Q4)

i)

Estimated Intrinsic Camera Parameters:
Focal Length (fx): 2405.1291687250086
Focal Length (fy): 2513.8427501252777
Skew Parameter: 0.0
Principal Point (cx): 145.57576139921184
Principal Point (cy): 382.4055697557592

Calibration Error Estimates:
Reprojection Error: 2.958076002188282

ii) Values of the rotation matrix and translation vectors stored in separate text files for each image (q4_dataset\per_image_extrinsic_parameters).

Estimated Extrinsic Camera Parameters (Rotation Matrix and Translation Vector) for each image:

Image: 1.jpg
Rotation Matrix:
[[0.58733259 0.0249065 -0.80896235]
 [0.02320434 0.99859729 0.04759211]
 [0.80901297 -0.04672384 0.5859308]]
Translation Vector:
[[0.91295384]
 [-2.41349572]
 [30.8119782]]

Image: 10.jpg
Rotation Matrix:
[[0.26608634 -0.03350616 0.9633667]
 [-0.07694515 0.99546843 0.05587527]
 [-0.9608733 -0.08899404 0.26230242]]
Translation Vector:
[[0.25989674]
 [-1.1206529]
 [34.5925466]]

Image: 11.jpg
Rotation Matrix:
[[0.80155592 -0.00838205 -0.59786106]
 [-0.00922487 0.99960936 -0.02638242]
 [0.59784865 0.02666218 0.80116548]]
Translation Vector:
[[0.71515653]
 [-4.43616638]
 [46.87083174]]

Image: 12.jpg

Rotation Matrix:
[[0.88443535 -0.02774741 -0.46583709]
 [0.01309927 0.99931353 -0.03465355]
 [0.46647885 0.0245467   0.88419169]]
Translation Vector:
[[1.84209127]
 [-4.58859176]
 [49.83198399]]

Image: 13.jpg
Rotation Matrix:
[[9.17377120e-01 -6.32500015e-04 3.98018617e-01]
 [8.07032203e-04 9.99999638e-01 -2.70974996e-04]
 [-3.98018301e-01 5.69800103e-04 9.17377298e-01]]
Translation Vector:
[[1.30276344]
 [-5.92235512]
 [59.50441575]]

Image: 14.jpg
Rotation Matrix:
[[0.49490939 0.02296185 -0.86864115]
 [0.00233082 0.99961212 0.02775195]
 [0.86894146 -0.01575935 0.49466391]]
Translation Vector:
[[0.5114089]
 [-4.95469251]
 [57.6249977]]

Image: 15.jpg
Rotation Matrix:
[[0.91154555 -0.02413752 0.41049006]
 [0.01272786 0.99945355 0.03050573]
 [-0.41100209 -0.02258271 0.91135465]]
Translation Vector:
[[0.29494125]
 [-4.45434261]
 [51.26675474]]

Image: 16.jpg
Rotation Matrix:
[[0.96065944 -0.05776571 0.27165524]
 [0.0178074   0.98892907 0.14731663]
 [-0.27715761 -0.13668364 0.9510527]]
Translation Vector:
[[1.42193794]
 [-3.95852162]
 [48.50472348]]

Image: 17.jpg
Rotation Matrix:
[[0.9889194 -0.02094399 0.14696863]
 [-0.0169592   0.96757838 0.25200091]
 [-0.14748157 -0.25170106 0.95650184]]

Translation Vector:
[[0.78546977]
 [-3.88817746]
 [47.9776804]]

Image: 18.jpg
Rotation Matrix:
[[0.86138503 -0.00935387 0.50786646]
 [-0.01019146 0.99931091 0.03569085]
 [-0.50785034 -0.03591946 0.86069613]]
Translation Vector:
[[-1.31684606e-02]
 [-4.26739426e+00]
 [5.02400623e+01]]

Image: 19.jpg
Rotation Matrix:
[[0.63611762 -0.03140736 0.77095262]
 [-0.00643724 0.99892043 0.0460058]
 [-0.77156525 -0.03422791 0.63522871]]
Translation Vector:
[[2.66012133e-02]
 [-3.80071870e+00]
 [5.38808293e+01]]

Image: 2.jpg
Rotation Matrix:
[[0.99474834 0.03075706 0.09762042]
 [-0.00140208 0.95778552 -0.28748033]
 [-0.10234148 0.28583371 0.95279867]]
Translation Vector:
[[0.90977874]
 [-3.42961467]
 [35.57642002]]

Image: 20.jpg
Rotation Matrix:
[[0.69782278 0.02782257 -0.71572989]
 [-0.00636382 0.99944669 0.03264689]
 [0.71624219 -0.01822697 0.69761372]]
Translation Vector:
[[-0.16893056]
 [-6.5857787]
 [63.31548403]]

Image: 21.jpg
Rotation Matrix:
[[0.82840203 -0.01181299 -0.5600094]
 [0.00110913 0.99981022 -0.01944957]
 [0.56013288 0.01549094 0.82825792]]
Translation Vector:
[[-1.4429944]
 [-11.89822769]
 [99.91381733]]

Image: 22.jpg
Rotation Matrix:
[[9.99883826e-01 1.52227488e-02 7.76518952e-04]
 [-1.50305133e-02 9.76227255e-01 2.16227730e-01]
 [2.53352144e-03 -2.16214281e-01 9.76342648e-01]]
Translation Vector:
[[-0.20750752]
 [-6.12946717]
 [80.29037918]]

Image: 23.jpg
Rotation Matrix:
[[0.99513879 0.0490664 -0.08538902]
 [-0.03033099 0.97760132 0.20826831]
 [0.0936954 -0.20466594 0.97433722]]
Translation Vector:
[[0.49253077]
 [-7.97893646]
 [78.99130088]]

Image: 24.jpg
Rotation Matrix:
[[0.99983055 0.01787374 -0.00440483]
 [-0.01623153 0.96885878 0.24708138]
 [0.00868392 -0.24696801 0.96898472]]
Translation Vector:
[[-0.11227616]
 [-9.6741807]
 [82.83651553]]

Image: 25.jpg
Rotation Matrix:
[[0.81648388 0.02845323 0.5766667]
 [-0.00739903 0.99921858 -0.03882625]
 [-0.57732081 0.02743423 0.8160564]]
Translation Vector:
[[-1.62658373]
 [-9.58954935]
 [91.70044369]]

Image: 26.jpg
Rotation Matrix:
[[0.83570469 -0.00423114 0.54916279]
 [0.01932103 0.9995778 -0.0217009]
 [-0.54883911 0.02874593 0.8354336]]
Translation Vector:
[[-1.88221365]
 [-8.51207211]
 [82.3775885]]

Image: 27.jpg
Rotation Matrix:
[[0.68262906 -0.00288869 0.73075934]

[-0.01035887 0.99985346 0.01362901]
 [-0.73069163 -0.0168734   0.68249911]]
Translation Vector:
[[-1.36448977]
 [-7.64909713]
 [83.06741145]]

Image: 28.jpg
Rotation Matrix:
[[0.81661862 -0.0105613   0.577081]
 [0.02793577 0.99938401 -0.02124152]
 [-0.57650119 0.03346742 0.81641063]]
Translation Vector:
[[3.10498898]
 [-9.32903396]
 [81.69020962]]

Image: 29.jpg
Rotation Matrix:
[[0.99895595 -0.0253539 -0.03800244]
 [0.04490107 0.69823308 0.71446096]
 [0.00842019 -0.71542138 0.69864251]]
Translation Vector:
[[2.63775588]
 [-1.76179687]
 [31.08277723]]

Image: 3.jpg
Rotation Matrix:
[[0.3980444 -0.03939841 0.91651973]
 [-0.0531188   0.99641121 0.06590219]
 [-0.91582698 -0.07491643 0.3945231]]
Translation Vector:
[[-0.41000719]
 [-1.60845064]
 [34.52345409]]

Image: 30.jpg
Rotation Matrix:
[[0.99387354 0.09272514 0.06014505]
 [-0.03618279 0.78717237 -0.61567075]
 [-0.10443268 0.60972265 0.78570484]]
Translation Vector:
[[1.74698757]
 [-0.27066461]
 [25.47794054]]

Image: 4.jpg
Rotation Matrix:
[[0.28003914 -0.00358431 0.95998189]
 [-0.05563822 0.99825151 0.01995758]
 [-0.95837491 -0.05900058 0.27935007]]
Translation Vector:
[[-0.54611436]

[-1.86431115]
[34.59794135]]

Image: 5.jpg
Rotation Matrix:
[[0.55499572 -0.02963734 0.83132507]
 [-0.04002649 0.9972561   0.06227474]
 [-0.83088966 -0.06783724 0.55228659]]
Translation Vector:
[[0.39493784]
 [-1.92348978]
 [36.4866853]]

Image: 6.jpg
Rotation Matrix:
[[0.58790517 0.01991125 -0.80868477]
 [-0.01037372 0.99980039 0.01707527]
 [0.80886334 -0.00164958 0.58799437]]
Translation Vector:
[[-0.07357859]
 [-3.76512642]
 [44.62331838]]

Image: 7.jpg
Rotation Matrix:
[[0.99627886 0.02980821 0.08086964]
 [-0.00596942 0.95990123 -0.28027487]
 [-0.08598136 0.27874919 0.95650724]]
Translation Vector:
[[0.77642987]
 [-4.60419744]
 [47.67227792]]

Image: 8.jpg
Rotation Matrix:
[[0.84667584 0.01574885 -0.53187592]
 [-0.01227393 0.99987399 0.01006781]
 [0.53196745 -0.00199597 0.84676245]]
Translation Vector:
[[1.37441611e-02]
 [-4.39101290e+00]
 [4.62716839e+01]]

Image: 9.jpg
Rotation Matrix:
[[0.37487053 0.02236872 -0.92680727]
 [0.03760898 0.99851899 0.0393114]
 [0.92631401 -0.04959296 0.37347408]]
Translation Vector:
[[0.11733522]
 [-2.07058475]
 [29.25983739]]

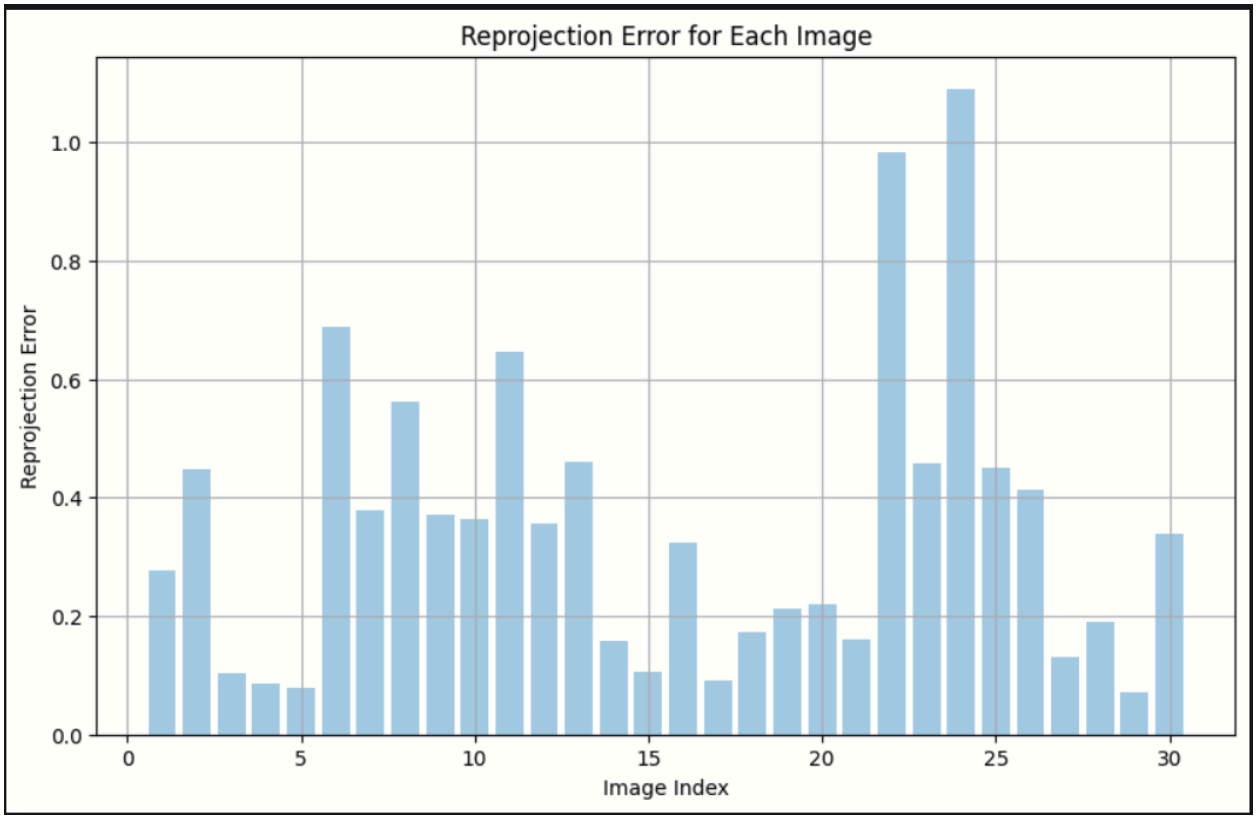
iii)

Estimated Radial Distortion Coefficients: [[5.61266206e-01 -2.79864931e+01 1.52101063e-02 1.20389566e-02
  1.47216665e+02]]

All the raw images were undistorted and stored in a separate directory (q4_dataset\undistorted_images)

After undistorting the images, the straight lines at the corner of the images become curved. This effect is especially apparent for images 5,13,15,16,17 and 18.

iv)




```
Mean Reprojection Error: 0.34632004720324583
Standard Deviation of Reprojection Error: 0.2497168816265701
```

v)

All the images with detected corner and reprojected points are stored in a separate directory in the submitted zip folder (q4_dataset\with_corners) .

Re-projection error is computed by projecting 3D object points onto the image plane using estimated camera parameters, including the camera matrix, distortion coefficients, rotation vector, and translation vector. This mimics the imaging process, producing anticipated image coordinates based on the estimated parameters. The re-projected image coordinates are then compared with the actual detected image coordinates of the object points. The re-projection error is quantified as the L2 norm (Euclidean distance) between these re-projected and detected image coordinates, normalized by the number of object points. A lower re-projection error indicates that the estimated camera parameters effectively model the imaging process, accurately mapping 3D object points onto the image plane with minimal deviation from observed coordinates. Conversely, a higher re-projection error suggests potential inaccuracies in estimated camera parameters or other factors such as noise or imprecise detected image coordinates contributing to the discrepancy.

vi)

Plane normals for each image calculated and stored in a separate text file in the submitted zip folder (q4_dataset\board_plane_normals_camera_coordinate_frame.txt) .

Q5)
i)

```
def compute_plane_normals_and_offsets(points):
    centroids = np.mean(points, axis=0)

    # point_T = np.transpose(points)

    centered_points = points - centroids

    # centered_points_T = np.transpose(centered_points)

    u , s , v = np.linalg.svd(centered_points)

    normal = v[-1,:]
    normal = normal/np.linalg.norm(normal)


    offset = np.dot(normal, centroids)

    if(offset < 0) :
        offset = -1*offset
        normal = -1*normal


    return normal, offset
```

Normal and offsets stored in separate directories for each img

ii)

1. Reprojection Error:

Let P_c be a point observed by the camera and P_l be the corresponding point observed by the laser (both in their respective frames). We want a transformation (R1, t1) to bring P_l to the camera frame and align it with P_c.

The reprojection error (e) can be expressed as the difference between the transformed laser point ($R_1P\_l + t_1$) and the camera point (P_c):

$e = P\_c - (R_1P\_l + t_1)$

2. Relating Planes and Points:

Both camera and laser observe planes. We can represent points on these planes using plane normals (θ_c, θ_l) and distances (α_c, α_l):

$P\_c = \alpha\_c * \theta\_c$
$P\_l = \alpha\_l * \theta\_l$

3. Substituting and Minimizing:

Substitute these point representations into the reprojection error equation:

$e = \alpha\_c * \theta\_c - (R_1 (\alpha\_l * \theta\_l) + t_1)$

We want to minimize the squared norm (sum of squares) of the error (e) across multiple corresponding planes (i):

$\Sigma \| e\_i \|^2 = \Sigma \| \alpha\_c * \theta\_c - (R_1 (\alpha\_l * \theta\_l) + t_1) \|^2$ (minimize this)

4. Solving for Rotation (R_1):

Minimizing the error function with respect to R_1 is mathematically complex. The article uses SVD (Singular Value Decomposition) of the matrix product ($\theta\_l\theta\_c^T$) to find a solution for R_1.

Here's an intuitive explanation (without full derivation):

SVD decomposes ($\theta\_l\theta\_c^T$) into $U\Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix.
Minimization often leads to a solution where one of the singular values in Σ becomes very small (approaches zero).

By neglecting this small singular value and its corresponding columns in U and V, we can derive an approximate solution for R_l:
$R_l \approx V * U^T$

5. Solving for Translation (t_l):

Once we have R_l, we can solve for the translation (t_l) by rewriting the original error equation:

$\alpha\_c * \theta\_c - (R_l (\alpha\_l * \theta\_l) + t_l) = 0$

Since R_l is now known, we can solve this linear system of equations for t_l using techniques like least squares:

$t_l \approx (\theta\_c^T\theta\_c)^{(-1)} \theta\_c^T (\alpha\_c * I - R_l (\alpha\_l * \theta\_l))$

iii)

```
def lidar_translation_matrix(theta_c , alpha_c , alpha_l) :
    theta_c = np.array(theta_c)
    alpha_c = np.array(alpha_c)
    alpha_l = np.array(alpha_l)


    theta = np.linalg.inv(theta_c.T @ theta_c)

    # print(theta.shape)

    # t1 = np.dot(np.dot(theta , theta_c.T) , (alpha_c-alpha_l))

    t1 = theta @ theta_c.T @ (alpha_c - alpha_l)

    return t1
```

[3605] ✓ 0.0s

```
t1 = lidar_translation_matrix(theta_c,alpha_c,alpha_l)
```

[3606] ✓ 0.0s

```
t1.shape
```

[3607] ✓ 0.0s

```
(3, 1)
```

```python
def rotation_matrix(theta_l,theta_c) :
    theta_l = np.array(theta_l)
    theta_c = np.array(theta_c)

    # print(theta_l.shape)
    # print(theta_c.shape)

    theta = theta_l.T @ theta_c

    # theta = theta_l @ theta_c.T

    # print(theta.shape)

    U , _, V_t = np.linalg.svd(theta)

    # print(V_t.shape)
    # print(U.shape)

    # R1 = np.dot(V_t.T , U.T)

    R1 = V_t.T @ U.T

    return U, V_t, R1
```

✓ 0.0s

```python
U, V_t , R1 = rotation_matrix(theta_l , theta_c)
```

✓ 0.0s

```python
R1.shape
```

✓ 0.0s

```python
print(np.linalg.det(R1))
```

✓ 0.0s

```
0.9999999999999999
```

```python
def estimate_transformation(U,V_t,R1,t1) :
    # if np.linalg.det(R1) < 0 :
    #     V_t[2,:] *= -1
    #     R1 = np.dot(V_t.T, U.T)

    print(f"det(R1) = {np.linalg.det(R1)}")

    # print(R1.shape)
    # print(t1.shape)

    Ct_l = np.hstack((R1,t1))

    Ct_l = np.vstack((Ct_l, [0,0,0,1]))

    return Ct_l
```

✓ 0.0s

```python
Ct_l = estimate_transformation(U,V_t,R1,t1)
print(Ct_l.shape)
print(Ct_l)
```

✓ 0.0s

```
det(R1) = 0.999999999999999
(4, 4)
[[-1.75158876e-01 -9.84540172e-01  1.36164316e-04  8.85565624e-02]
 [ 1.53559935e-02 -2.87025999e-03 -9.99877970e-01 -3.62081439e-01]
 [ 9.84420419e-01 -1.75135410e-01  1.56213440e-02 -5.99256711e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

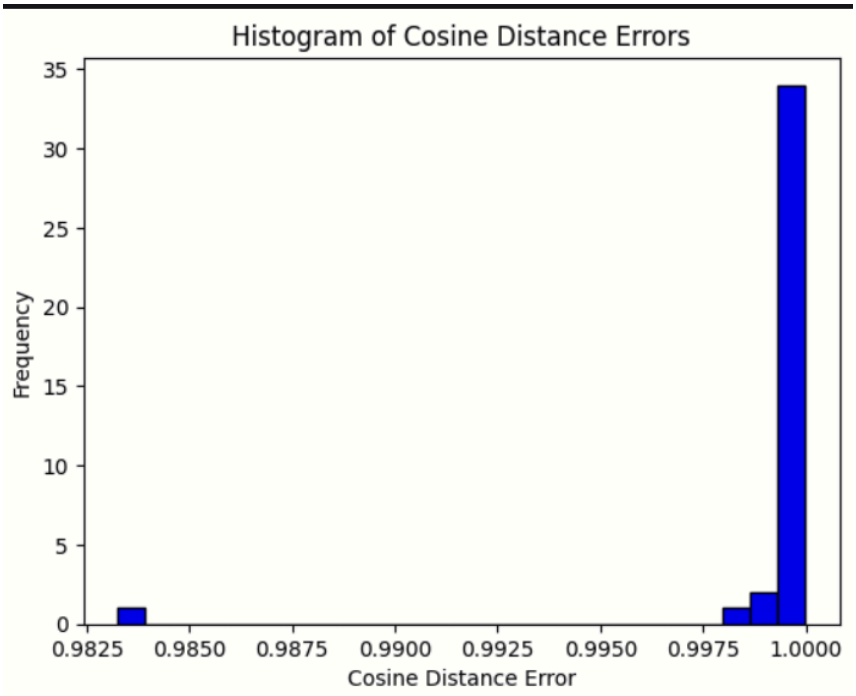
iv)

Majority of the points are within the checkerboard patterns boundary

Images with projected points are saved in a separate directory in the submitted zip folder (CV-A2-calibration\point_projected_images_final)

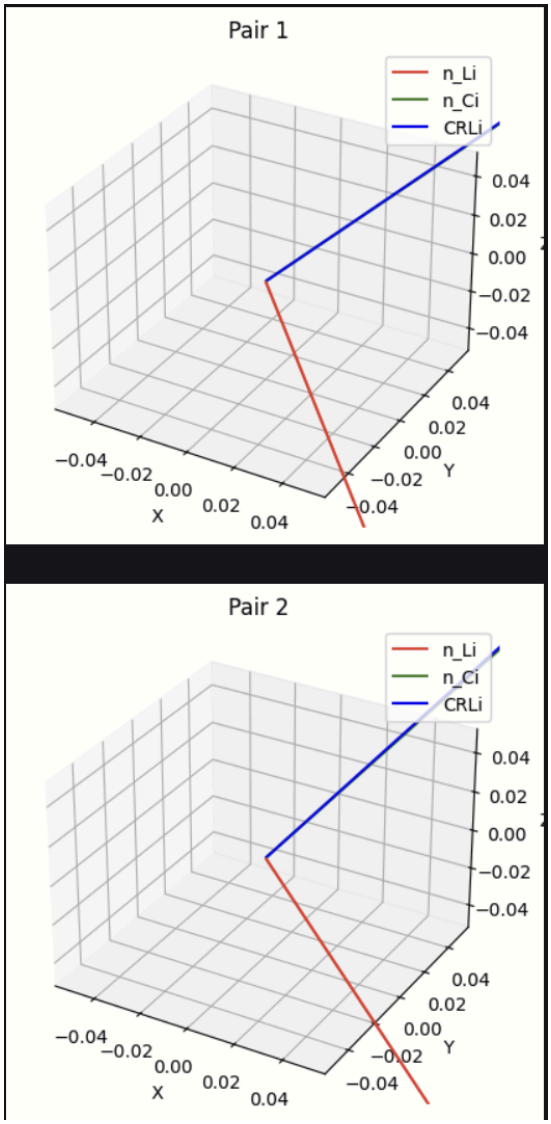
Eg:

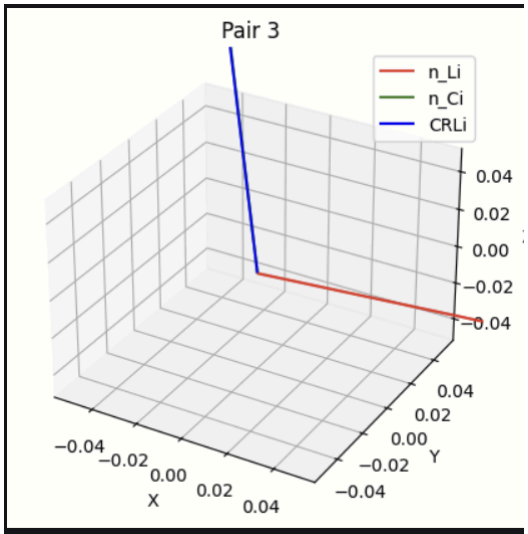
v)

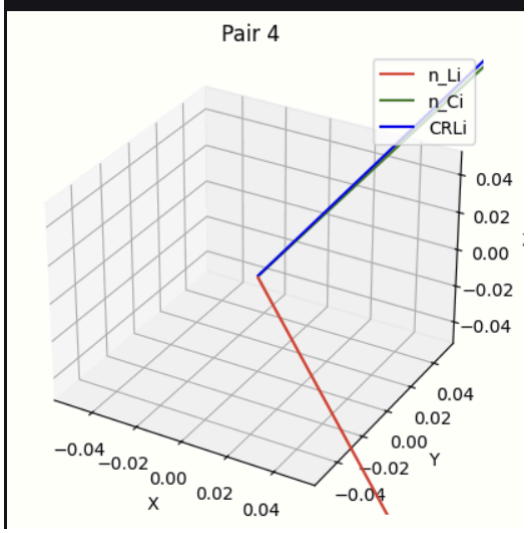


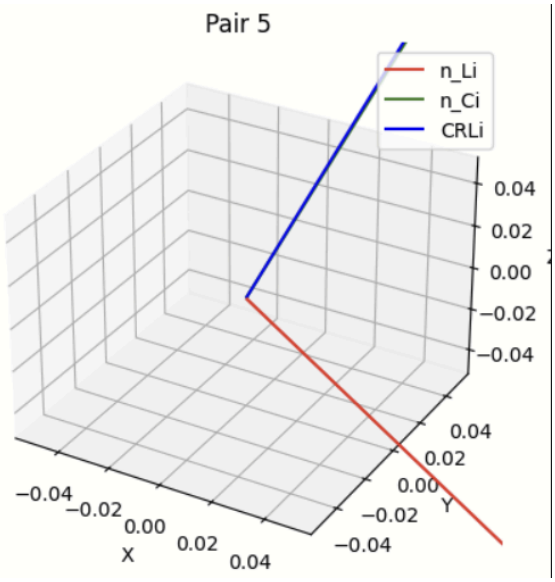

```
Average error: 0.9993530978365598
Standard deviation: 0.002661000072088747
```

Normal vectors for any 5 images :


Pair 1
n_Li
n_Ci
CRLi
0.04
0.02
0.00
−0.02
−0.04
X
Y
Pair 2
n_Li
n_Ci
CRLi
0.04
0.02
0.00
−0.02
−0.04
X
Y







References :
1) https://www.youtube.com/watch?v=C2wSyDIq9X8
2) https://www.google.com/url?q=https://www.ri.cmu.edu/pub_files/pub4/unnikrishnan_ranjith_2005_3/unnikrishnan_ranjith_2005_3.pdf&sa=D&source=apps-viewer-frontend&ust=1711389105894331&usg=AOvVaw3RzcbaYDohlZasvsQalQho&hl=en

3) https://planetmath.org/proofofrodriguesrotationformula