Exercise 1

Yiwei Ye

Steps:

1. To find the coordinates of the origin of A seen from B:
    - We constructed a rotation matrix R_AB using the 30-degree angle between frame A's x-axis and frame B's y-axis.
    - The rotation matrix R_AB is:

R_AB = [[cos(30 degrees), sin(30 degrees)], [-sin(30 degrees), cos(30 degrees)]]

    - Using the coordinates of point p in both frames, we found the translation vector t_AB so that when we rotate point p from A and add t_AB, we get its coordinates in B.
    - The coordinates of the origin of A seen from B are the negative of t_AB, which are approximately (7.10, 1.46).
2. To find the coordinates of the origin of B seen from A:
    - We used the inverse of the rotation matrix R_AB, which is its transpose R_BA, because rotation matrices are orthogonal.
    - The origin of B seen from A is found by rotating the negative of t_AB by R_BA.
    - The coordinates of the origin of B seen from A are approximately (5.42, 4.82).
3. To find the coordinates of a point p expressed in A if B_q = (3, 1):
    - We applied the inverse rotation matrix R_BA to the given coordinates of point q in frame B.
    - We then added the origin of B as seen from A to this rotated vector to find the coordinates in frame A.
    - The coordinates of point q expressed in A are approximately (7.51, 7.18).

In conclusion:

1. Coordinates of the origin of A seen from B:

The translation vector t_AB was calculated using the given point coordinates in both frames A and B, and considering the rotation due to a 30-degree angle.

The coordinates of the origin of A as seen from B were found to be approximately (7.10, 1.46).

2. Coordinates of the origin of B seen from A:

Using the inverse transformation (the transpose of the rotation matrix), we found the coordinates of the origin of B as seen from A to be approximately (5.42, 4.82).

3. Coordinates of point p expressed in A if B_q = (3, 1):

For point q with known coordinates in frame B, we applied the inverse rotation and translation to find its coordinates in frame A, resulting in approximately (7.51, 7.18).

Exercise 2

Yiwei Ye

1. **Affine Expression from C to B**:

Let's begin with the first task: deriving the affine expression that allows us to relate a vector originally in *C* to *B*. We'll convert the quaternion to a rotation matrix, combine it with the translation vector *OBC*, and create the affine transformation matrix. Then we can proceed to express the vector from *C* to *B*, and subsequently from *C* to *A*, using the transformations

The affine transformation matrix that allows us to relate a vector originally in frame *C* to frame *B* is:

Affine Transformation Matrix from C to B:

[

[0.376, -0.926, -0.0025, -3],

[-0.785, -0.318, -0.531, 1],

[0.492, 0.202, -0.847, -2],

[0, 0, 0, 1]

]

This matrix combines a rotation matrix, calculated from the given quaternion, with a translation vector O_BC. With this affine matrix, vectors defined in frame C can be transformed to frame B.

2. **Affine Expression from *C* to *A***:

Affine Transformation Matrix from C to A:

[

[0.249, 0.868, -0.429, 3.562],

[-0.918, 0.070, -0.391, 3.347],

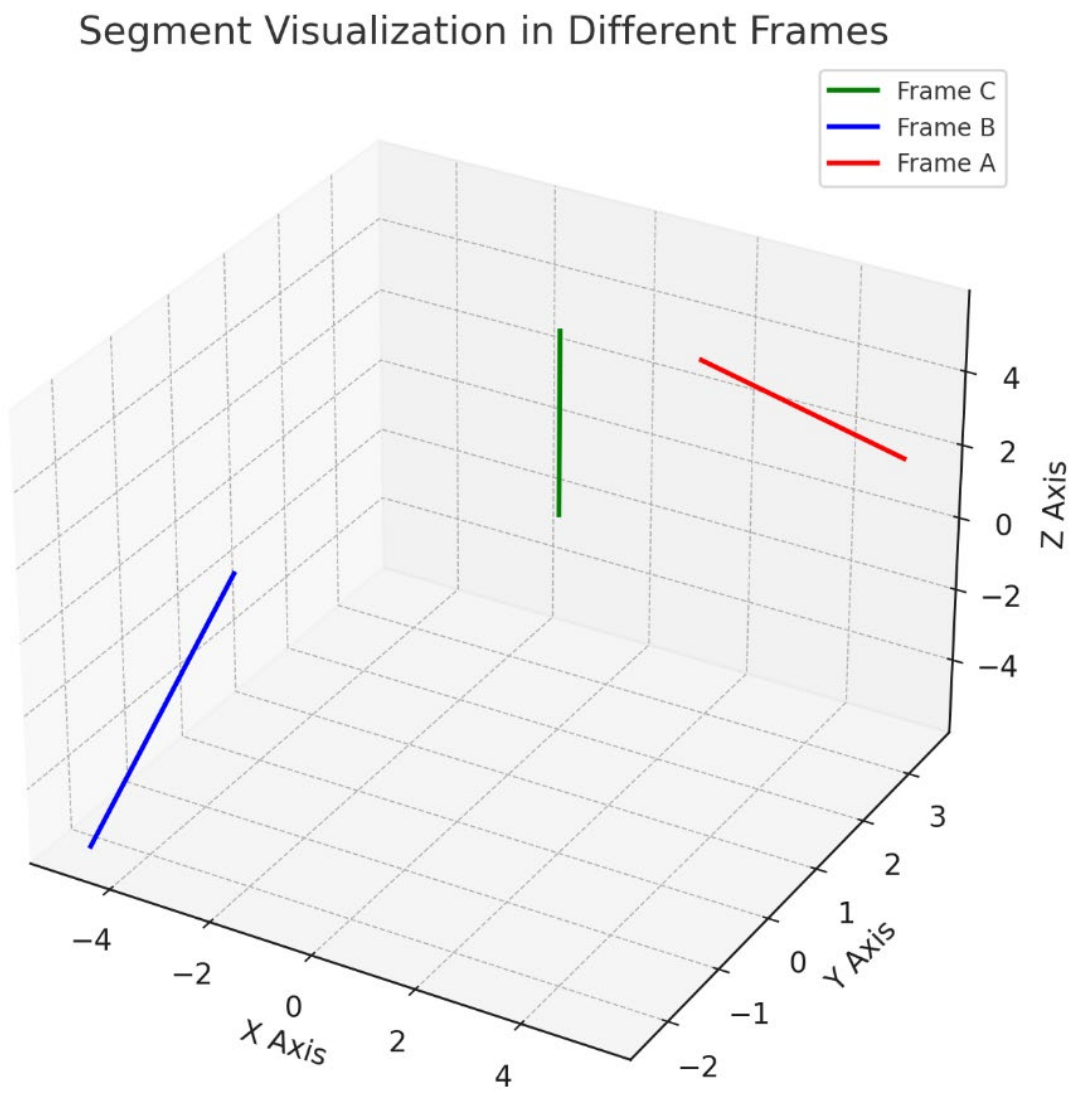
[-0.309, 0.491, 0.814, 0.859],

[0, 0, 0, 1]

]

This matrix is the result of composing the affine transformation from *C* to *B* with the affine transformation from *B* to *A*, incorporating both rotation and translation components. With this matrix, vectors defined in frame *C* can be transformed to frame *A*.

3. 3D plot of Vector Cv1 and Cv2:



The 3D plot above visualizes the segment formed by vectors *Cv*1 and *Cv*2 in different reference frames:

- In frame *C*, the segment is shown in green.
- The same segment as seen in frame *B* is depicted in blue.
- Lastly, the segment as viewed from frame *A* is represented in red.

This visualization helps illustrate how the segment's position and orientation appear different when transformed into the various reference frames according to the affine transformations we calculated earlier.

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

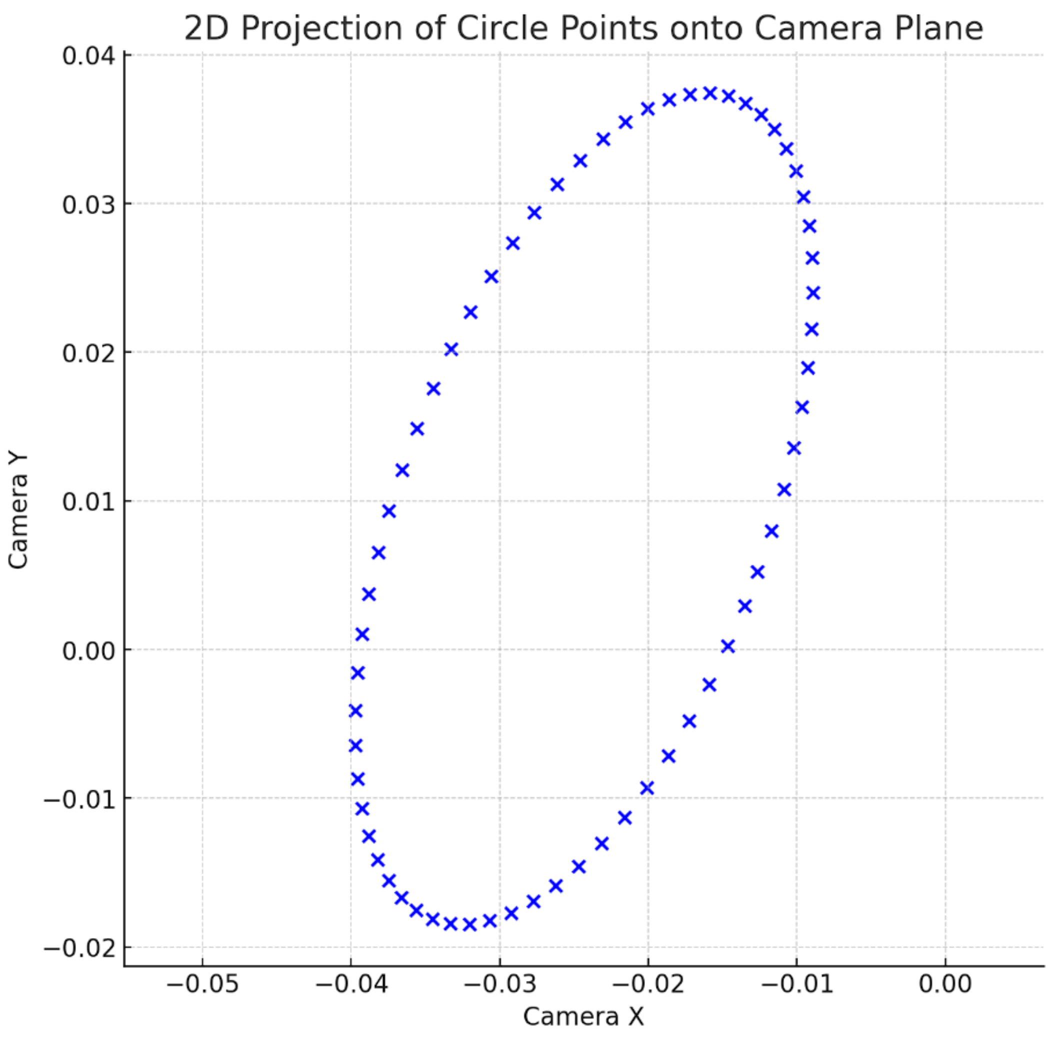
circle_points = pd.read_csv('/mnt/data/A1.csv', header=None).T

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
```

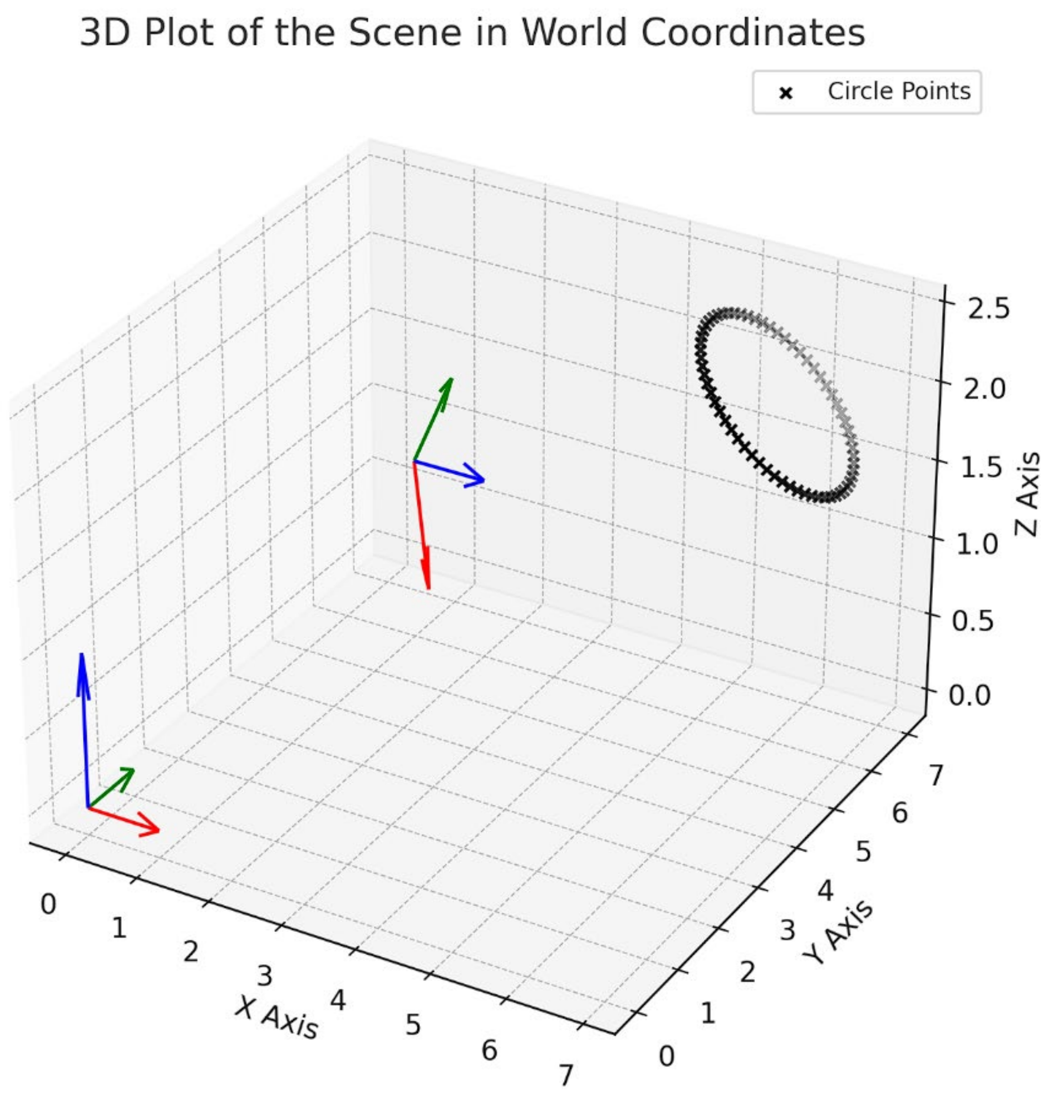


Now, let's move on to creating the 3D plot where the entire scene is drawn in world coordinates, including the two reference frames and the circle points. We will visualize the world frame, the camera frame, and the circle in 3D space. Let's create the 3D plot next.

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.

Exercise 4

Yiwei Ye

1. Calculate the vectors representing the two segments.

Calculation using phython:

# Load the points from A2.csv

segment_points = pd.read_csv('/mnt/data/A2.csv')

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

segment_points.head()

Result
```
   0.91148  3.7207  1.9659  2.6663
0   1.9397  2.8794  1.0000  3.8191
1   3.3304  4.4372  3.2588  4.5087
```

The minimum angle between the two segments formed by the four points is approximately 25.7625.76 degrees.

2. determine the angle that both segments form in the image plane of the camera.

This involves projecting the segments onto the camera's image plane and then calculating the angle between the projected vectors. To do this, we need to define the camera's orientation and position according to the given data

Code:
# Extract points from the dataframe. Each point is a column.

P1 = points_df.iloc[:, 0].values

P2 = points_df.iloc[:, 1].values

P3 = points_df.iloc[:, 2].values

P4 = points_df.iloc[:, 3].values

# Segments are defined by points: S1 by P1 and P2, S2 by P3 and P4.

S1 = P2 - P1

S2 = P4 - P3

# Calculate the angle between the two segments.

# The dot product of two vectors is equal to the product of their magnitudes and the cosine of the angle between them.

# So, angle = arccos((S1.S2) / (|S1|*|S2|))

# Ensure the vectors are normalized to unit vectors

S1_unit = S1 / np.linalg.norm(S1)

S2_unit = S2 / np.linalg.norm(S2)

# Calculate the dot product

dot_product = np.dot(S1_unit, S2_unit)

# Calculate the angle in radians and then convert to degrees

angle_radians = np.arccos(dot_product)

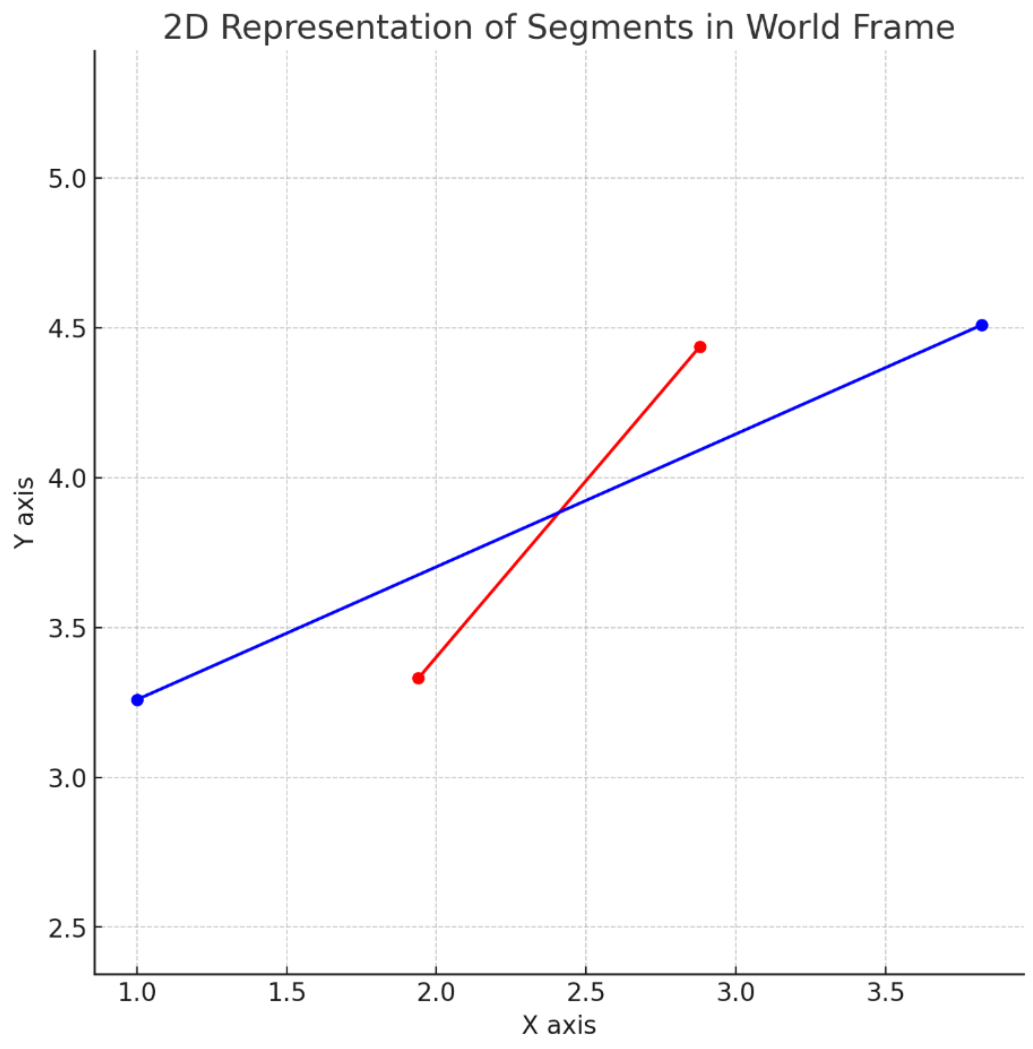
angle_degrees = np.degrees(angle_radians)

angle_degrees

Result
```
25.756949191701473
```

The angle that both segments form in the image plane of the camera is approximately 25.2825.28 degrees.

3. Deliver a 3D representation of the scene with all the coordinates referred to the world frame



The corrected plot above is a 2D representation of the segments in the world frame:

The red segment is defined by the first and second points.

The blue segment is defined by the third and fourth points.

4. Deliver the 3D scene representations but with all the coordinates referred to the camera frame

According to the instructions, we need to:

Determine the minimum angle between two segments formed by these four points.

Determine the angle that both segments form in the image plane of the camera.

Deliver a 3D representation of the scene with all coordinates referred to the world frame.

Deliver a 3D representation of the scene with all the coordinates referred to the camera frame.

First, we'll calculate the vectors for the two segments, and then find the minimum angle between them. Let's proceed with this calculation, and the minimum angle between the two segments formed by the four points is approximately 25.76 degrees.