Computer Vision

Assignment 2

Sahil Goyal

2020326

1.

Explanation:

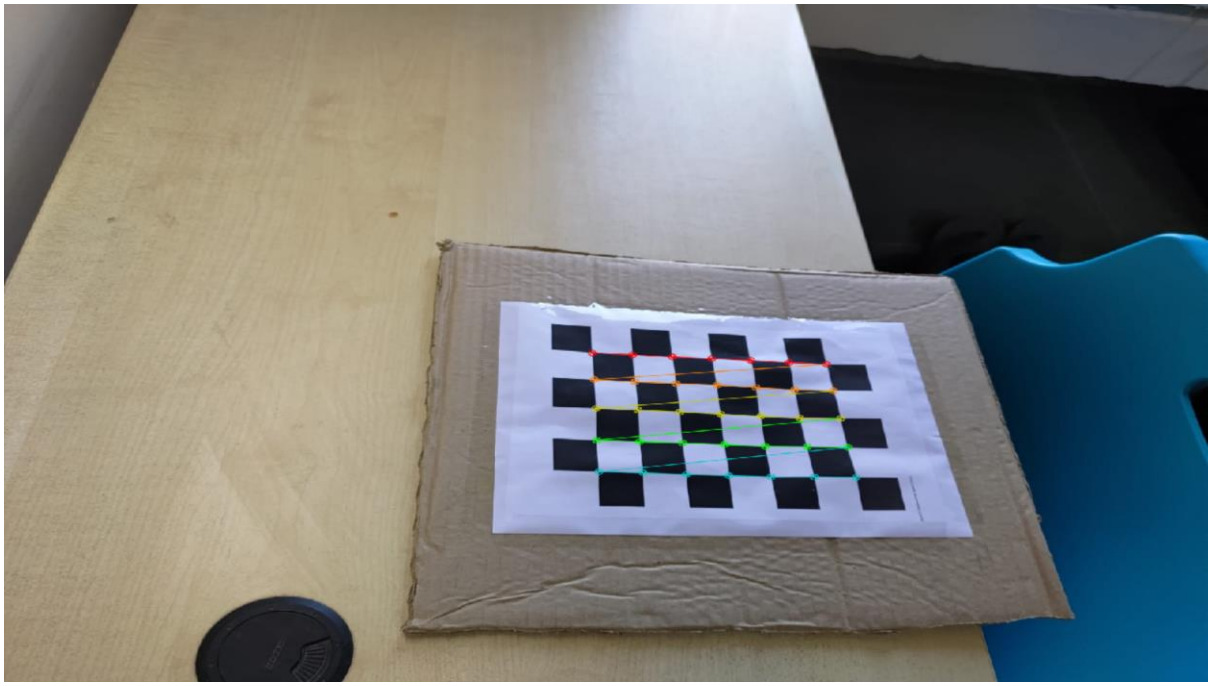
- I have taken 25 images of the chessboard in different orientations and have covered all of the degrees of freedom.
- The entire dataset is in the folder cv_data_1

1.

Explanation of Code:

- First, we use cv2.findChessBoardCorners() in order to find the corners in the chessboard, and then we refine them using cv2.cornerSubPix(). We have also drawn the images using cv2.drawChessBoardCorners() and cv2.imshow() in order to confirm that the corners were detected properly.
- After that, we use the cv2.CalibrateCamera() method in order to get the required parameters.

Outputs:



```
Intrinsic Matrix:
[[1.43707979e+03 0.00000000e+00 7.84957604e+02]
 [0.00000000e+00 1.45302350e+03 4.82753728e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
Focal Length:  1437.079792351073 1453.0234989407277
Skew:  0.0
Principal Point:  784.9576038512337 482.7537281703449
```

2.

Explanation of Code:

- We also found the rotation and translation vectors for all the images.

Outputs:

```
For Image  1 :
Rotation Matrix:  [[-0.6544898 ]
 [-0.45594445]
 [-0.25846126]]
Translation Matrix:  [[-2.55879693]
 [-2.03936763]
 [17.99305745]]
For Image  2 :
Rotation Matrix:  [[-0.49592048]
 [-0.01116468]
 [-0.02617083]]
Translation Matrix:  [[-2.91882279]
 [-1.06891417]
 [12.46696105]]
For Image  3 :
Rotation Matrix:  [[-0.25726757]
 [ 0.5687323 ]
 [ 1.33827353]]
Translation Matrix:  [[-2.11630885]
 [-2.16275911]
 [17.86607245]]
```

```
For Image  4 :
Rotation Matrix:  [[-0.35955576]
 [ 0.07008381]
 [ 0.01690072]]
Translation Matrix:  [[-4.21362216]
 [-1.95239428]
 [12.79837864]]
For Image  5 :
Rotation Matrix:  [[-0.62296185]
 [-0.2161796 ]
 [ 0.02579599]]
Translation Matrix:  [[-0.08193684]
 [ 2.35966339]
 [25.68973993]]
For Image  6 :
Rotation Matrix:  [[-0.08454305]
 [ 0.08046353]
 [ 1.55295205]]
Translation Matrix:  [[ 0.37671496]
 [-0.65467644]
 [12.30360872]]
For Image  7 :
Rotation Matrix:  [[-0.65541316]
 [ 0.35137508]
 [ 1.37136539]]
Translation Matrix:  [[-0.73203185]
 [ 0.34149916]
 [18.40011921]]
```

```
For Image  8 :
Rotation Matrix:  [[-0.08149423]
 [ 0.11576941]
 [-0.01859277]]
Translation Matrix:  [[-4.6281866 ]
 [-1.11141299]
 [15.59396579]]
For Image  9 :
Rotation Matrix:  [[-0.46287508]
 [-0.33380617]
 [ 0.04649107]]
Translation Matrix:  [[-2.34473846]
 [-1.55033596]
 [18.27626482]]
For Image  10 :
Rotation Matrix:  [[-0.03211804]
 [-0.29585823]
 [ 0.00227648]]
Translation Matrix:  [[-2.02408598]
 [ 0.27382024]
 [26.80019942]]
For Image  11 :
Rotation Matrix:  [[-0.43761518]
 [ 0.69527299]
 [ 1.589708  ]]
Translation Matrix:  [[-0.68307157]
 [-0.64728144]
 [21.8960317 ]]
```

```
For Image  12 :
Rotation Matrix:  [[-0.6321968 ]
 [ 0.12026075]
 [ 0.17947041]]
Translation Matrix:  [[-2.14607656]
 [-2.18740101]
 [13.97402184]]
For Image  13 :
Rotation Matrix:  [[-0.45656729]
 [ 0.39181978]
 [-0.04451564]]
Translation Matrix:  [[-3.20303756]
 [-0.77773464]
 [20.68450614]]
For Image  14 :
Rotation Matrix:  [[-0.77141174]
 [ 0.07737759]
 [-0.09030625]]
Translation Matrix:  [[-3.03357033]
 [-0.29679813]
 [21.80349483]]
For Image  15 :
Rotation Matrix:  [[-0.32372014]
 [-0.06412011]
 [ 0.01678987]]
Translation Matrix:  [[-2.3566139 ]
 [-1.3371789 ]
 [15.60595787]]
```

```
For Image  16 :
Rotation Matrix:  [[-0.57018207]
 [ 0.40259705]
 [ 0.03819668]]
Translation Matrix:  [[-3.84310532]
 [-0.20203017]
 [20.97578464]]
For Image  17 :
Rotation Matrix:  [[-0.60399006]
 [ 0.37675133]
 [-0.26921472]]
Translation Matrix:  [[-5.20476169]
 [ 0.09639681]
 [18.0491597 ]]
For Image  18 :
Rotation Matrix:  [[-0.65750645]
 [-0.23625501]
 [-0.20688458]]
Translation Matrix:  [[-4.55215489]
 [-0.14254139]
 [29.5619598 ]]
For Image  19 :
Rotation Matrix:  [[-0.08782421]
 [-0.09698588]
 [-1.5686902 ]]
Translation Matrix:  [[-4.0651211 ]
 [ 6.39263525]
 [14.95993408]]
```

```
For Image  20 :
Rotation Matrix:  [[-0.85152735]
 [ 0.08586742]
 [-0.04576333]]
Translation Matrix:  [[-2.12442267]
 [ 0.16769556]
 [21.20100959]]
For Image  21 :
Rotation Matrix:  [[-0.67495499]
 [-0.25242454]
 [-0.55188163]]
Translation Matrix:  [[-2.9258171 ]
 [-0.39010828]
 [17.13253556]]
For Image  22 :
Rotation Matrix:  [[-0.68335497]
 [ 0.20019995]
 [ 0.11557826]]
Translation Matrix:  [[-5.1928859 ]
 [-2.55033576]
 [19.67570236]]
For Image  23 :
Rotation Matrix:  [[-0.69920284]
 [ 0.03604341]
 [ 0.13046402]]
Translation Matrix:  [[-1.38845712]
 [-1.63053973]
 [19.64148516]]
```

```
For Image  24 :
Rotation Matrix:  [[-0.48798333]
 [-0.69225667]
 [-1.4700423 ]]
Translation Matrix:  [[-3.81153663]
 [ 3.99337951]
 [12.1457306 ]]
For Image  25 :
Rotation Matrix:  [[-0.48236168]
 [ 0.2411545 ]
 [-0.02658123]]
Translation Matrix:  [[-2.54200956]
 [-0.53852594]
 [16.98553635]]
```

3.

Explanation of Code:

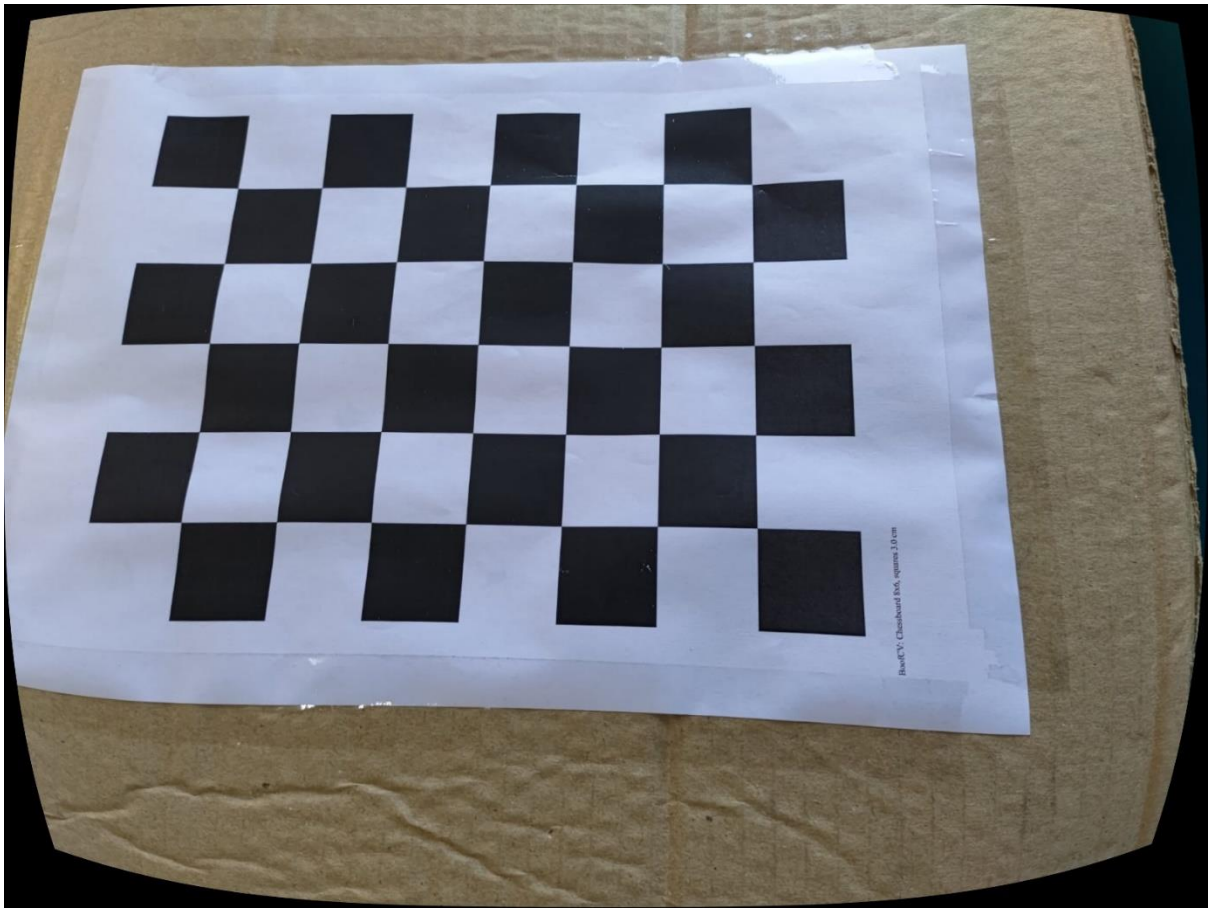
- We have used the radial distortion coefficients in order to undistort 5 images.

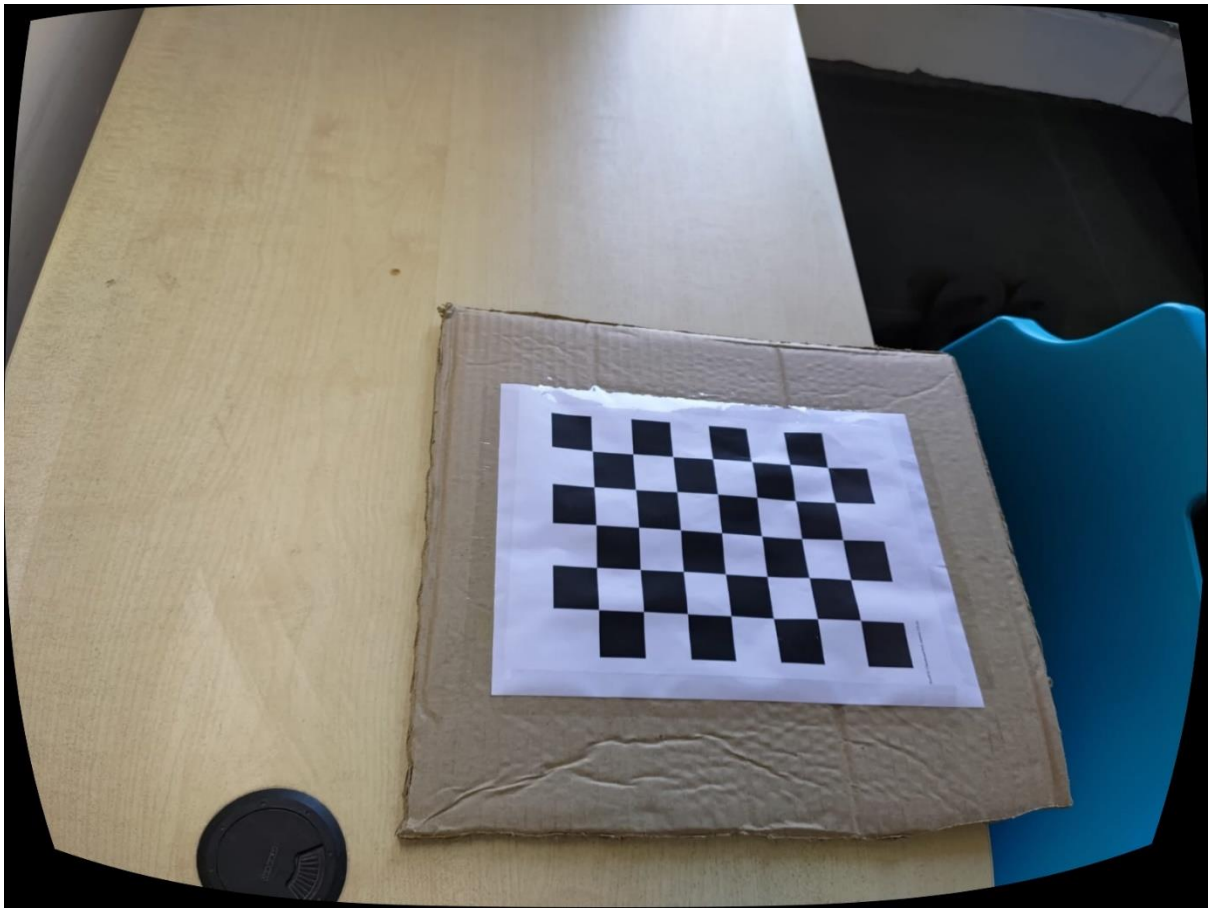
Observations:

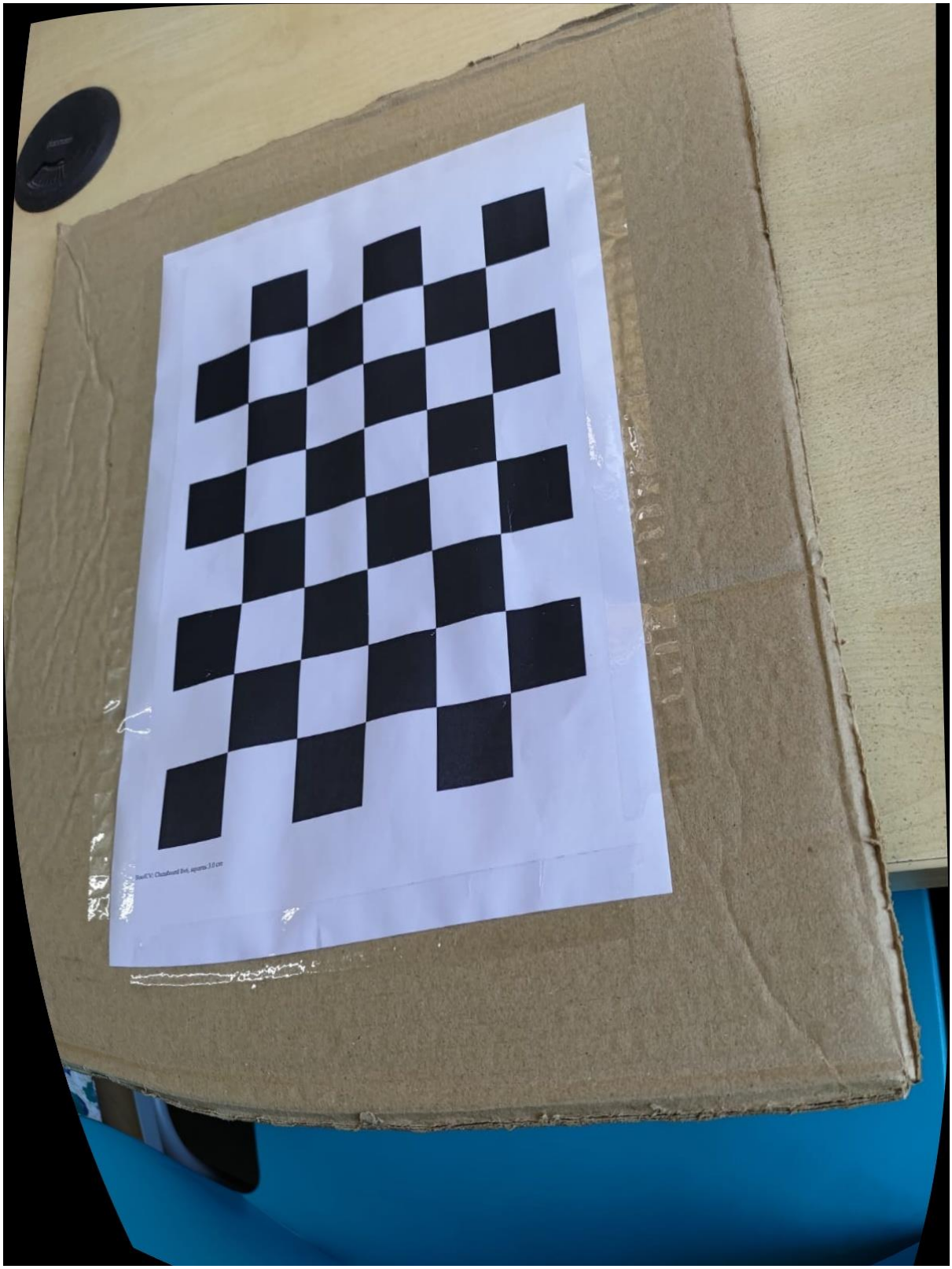
- We can see that the lines at the corners of the images appear straight.

Outputs:

```
Radial Distortion Coefficients:
[[ 1.77258020e-01 -1.89642360e+00 -2.72124052e-02 -1.25037647e-03
   4.46750683e+00]]
```







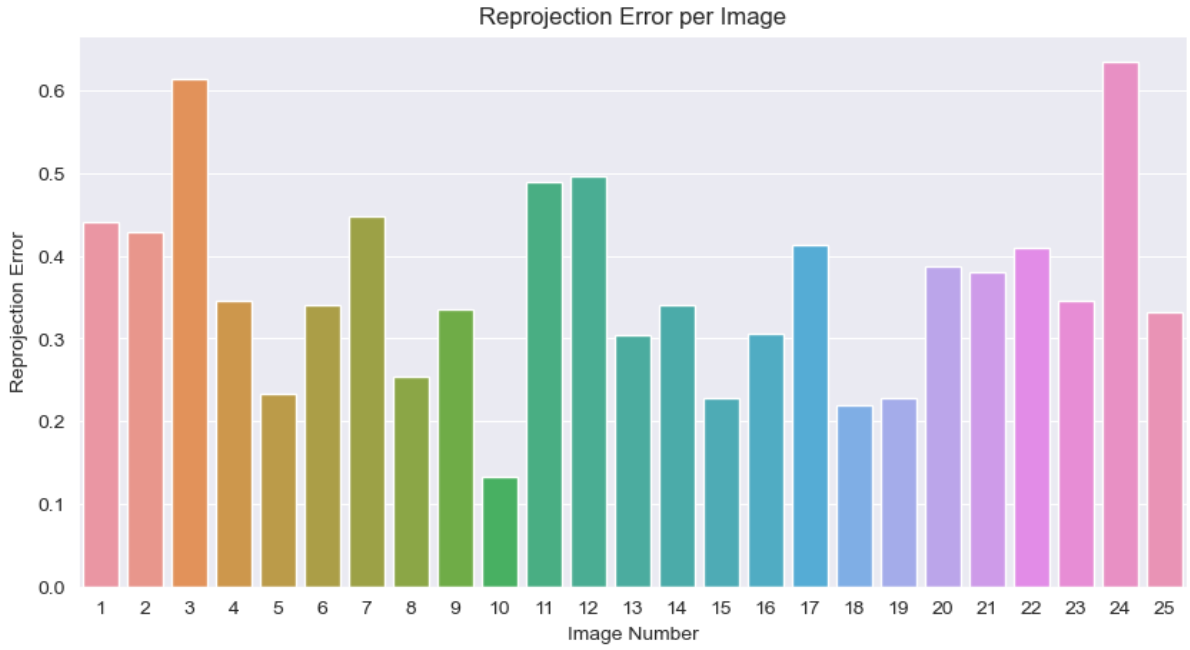
4.

Explanation of Code:

- We use the cv2.projectPoints() function in order to find the projection of the points, and the find the error by finding the difference between the points we had and the ones we got from reprojection.
- We added these errors to a list, and then we reported the mean and standard deviation, and also plotted the bar graph.

Outputs:

```
Reprojection Error:
[0.44097008 0.42836692 0.61328891 0.34474652 0.23344168 0.34093395
 0.44853345 0.25415031 0.33544511 0.13359087 0.4892172  0.49596546
 0.30410613 0.34043321 0.22725741 0.30636541 0.41263508 0.21989016
 0.2271324  0.3869978  0.3807433  0.4090344  0.34609802 0.63444835
 0.33184166]
Average Reprojection Error:  0.3634253512805252
Standard Deviation of Reprojection Error:  0.11691471097732259
```

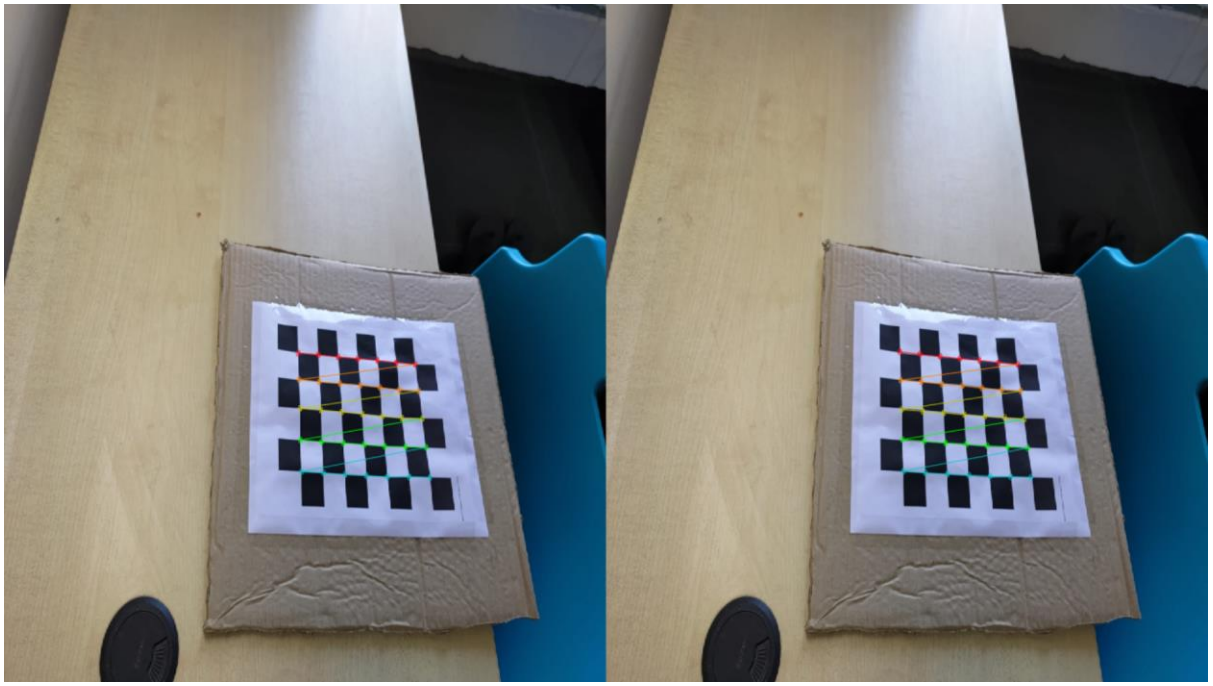


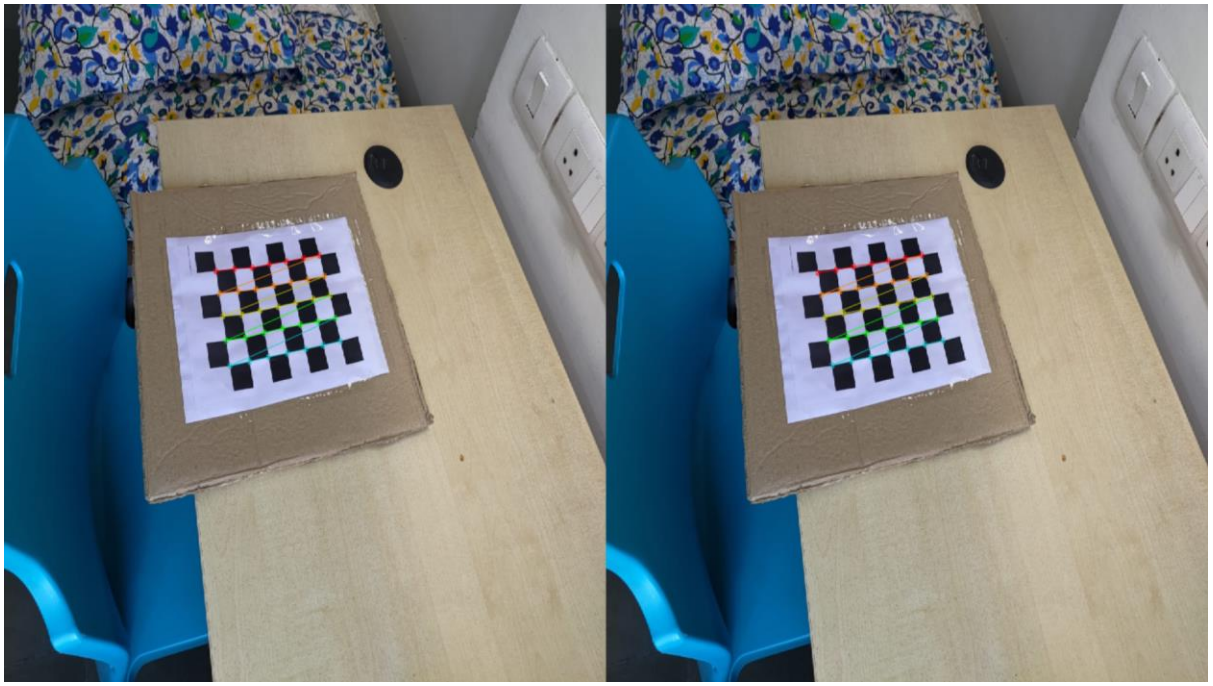
5.

Explanation of Code:

- We read the image twice and then, in one, we use cv2.drawChessBoardCorners() to draw the original chessboard corners we got, and the other, we draw the projected points.
- We horizontally concatenate the images, and show them side by side.
- The reprojection error refers to the measurement of the distance between the image point and its corresponding reprojected point. To find the reprojection error, the camera matrix and distortion coefficients are estimated, and a set of observed image points are selected from the distorted image. These observed points are then reprojected onto the undistorted image plane. By calculating the distance between each observed point and its corresponding reprojected point, and then averaging these distances over all observed points, the reprojection error is obtained.

Outputs:





6.

Explanation of Code:

- We iterate through all of the images, and get the normals in the world coordinate frame, which we later convert to the camera coordinate frame.

Output:

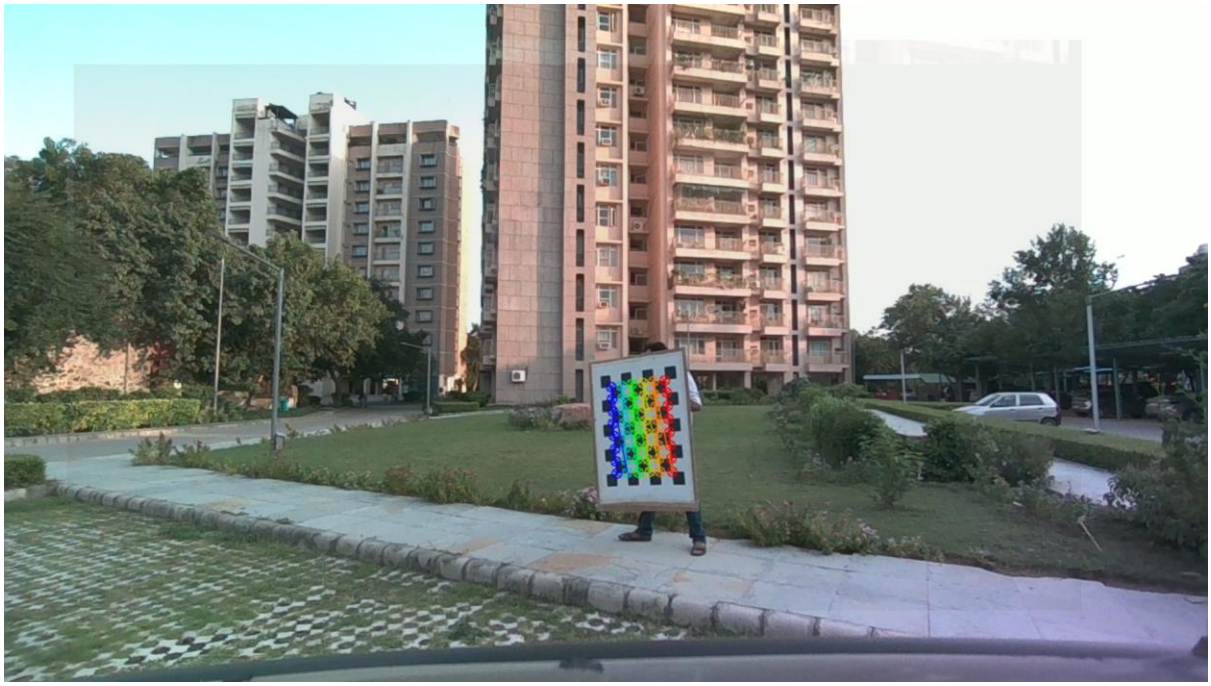
```
Normal Vectors:
[-0.31027868  0.64013889  0.70281529]
[0.01522908 0.4834549  0.87523679]
[0.24079865 0.45063435 0.85961892]
[0.05188252 0.32709875 0.94356484]
[-0.17873227  0.58481553  0.79123042]
[-0.01879286 -0.005597    0.99980773]
[-0.14307009  0.62306425  0.76897457]
[0.10048358 0.01798964 0.99477607]
[-0.32158324  0.43020953  0.84350695]
[-0.27370248  0.01603966  0.96168065]
[0.12371823 0.6460629  0.7531909 ]
[0.06977162 0.6010547  0.7961565 ]
[0.3739895  0.41944133 0.82716433]
[0.10689002 0.68555814 0.72012816]
[-0.0553238   0.30993344  0.94914727]
[0.34536788 0.52411743 0.77847412]
[0.39340062 0.48510984 0.78096376]
[-0.15435816  0.6288071   0.76208608]
[-0.01187069 -0.01903937  0.99974826]
[-0.13230335 -0.73390227  0.66624566]
[-0.04033324  0.64940787  0.7593699 ]
[0.12761054 0.6237703  0.77112007]
[-4.92680550e-04  6.44155772e-01  7.64894175e-01]
[-0.15532096  0.69466333  0.70236618]
[0.2362828  0.46339671 0.85406904]
```

2.

Explanation of Code:

- First, we iterate through the images. We find the corners using cv2.findChessBoardCorners(). We refine them using cv2.cornerSubPix and then draw them on the image with cv2.drawChessBoardCorners(), and finally show the output with cv2.imshow().
- We store the corners in a numpy array chessboard_camera_points

Output:



1.

Explanation of Code:

- First, we iterate through the lidar scans. We load them through open3d and then find the normals and the offsets.

Output:

```
offsets
[0.84488157]
[0.31996063]
[-0.97422567]
[-1.16536684]
[-1.67681087]
[1.2729507]
[-3.79204272]
[-2.49011658]
[-1.26051152]
[-0.99364483]
[1.11275098]
[-1.11790516]
[3.76579727]
[3.03901192]
[-1.17382667]
[0.65419827]
[-0.98668746]
[-2.82216585]
[0.16709027]
[-3.20174756]
[1.16240308]
[-0.94781817]
[0.82494079]
[-1.56653819]
[-0.31182213]
[-0.87961006]
[-0.6962598]
[-3.40119465]
[1.06890249]
```

```
[1.0132697]
[-1.16476713]
[-0.64681465]
[1.2214109]
[-0.79322753]
[-1.22242456]
[-1.87866918]
Normals
[0.46434414 0.1951279  0.86389214]
[-0.19416673 -0.43777446 -0.87786833]
[-0.22898373  0.21180999 -0.95010682]
[-0.02600076  0.3396704   0.94018508]
[-0.17343056 -0.17108949  0.96987124]
[ 0.13418357 -0.38012254 -0.91515115]
[-0.60212489 -0.21290389 -0.76949175]
[-0.34394565  0.42752023  0.83601905]
[-0.09198474  0.06912708  0.99335807]
[-0.09757995 -0.00560913  0.99521188]
[0.43241828 0.07160028 0.89882581]
[-0.53289683 -0.2487181  -0.80880175]
[ 0.46712677 -0.58209889  0.66554749]
[ 0.42576259 -0.33973658  0.83863298]
[0.37461525 0.70607328 0.60093588]
[-0.00834859 -0.18250687 -0.98316913]
[-0.08942928  0.39943664 -0.9123885 ]
[-0.17408867  0.43831895  0.88179909]
[-0.0458275   0.61665218  0.78590072]
[-0.43957528  0.45883727  0.77216704]
[0.3822704  0.66752668 0.63896593]
[-0.16711095  0.12043922 -0.9785542 ]
```

```
[0.47778673 0.37055385 0.79649839]
[-0.551609   -0.44128102  0.70781253]
[-0.11792459  0.07680946 -0.99004752]
[-0.01725182  0.01081867  0.99979264]
[-0.25928043  0.19471126 -0.94597103]
[-0.55509853  0.11984621 -0.8231054 ]
[ 0.04242457  0.02792697 -0.99870929]
[ 0.01168615 -0.03421401 -0.9993462 ]
[-0.14090621 -0.31390614  0.93894003]
[0.08346221 0.3149461  0.94543271]
[ 0.13068086  0.40145978 -0.90650568]
[0. 0. 1.]
[-0.19715885  0.11434221 -0.97368077]
[-0.32284531  0.15627658 -0.93346052]
```

3.

Explanation of Code:

Q2

2. First, we will find the camera normals and offsets and the lidar normals and offsets We obtain lidar normal and offsets through SVD

$$\Theta_c = [\theta_{c1}\ \theta_{c2} \cdots \theta_{cn}]^T_{(n\times 3)}$$

$$\Theta_l = [\theta_{l1}\ \theta_{l2} \cdots \theta_{ln}]^T_{(n\times 3)}$$

are the camera and lidar normals.

Here

θ_{ci} and θ_{li} are the camera and lidar normals for the ith image

$$d_c = [d_{c1}\ d_{c2} \cdots d_{cn}]^T_{(n\times 1)}$$

$$d_l = [d_{l1}\ d_{l2} \cdots d_{ln}]^T_{(n\times 1)}$$

are the camera and lidar offsets

Here d_{ci} and d_{li} are the camera and lidar offsets for ith image.

The translation vector has a value of

$$t_c = (\theta_c^T \theta_c)^{-1} \theta_c^T (x_c - x_v)$$

[redacted]

We will take SVD of

$$\theta_c \theta_c^T = U S V^T$$

and we will use this to find R

$$R_c = V U^T$$

We use the rotation matrix and translation vectors to get the required transformation Matrix

$$T_c = [R_c | t_c]$$

- We want to find the transformation Matrix T_L^C
- We have only 48 corners per chessboard images (as the number of interior corners are (8 X 6)). So, in order to match the number of points, we take 48 random points from the lidar scan points.
- We use cv2.solvePnPRansac() in order to find the rotation and translation vectors and then in order to find the rotation matrix we use cv2.Rodrigues() to convert the rotation vector into the 3X3 rotation matrix.
- We use these to get the final transformation matrix.

Output:

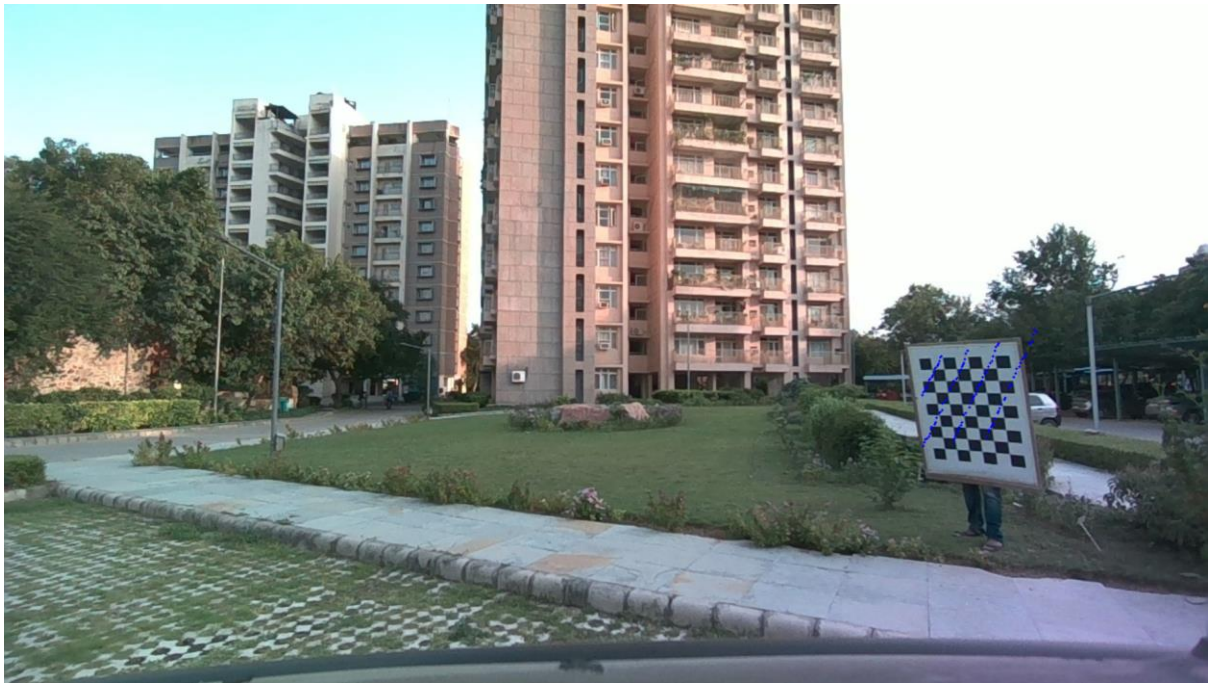
```
Transformation Matrix
[[-0.09816138  0.27031424  0.01741534  1.63359077]
 [-0.18908696 -0.10550818  0.00487244  1.04709033]
 [ 0.10807137  0.00619036 -0.03996133  3.16389325]]
```

4.

Explanation of Code:

- We iterate through the images, and use the transformation matrix to project the lidar points to the chessboard.

Output:



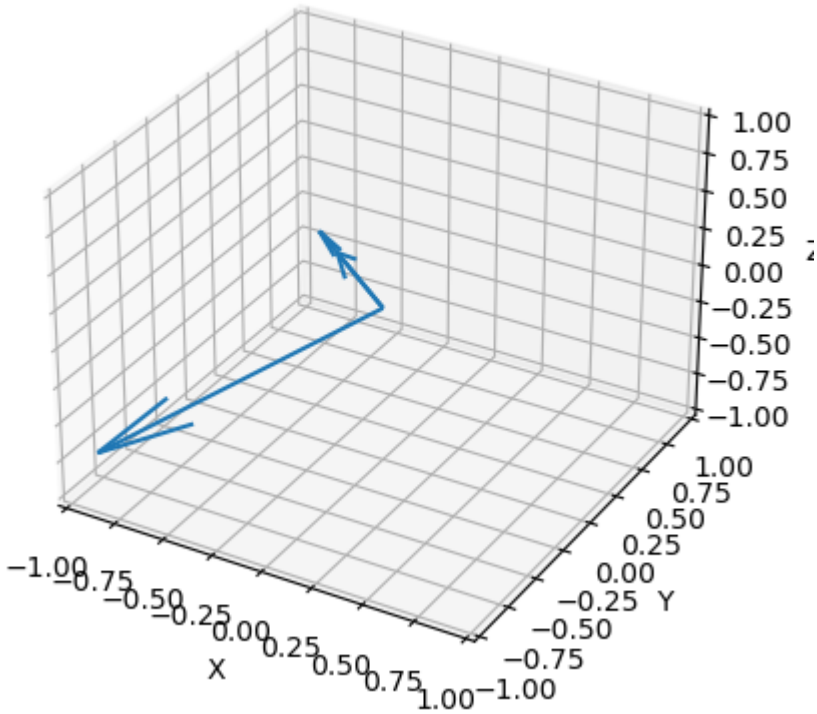
- As we can see, projection is working well, but there are still some points that are outside the boundary of the chessboard.

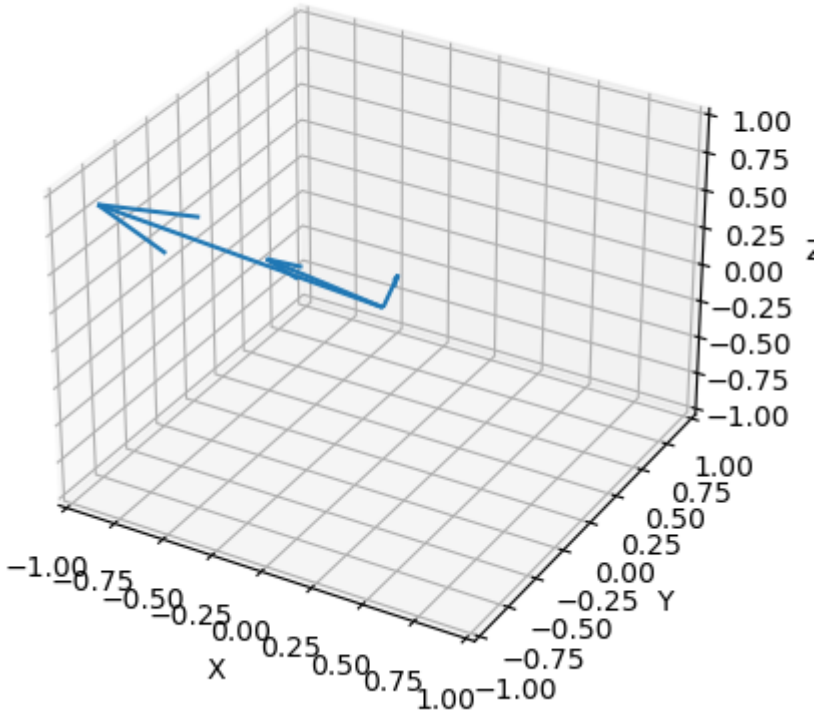
5.

Explanation of Code:

- We have to plot the normal vectors for any 5 images and LIDAR scan pairs. We use the quiver function to plot the vectors, and set the limits to (-1,1) for all three axis.
- We create two arrays for storing the cosine errors and find all the cosine errors. At the end, we print out the mean and standard deviation of the cosine errors and plot the necessary histograms.

Output and graph:


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Z
1.00
0.75
0.50
0.25
0.00
−0.25
Y
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X

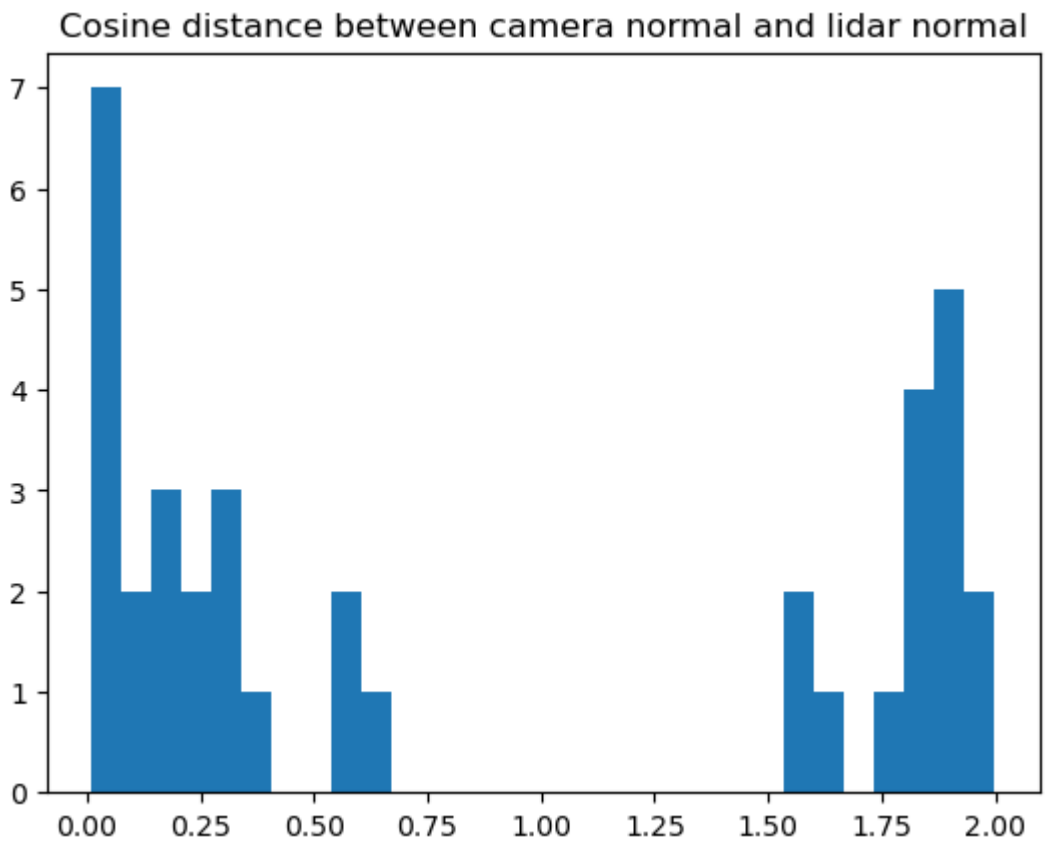

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Z
1.00
0.75
0.50
0.25
0.00
−0.25
Y
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X

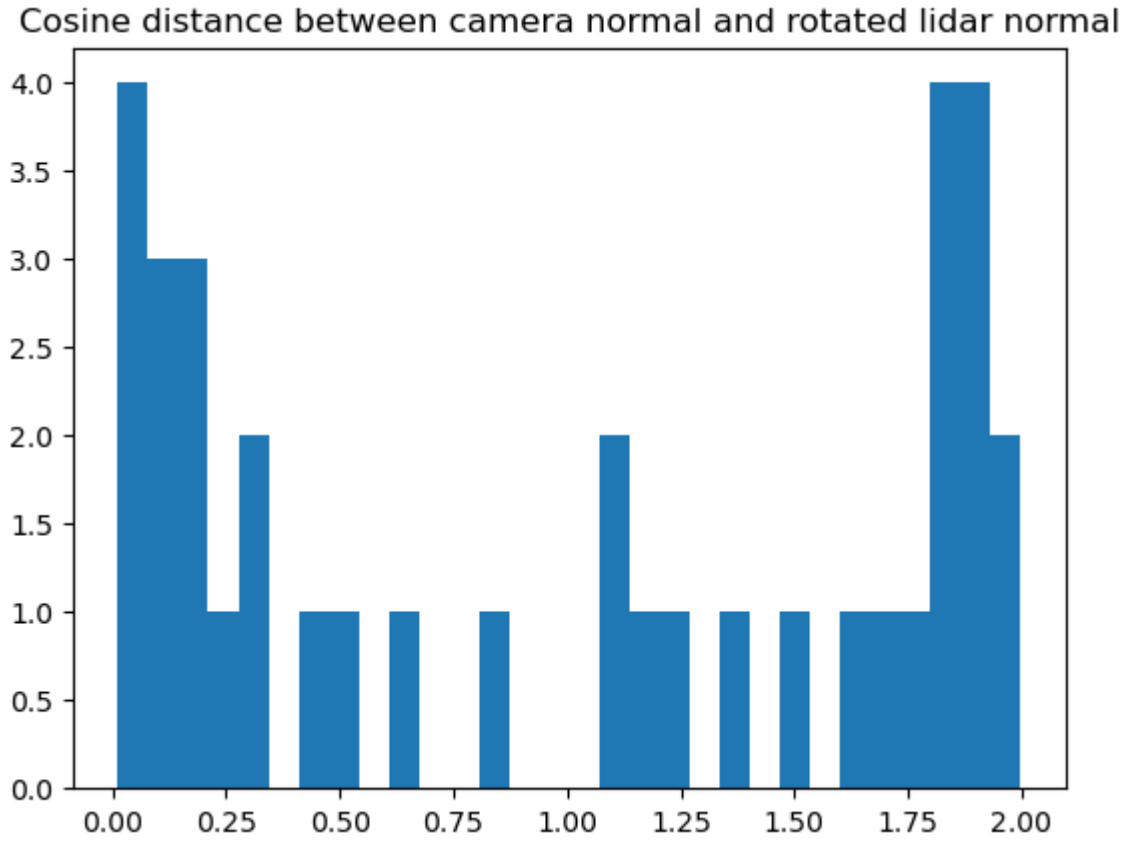
```
Cosine distance between camera normal and lidar normal:  [1.86268308]
Cosine distance between camera normal and rotated lidar normal:  [1.37602167]
Cosine distance between camera normal and lidar normal:  [1.86379527]
Cosine distance between camera normal and rotated lidar normal:  [1.83803427]
Cosine distance between camera normal and lidar normal:  [0.00945222]
Cosine distance between camera normal and rotated lidar normal:  [0.17130264]
Cosine distance between camera normal and lidar normal:  [0.19076924]
Cosine distance between camera normal and rotated lidar normal:  [1.90312957]
Cosine distance between camera normal and lidar normal:  [0.22372366]
Cosine distance between camera normal and rotated lidar normal:  [0.44623199]
Cosine distance between camera normal and lidar normal:  [0.28641907]
Cosine distance between camera normal and rotated lidar normal:  [1.98214947]
Cosine distance between camera normal and lidar normal:  [0.31564151]
Cosine distance between camera normal and rotated lidar normal:  [0.01178498]
Cosine distance between camera normal and lidar normal:  [0.58057589]
Cosine distance between camera normal and rotated lidar normal:  [0.10653802]
Cosine distance between camera normal and lidar normal:  [0.05849056]
Cosine distance between camera normal and rotated lidar normal:  [0.18163065]
Cosine distance between camera normal and lidar normal:  [0.02647316]
Cosine distance between camera normal and rotated lidar normal:  [0.20739079]
Cosine distance between camera normal and lidar normal:  [1.89437582]
Cosine distance between camera normal and rotated lidar normal:  [1.88173884]
Cosine distance between camera normal and lidar normal:  [0.37144678]
Cosine distance between camera normal and rotated lidar normal:  [1.19519794]
Cosine distance between camera normal and lidar normal:  [1.5655639]
Cosine distance between camera normal and rotated lidar normal:  [0.65203432]
Cosine distance between camera normal and lidar normal:  [0.0474116]
Cosine distance between camera normal and rotated lidar normal:  [1.85286065]
Cosine distance between camera normal and lidar normal:  [0.1874289]
Cosine distance between camera normal and rotated lidar normal:  [1.23655056]
```

```
Cosine distance between camera normal and lidar normal:  [1.92222835]
Cosine distance between camera normal and rotated lidar normal:  [0.13167986]
Cosine distance between camera normal and lidar normal:  [0.0704226]
Cosine distance between camera normal and rotated lidar normal:  [0.3168895]
Cosine distance between camera normal and lidar normal:  [0.11247265]
Cosine distance between camera normal and rotated lidar normal:  [0.02315575]
Cosine distance between camera normal and lidar normal:  [1.85945189]
Cosine distance between camera normal and rotated lidar normal:  [1.86781458]
Cosine distance between camera normal and lidar normal:  [1.87325066]
Cosine distance between camera normal and rotated lidar normal:  [1.5070019]
Cosine distance between camera normal and lidar normal:  [1.73526169]
Cosine distance between camera normal and rotated lidar normal:  [1.62227174]
Cosine distance between camera normal and lidar normal:  [0.22263341]
Cosine distance between camera normal and rotated lidar normal:  [1.6903639]
Cosine distance between camera normal and lidar normal:  [1.92879018]
Cosine distance between camera normal and rotated lidar normal:  [1.08884178]
Cosine distance between camera normal and lidar normal:  [0.05638266]
Cosine distance between camera normal and rotated lidar normal:  [0.83364261]
Cosine distance between camera normal and lidar normal:  [1.96894807]
Cosine distance between camera normal and rotated lidar normal:  [0.0118715]
Cosine distance between camera normal and lidar normal:  [1.65330378]
Cosine distance between camera normal and rotated lidar normal:  [0.53108421]
Cosine distance between camera normal and lidar normal:  [1.82584047]
Cosine distance between camera normal and rotated lidar normal:  [1.8009093]
Cosine distance between camera normal and lidar normal:  [0.59341014]
Cosine distance between camera normal and rotated lidar normal:  [1.11504274]
Cosine distance between camera normal and lidar normal:  [0.29328102]
Cosine distance between camera normal and rotated lidar normal:  [1.74389904]
Cosine distance between camera normal and lidar normal:  [1.89019098]
Cosine distance between camera normal and rotated lidar normal:  [0.27821797]
```

```
Cosine distance between camera normal and lidar normal:  [0.13089659]
Cosine distance between camera normal and rotated lidar normal:  [0.12933689]
Cosine distance between camera normal and lidar normal:  [0.02190473]
Cosine distance between camera normal and rotated lidar normal:  [0.03038182]
Cosine distance between camera normal and lidar normal:  [0.19382548]
Cosine distance between camera normal and rotated lidar normal:  [1.89860673]
Cosine distance between camera normal and lidar normal:  [1.99771732]
Cosine distance between camera normal and rotated lidar normal:  [1.99771732]
Cosine distance between camera normal and lidar normal:  [1.57993466]
Cosine distance between camera normal and rotated lidar normal:  [0.24031614]
Cosine distance between camera normal and lidar normal:  [0.64170379]
Cosine distance between camera normal and rotated lidar normal:  [1.84043428]
```

```
Mean cosine distance between camera normal and lidar normal:  0.890447271978858
Mean cosine distance between camera normal and rotated lidar normal:  0.9928354419677142
Standard deviation cosine distance between camera normal and lidar normal:  0.8095887445962261
Standard deviation cosine distance between camera normal and rotated lidar normal:  0.749121543025320
```


Cosine distance between camera normal and lidar normal
0
1
2
3
4
5
6
7
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Cosine distance between camera normal and rotated lidar normal
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00