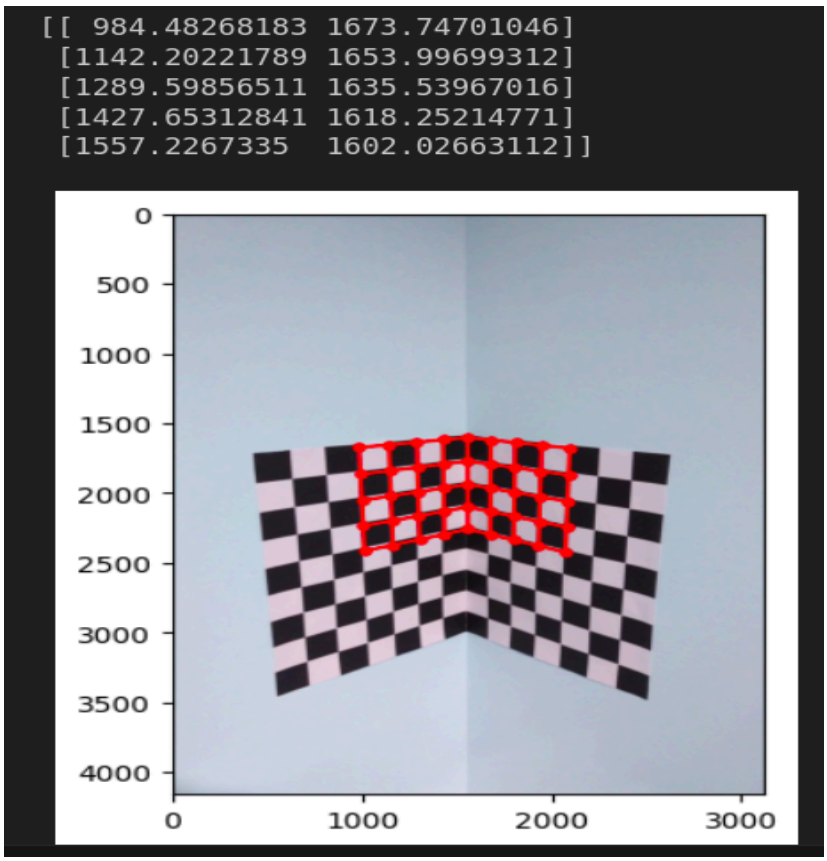
- Scatter plot: Scatter plot the 2D image coordinates on the image using red color.

Vertical Lines: Iterate through the range of points in the wireframe (projections) with a step of 1.For each pair of points (u, v) where v is 1 more than u, check if (u+1)%9 != 0 (this ensures the points are not on the same vertical line).Plot a line connecting points (u, v) on the image using red color.

Horizontal Lines: Iterate through each row of the wireframe (there are 9 rows as per the code).Within each row, iterate through the points with a step of 9 (to move to the next row).Plot lines connecting each pair of consecutive points within the same row.

Display the image with the plotted wireframe using plt.imshow and plot.show.

Results:



Challenges:

Modularity: The code combines data processing, plotting, and main execution within a single function. This lack of modularity makes it harder to maintain, debug, and extend the code. Separating concerns into distinct functions would improve readability and reusability.

Hardcoded Values: The code contains hardcoded values such as ranges in for loops, which limits its flexibility and adaptability to different datasets or scenarios. Parameterizing these values would make the code more versatile and easier to customize

Learnings:

Modularization: Breaking down the code into smaller, modular functions with well-defined purposes improves readability, maintainability, and reusability. Each function should ideally perform a single task or encapsulate a specific functionality.

1.4 Calculating euler angles

- Rotation Matrix (R):The input is a 3x3 rotation matrix R representing a rotation transformation in 3D space.

- Calculate Beta: The Euler angle beta is calculated using the arcsine of the negative value of the (1,3) element of the rotation matrix R. This corresponds to the sin of the rotation angle about the y-axis (often denoted as pitch).
- Check Beta's Validity: Check if beta is not equal to pi/2 or -pi/2. If it is, some values cannot be determined uniquely. In such cases, alpha is set to 0 (can be chosen arbitrarily), and gamma is calculated differently. If beta is not at the limits, then alpha and gamma are calculated using arctan2.
- Convert Angles to Degrees: Convert the calculated angles from radians to degrees. And printed euler angles.

Results:

```
Euler Angles:
Alpha: -0.005342295563308307
Beta: -0.0242629761789889
Gamma: -153.0935616836494
```

Question 2

2.1

Procedure: Here's a brief explanation of the procedure:

1. **Initialization:**
- Define object points (obj_points) and image points (image_points). These points represent corresponding 3D points in the world and their 2D locations in the image, respectively.
- Set an initial guess for the camera matrix (initial_mtx) and distortion coefficients (dist).
- Read the image (image_path) and convert it to grayscale.

2. **Calibration:**
- Use cv2.calibrateCamera to calibrate the camera. This function estimates the camera matrix, distortion coefficients, rotation vectors (rvecs), and translation vectors (tvecs) that minimize reprojection error between the detected points and the corresponding points in the real world.
- Pass object points, image points, the shape of the grayscale image, initial guess camera matrix, initial guess distortion coefficients, and optional flags.
3. **Defining Matrices and Parameters:**
- It defines a rotational matrix R, translational coefficients t, and the final camera matrix K.
- Constructs the extrinsic matrix by horizontally stacking R and t.

- Calculates the camera projection matrix P by multiplying K and the extrinsic matrix.

4. **Projection of 3D Points to 2D Image Plane:**

- Defines a set of 3D points.
- Projects these points onto the 2D image plane using the camera projection matrix P and a custom function Imgpoint.

5. **Visualization:**

- Generates wireframe points representing the 3D structure of an object.
- Uses OpenCV to plot the wireframe onto an image.

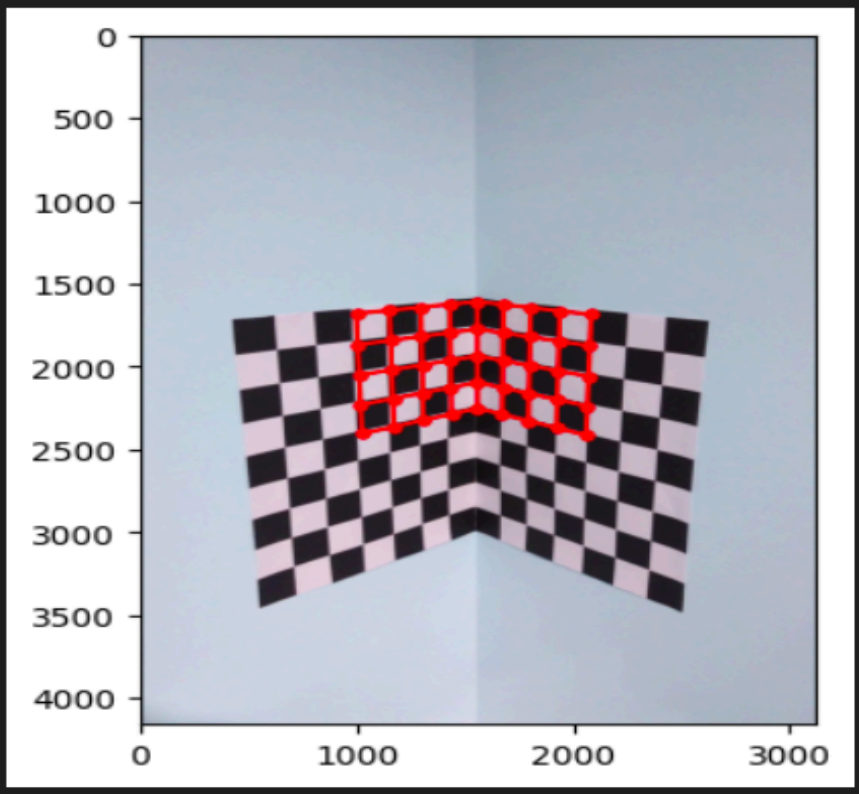
6. **Output:**

- Print the final camera matrix (mtx) and distortion coefficients (dist).
- Convert the rotation vector (rvecs) to a rotational matrix using cv2.Rodrigues.
- Print the rotational matrix (final_rotational) and translation vector (tvecs).
- Prints a subset of the projected wireframe points.
- Plots the wireframe on the image.

Results:

```
Final Camera Matrix:
[[3.77724227e+03 0.00000000e+00 1.62093325e+03]
 [0.00000000e+00 3.78338152e+03 1.98728913e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion Coefficients:
[[ 1.05849347e+00]
 [-1.04501193e+01]
 [-4.64803754e-03]
 [ 7.74114615e-03]
 [ 2.88394062e+01]]

Rotational Matrix:
[[ 0.69390918  0.00211712 -0.72005942]
 [ 0.17238117 -0.97140587  0.16326471]
 [-0.6991243  -0.23741557 -0.6744324 ]]

Translational Coefficients:
(array([[-10.50410288],
       [-63.11240283],
       [628.03270103]]),)
```

```
[[ 997.76434722 1680.32467401]
 [1151.64340943 1660.20034712]
 [1295.67648302 1641.3636801 ]
 [1430.77926923 1623.69491741]
 [1557.75733362 1607.08870706]]
```



Challenges:

- **Understanding Camera Calibration:** Camera calibration involves complex mathematical concepts like intrinsic and extrinsic parameters, distortion correction, and projection matrices. Understanding these concepts thoroughly can be challenging, especially for those new to computer vision.
- **Parameter Tuning:** Choosing appropriate initial guesses for camera matrices and distortion coefficients can significantly impact the calibration accuracy. Tuning these parameters effectively requires a good understanding of the camera model and calibration process.

Learnings:

- **Camera Calibration Fundamentals**: Gain a deeper understanding of the principles behind camera calibration, including intrinsic and extrinsic parameters, distortion models, and the calibration procedure.

2.2

Procedure

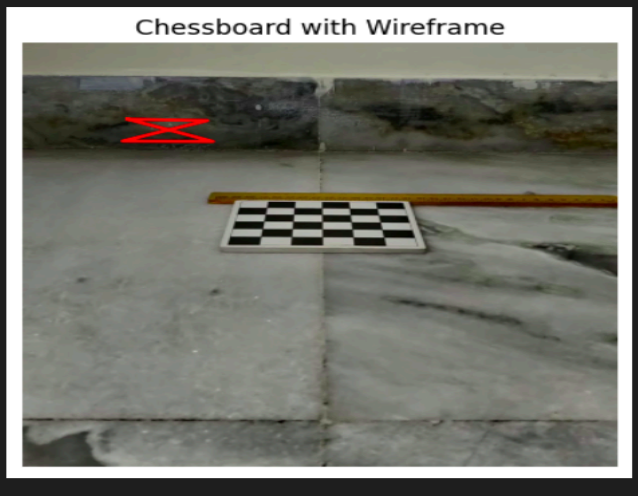
1. **Projecting 3D World Points:**
- The 3D world points (world_points) are projected onto the 2D image plane using the cv2.projectPoints function.
- The function takes the world points, rotation matrix (rotation_matrix), translation vector (tvecs[0]), camera matrix (mtx), and distortion coefficients (dist) as inputs.
- It returns the projected 2D image points (projected_image_points) and an optional parameter (here denoted by _, which is not used).
2. **Reshaping Projected Image Points:**
- The projected image points are reshaped to have two columns using np.squeeze and astype(int).
3. **Displaying Results:**
- The projected 2D image points are printed.
- The original image is loaded using plt.imread.
- The image and wireframe points are plotted using Matplotlib.
- The wireframe is drawn by connecting the projected image points in a closed loop.
4. **Plotting Wireframe on Image:**
- The wireframe is created by joining the projected image points in a closed loop.
- The wireframe and image are plotted together, and the plot is displayed using plt.show().

Result:

```
Camera Calibration Results:
===========================
Final Camera Matrix (mtx):
[[7.96640706e+03 0.00000000e+00 9.92458159e+02]
 [0.00000000e+00 1.80977153e+03 1.22027815e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion Coefficients (dist):
[[-1.02420975 -4.97583428 -0.21727943  0.02655579 10.79028169]]

Rotation Matrix (rotation_matrix):
[ 0.41432665  0.1225863  -0.90183481]

Translation Vector (tvecs):
[[ -5.75394761]
 [-29.18207928]
 [ 58.79276217]]
```

```
Projection Matrix:
[[ 4.16359372e+03  1.21435953e+03 -6.75562232e+03  1.25110676e+04]
 [ 5.74088448e+02  2.03536776e+03  5.40418647e+02  1.89306269e+04]
 [ 8.69456239e-01  2.39594145e-01  4.32019090e-01  5.87927622e+01]]
```

```
Projected 2D Image Points:
==========================
[[374 379]
 [656 386]
 [356 488]
 [676 490]]
```



Challenges:

- **Understanding Projection:** Understanding how 3D world points are projected onto a 2D image plane can be challenging, especially for those new to computer vision. It involves comprehending concepts such as camera calibration, extrinsic and intrinsic parameters, and perspective projection.
- **Data Formatting:** Properly formatting the input data (e.g., world points, rotation matrix, translation vector, camera matrix, distortion coefficients) for the projection function can be challenging. Any inconsistencies or errors in the data can result in incorrect projections.

Learnings:

- **Projection Fundamentals:** Gain a deeper understanding of how 3D points are projected onto a 2D image plane in the context of camera calibration and computer vision.
-

2.3

Procedure:

1. **Extrinsic Transformation Matrix Calculation:**

- Given a rotation matrix and translation vector (rotation_matrix and tvecs), we combine them into a single transformation matrix (extrinsic_matrix). This matrix represents the extrinsic parameters of a camera, which includes both rotation and translation.

2. **Projection of World Origin onto Image Plane**:

- We have a world coordinate system where the origin is at [0, 0, 0]. This point is represented in homogeneous coordinates (world_origin).
- Using the transformation matrix (transformation_matrix), we project the world origin onto the image plane to obtain its coordinates in the image (image_coordinates).

- The resulting coordinates are in homogeneous form, so we convert them to 2D coordinates by dividing by the third component.

3. **Calculation of Euclidean Distance:**

- Two points (point1 and point2) in a 2D space are defined.
- The Euclidean distance between these two points is calculated using the formula distance = sqrt((x2 - x1)^2 + (y2 - y1)^2).

4. **Displaying Results:**

- The calculated image coordinates of the world origin are printed.
- The Euclidean distance between point1 and point2 is printed.

Results:

```
Image Coordinates of the World Origin:
[[212.79945253]
 [321.98907213]]
```

```
Euclidean distance between the two points: 1.26156102040516
```

- Its very close values. Its near to my observations.

Challenges:

- Understanding Coordinate Systems: Working with different coordinate systems (e.g., world coordinates, image coordinates) and understanding their transformations can be challenging, especially for beginners.

Learnings:

- Transformation Matrices: Understanding how to represent geometric transformations (rotation and translation) using transformation matrices is fundamental in computer vision and graphics.
- Euclidean Distance: Learning how to compute distances between points in Euclidean space and its significance in various applications, such as object tracking and shape analysis.

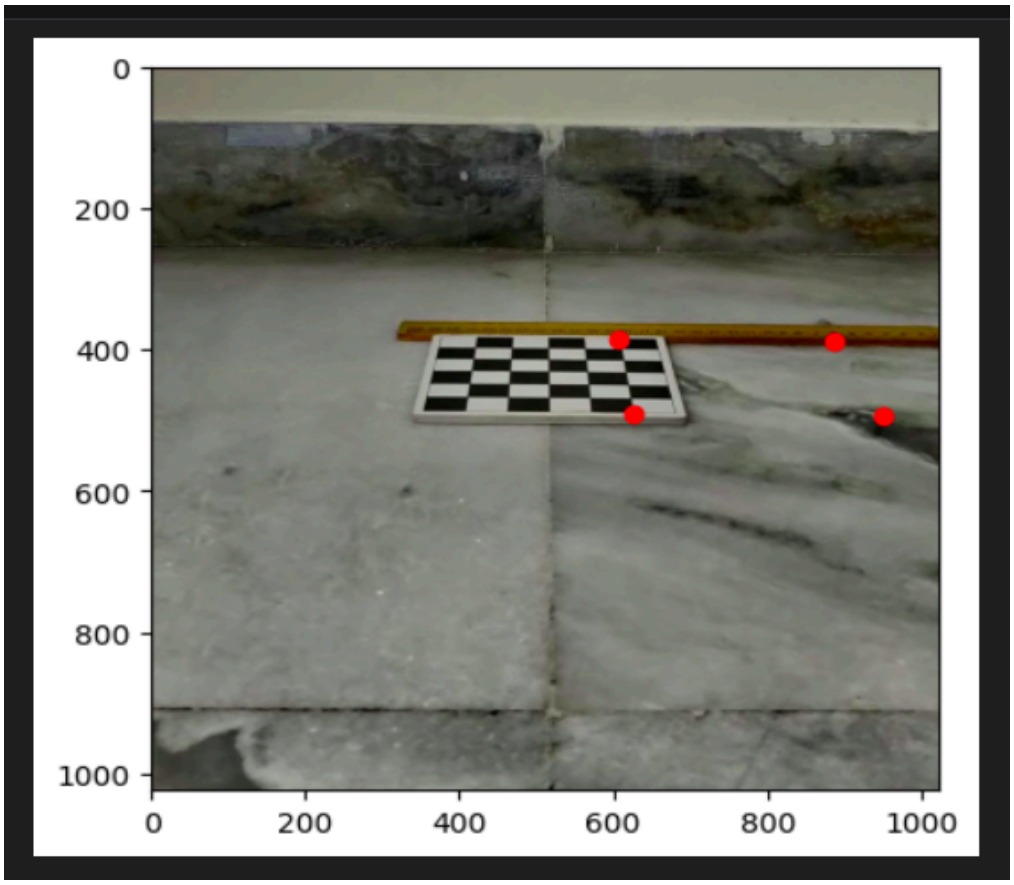
Question 3

3.1

Procedure:

First, I define a matrix world_points representing coordinates of points in a 3D world. It transposes this matrix and performs a matrix multiplication with another matrix P. The resulting points are then transposed and converted to Cartesian coordinates. These Cartesian coordinates are then plotted on an image loaded from the file 'assign1.jpg'. The image is displayed with the plotted points superimposed, showcasing the mapping of the 3D world points onto the 2D image plane.

Result:



Challenges:

- **Camera Calibration:** Obtaining accurate camera parameters through calibration can be challenging, especially ensuring precise measurements and proper calibration setup.
- **Coordinate Systems:** Understanding and managing different coordinate systems (e.g., world coordinates, camera coordinates, image coordinates) can be confusing, especially when transforming points between them.

Learnings:

- **Camera Calibration:** Learned about the importance of camera calibration for obtaining accurate intrinsic and extrinsic camera parameters.
- **Coordinate Systems:** Gained a deeper understanding of different coordinate systems involved in computer vision applications and how to transform points between them.

3.2

Procedure:

I perform a 3D to 2D projection of world points onto an image plane using camera parameters like rotation vector (rvec), translation vector (tvec), camera matrix (mtx), and distortion coefficients (dist). It first defines the world points representing corners of a chessboard. Then, it projects these points onto the 2D image plane using OpenCV's projectPoints function. Afterward, it converts the projected points to integer format and plots them on the provided image (assign1.jpg). Additionally, it draws lines to connect the projected points, creating a wireframe representation of the chessboard. Finally, it displays the image with the overlaid wireframe.

Challenges and Learnings:

One challenge is understanding the principles of camera calibration and 3D-to-2D projection, including concepts like intrinsic and extrinsic

camera parameters, distortion correction, and perspective transformation. Learning about these concepts provides a deeper understanding of how cameras capture the real world and how to accurately represent 3D objects in 2D images. Another challenge is implementing the projection process using libraries like OpenCV, which requires understanding the functions and their parameters. This involves learning about the OpenCV library's documentation, its functions, and how to correctly use them for various computer vision tasks. Additionally, interpreting and visualizing the results require skills in data manipulation, visualization, and interpretation. Understanding how to plot points and lines on an image helps in visualizing the projected points and validating the accuracy of the projection. Overall, the process involves a combination of theoretical understanding, practical implementation, and visualization skills, providing valuable learning experiences in computer vision and image processing.