Exercise 3

Yiwei Ye

1. Make a plot with the view of the points of the circle projected into the camera plane.

My interpretation:

- Each column likely represents a single point on the circle, with the rows corresponding to the *x*, *y*, and *z* coordinates.
- It appears that the first row might be *y* coordinates, the second row might be *z* coordinates, and the column headers might actually be the *x* coordinates.

Reconstruct the Data:

In Phython:

Load the circle points from A1.csv

circle_points = pd.read_csv('/mnt/data/A1.csv')

Display the first few rows to understand the structure of the data

circle_points.head()

Result

```
   6.9063  6.9018  6.8882  6.8658  6.8348  6.7954   6.748  6.6932  6.6314  \
0  6.0000  6.0998  6.1987  6.2955  6.3894  6.4794  6.5646  6.6442  6.7174
1  1.5774  1.5795  1.5858  1.5963  1.6107  1.6291  1.6512  1.6768  1.7056

   6.5634  ...  6.5024  6.5752  6.6423  6.7029  6.7565  6.8026  6.8406  \
0  6.7833  ...  5.1677  5.2272  5.2945  5.3687  5.4493  5.5354  5.6261
1  1.7373  ...  1.7657  1.7318  1.7005  1.6722  1.6472  1.6258  1.6080

   6.8702  6.8911  6.9032
0  5.7206  5.8178  5.9169
1  1.5942  1.5845  1.5788

[2 rows x 63 columns]
```

and then:

Next, I'll calculate the camera's orientation based on the specified rotations. Following that, we'll project the circle points onto the camera plane, taking into account the camera's focal length of 1/34 meters. Once the projection calculations are complete, we'll produce the required plots.

It appears that the data is transposed. The first row (which is currently the header) should be the x-coordinates.

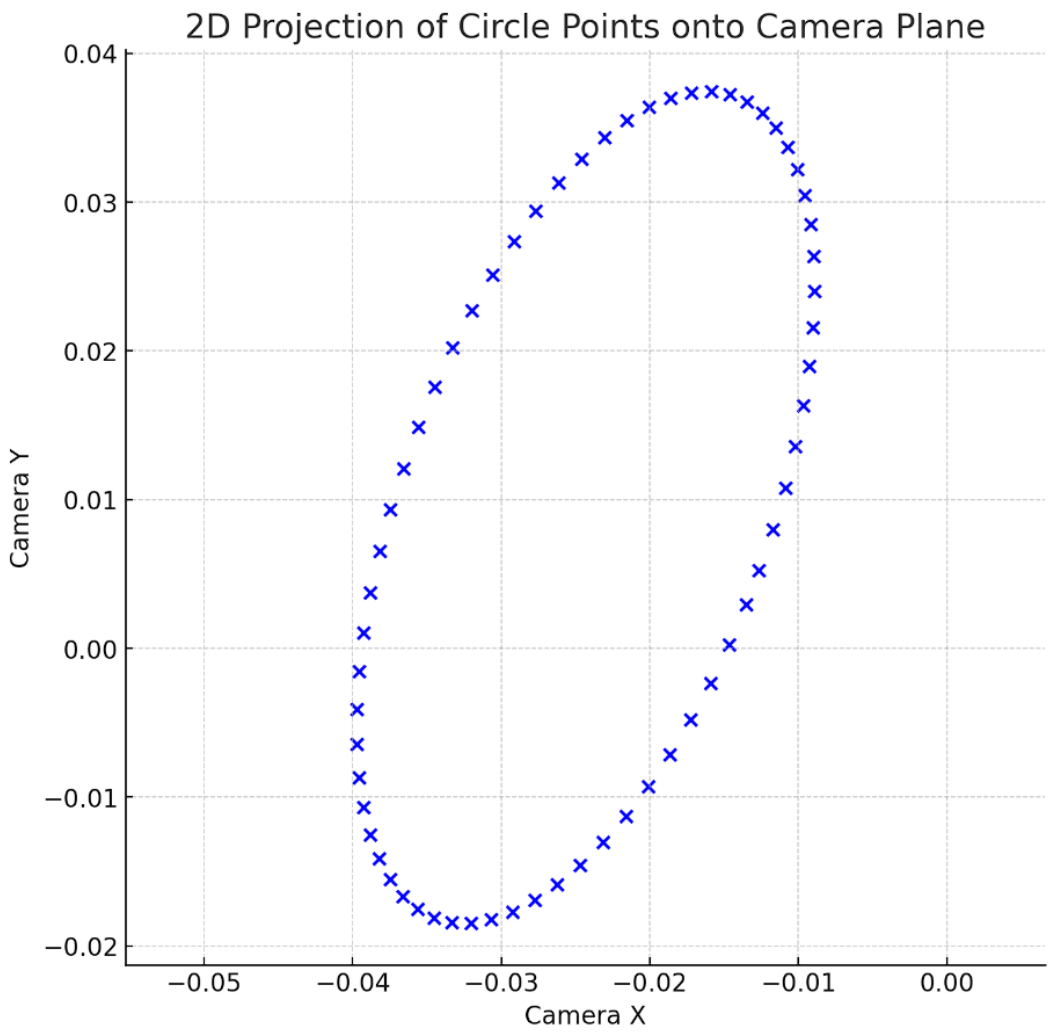
Code:

Let's fix the DataFrame by setting the header as the first row and then transpose it.

```
# Reset the header of the DataFrame and then transpose it

circle_points = pd.read_csv('/mnt/data/A1.csv', header=None).T

circle_points.columns = ['x', 'y', 'z']


# Display the corrected data

circle_points.head()
```

Result

```
        x       y       z
0  6.9063  6.0000  1.5774
1  6.9018  6.0998  1.5795
2  6.8882  6.1987  1.5858
3  6.8658  6.2955  1.5963
4  6.8348  6.3894  1.6107
```



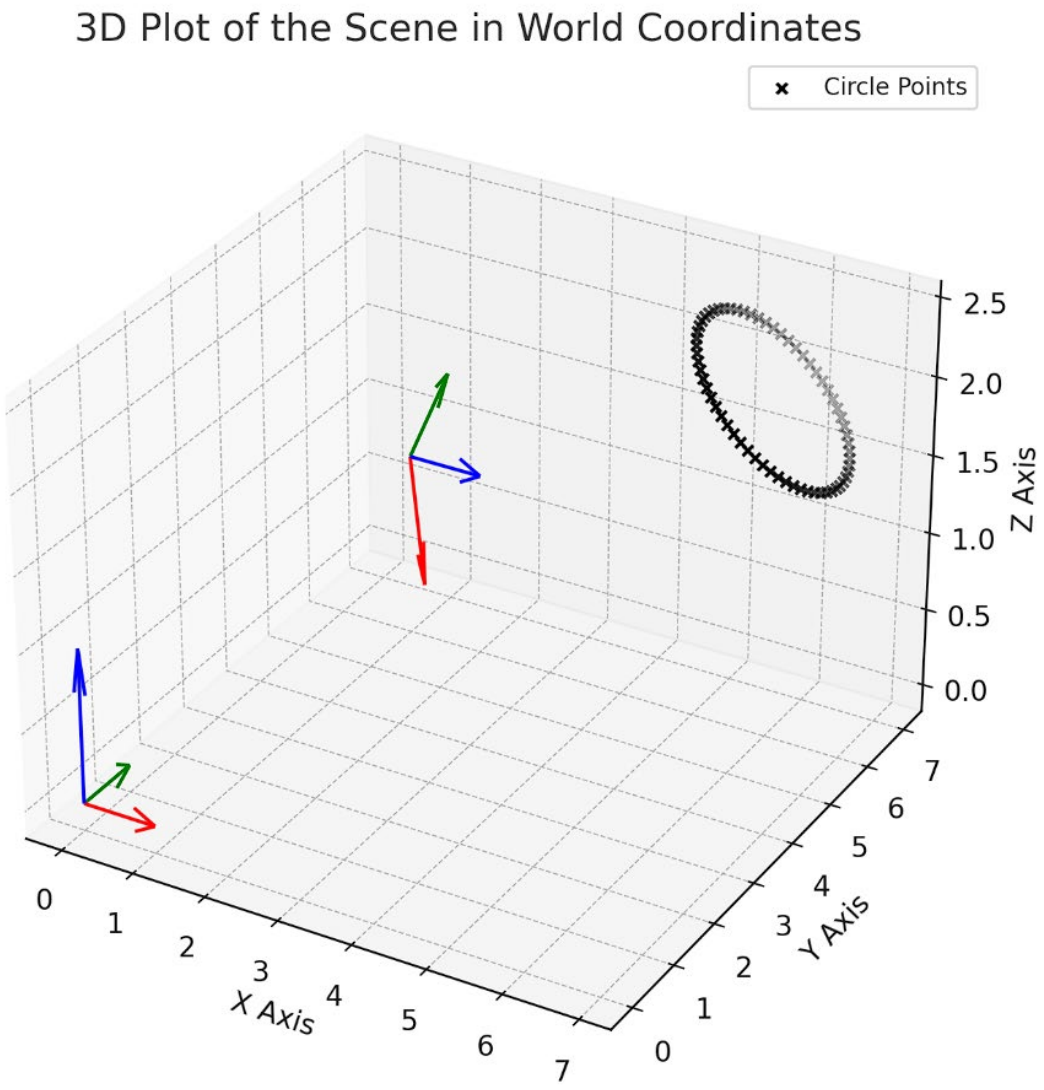
Now, let's move on to creating the 3D plot where the entire scene is drawn in world coordinates, including the two reference frames and the circle points. We will visualize the world frame, the camera frame, and the circle in 3D space. Let's create the 3D plot next.

Code for creation:

```
# Camera specifications
camera_position = np.array([1, 6, 1])  # The camera position cc in world frame W
focal_length = 1 / 34  # The focal length of the camera

# The camera's orientation is achieved by two rotations:
# -90 degrees about the world frame's y-axis (yaw)
# -20 degrees about the resulting z-axis (pitch)
rotation_yaw = R.from_euler('y', -90, degrees=True).as_matrix()
rotation_pitch = R.from_euler('z', -20, degrees=True).as_matrix()

# The overall rotation from the world frame to the camera frame
rotation_world_to_camera = rotation_pitch @ rotation_yaw

# The translation from the world frame to the camera frame is simply the negative of the camera position
translation_world_to_camera = -camera_position

# We will use a pinhole camera model for projection. The projection matrix can be written as:
projection_matrix = np.array([
    [focal_length, 0, 0],
    [0, focal_length, 0],
    [0, 0, 1]
])

# Now we will project the points onto the camera plane.
# We need to transform the points from world coordinates to camera coordinates first.
# This involves applying the rotation and then the translation.
```

```
# Convert the DataFrame to a numpy array for calculations
circle_points_np = circle_points.to_numpy()

# Add a column of 1's to the circle points for affine transformations
circle_points_homogeneous = np.hstack((circle_points_np, np.ones((circle_points_np.shape[0], 1))))

# Create the affine transformation matrix for world to camera
affine_world_to_camera = np.eye(4)
affine_world_to_camera[:3, :3] = rotation_world_to_camera
affine_world_to_camera[:3, 3] = translation_world_to_camera

# Apply the transformation to get camera coordinates
circle_points_camera_coords = (affine_world_to_camera @ circle_points_homogeneous.T).T[:, :3]

# Project the points onto the camera plane using the projection matrix
circle_points_projected = (projection_matrix @ circle_points_camera_coords.T).T

# For the plot, we can ignore the z-coordinate since we're projecting onto a 2D plane
circle_points_2d = circle_points_projected[:, :2]

# Plot the 2D projection of the circle points
plt.figure(figsize=(8, 8))
plt.scatter(circle_points_2d[:, 0], circle_points_2d[:, 1], c='blue')
plt.title('2D Projection of Circle Points onto Camera Plane')
plt.xlabel('Camera X')
plt.ylabel('Camera Y')
plt.grid(True)
plt.axis('equal')  # Ensuring the aspect ratio is equal to preserve the circle shape
plt.show()
```

We will create the 3D plot in the next cell to avoid overcrowding this one with code.



The 3D plot above displays the scene in world coordinates:

- The black points represent the circle in the world frame.
- The colored arrows originating from the world origin (0, 0, 0) represent the world frame axes, with red for the x-axis, green for the y-axis, and blue for the z-axis.
- The colored arrows originating from the camera position (1, 6, 1) represent the camera frame axes after applying the specified rotations, using the same color scheme.