**Computer Vision 2024**
**(CSE344/ CSE544/ ECE344/ ECE544)**
**Assignment-2**
Max Marks (UG/PG): 110 / 110 Due Date: 24/03/2024, 11:59 PM Instructions

**MOHAMMAD SHARIQ**
**2020220**

---

- Keep collaborations at high-level discussions. Copying/plagiarism will be dealt with strictly.
- Your submission should be a single zip file Roll Number HW[n].zip. Include only the relevant files arranged with proper names. A single .pdf report explaining your codes with relevant graphs, visualization and solution to theory questions.
- Remember to turn in after uploading on Google Classroom. No justifications would be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs during their office hours two days before the deadline.
- Kindly document your code. Don't forget to include all the necessary plots in your report.
- All [PG] questions, if any, are optional for UG students but are mandatory for PG students. UG students will get BONUS marks for solving that question.
- All [BONUS] questions, if any, are optional for all the students. As the name suggests, BONUS marks will be awarded to all the students who solve these questions.
- Your submission must include a single python (*.py*) file for each question. You can submit *.ipynb* along with the *.py* files. Failing to follow the naming convention or not submitting the python files will incur a penalty.

1. (15 points) Consider a vector $(2, 5, 1)^T$, which is rotated by $\pi/2$ about the Y-axis, followed by a rotation about X-axis by $-\pi/2$ and finally translated by $(-1, 3, 2)^T$.
    1. (4 points) What is the coordinate transformation matrix in the case?
    2. (3 points) Find the new coordinates of this vector. Where does the origin of the initial frame of reference get mapped to?
    3. (4 points) What is the direction of the axis of the combined rotation in the original frame of reference and what is the angle of rotation about this axis?
    4. (4 points) Using Rodrigues formula, show that you achieve the same rotation matrix as you sequentially apply the two rotations.
2. (5 points) Consider a rotation R of angle $\theta$ about the axis u (a unit vector). Show that $Rx = \cos\theta x + \sin\theta(u \times x) + (1 - \cos\theta)(u^\top x)u$
3. (10 points) [PG]

    The image formation process can be summarized in the equation $x = K[R|t]X$, where K is the intrinsic parameter matrix, [R|t] are the extrinsic parameters, X is the 3D point and x is the image point in the homogeneous coordinate system. Consider a scenario where there are two cameras ($C_1$&$C_2$) with intrinsic parameters $K_1$&$K_2$ and corresponding image points $x_1$&$x_2$ respectively. Assume that the first camera frame of reference is known and is used as the world coordinate frame. The second camera orientation is obtained by a pure 3D rotation R applied to the first camera's orientation. Show that the homogeneous coordinate representation of image points $x_1$ and $x_2$ of $C_1$ and $C_2$ respectively, are related by an equation $x_1 = Hx_2$, where H is an invertible 3×3 matrix. Find H in terms of $K_1$, $K_2$ & R.
4. (40 points) Camera Calibration:

    Refer to the following tutorials on camera calibration: Link1 and Link2. Place your camera (laptop or mobile phone) stationary on a table. Take the printout of a chessboard calibration pattern as shown in the links above and stick it on a hard, planar surface. Click ~ 25 pictures of this chessboard pattern in many different orientations. Be sure to cover *all degrees of freedom* across the different orientations and positions of the calibration pattern. Make sure that each image *fully* contains the chessboard pattern. Additionally, the corners in the chessboard pattern should be detected automatically and correctly using appropriate functions in the OpenCV library. Include the final set of images that you use for the calibration in your report.

    1. (5 points) Report the estimated intrinsic camera parameters, i.e., focal length(s), skew parameter and principal point along with error estimates if available.
    2. (5 points) Report the estimated extrinsic camera parameters, i.e., rotation matrix and translation vector for each of the selected images.

3. (5 points) Report the estimated radial distortion coefficients. Use the radial distor tion coefficients to undistort 5 of the raw images and include them in your report. Observe how straight lines at the corner of the images change upon application of the distortion coefficients. Comment briefly on this observation.
4. (5 points) Compute and report the re-projection error using the intrinsic and ex trinsic camera parameters for each of the 25 selected images. Plot the error using a bar chart. Also report the mean and standard deviation of the re-projection error.
5. (10 points) Plot figures showing corners detected in the image along with the corners after the re-projection onto the image for all the 25 images. Comment on how is the reprojection error computed.
6. (10 points) Compute the checkerboard plane normals $n^C_i$, $i \in \{1, ..25\}$ for each of the 25 selected images in the camera coordinate frame of reference ($O^c$).

5. (40 points) Camera-LIDAR Cross-Calibration:

Use this link to download the complete dataset. The dataset includes the RGB images and their corresponding LIDAR scans, the camera calibration parameters (intrinsic pa rameters, extrinsic parameters and distortion coefficients as returned by OpenCV) and the checkerboard plane normals in the camera reference frame, $n^C_i$. The dataset is ac quired by a monocular camera and a LIDAR sensor mounted on top of an autonomous vehicle. You can select any 25 corresponding images and LIDAR data points for this assignment. The size of the chessboard box is given to be 108mm. A sample image and the corresponding LIDAR scan are shown in Figure 1a and 1c. We have pre-processed the LIDAR scan points and extracted the LIDAR points on the chessboard as .pcd files. A sample of the extracted file is shown in Figure 1b. To cross-calibrate the LIDAR and the camera is to find the invertible Euclidean transformation (3D rotation and 3D translation) between the camera and LIDAR coordinate frames. Follow the steps below to complete the LIDAR-Camera cross-calibration.

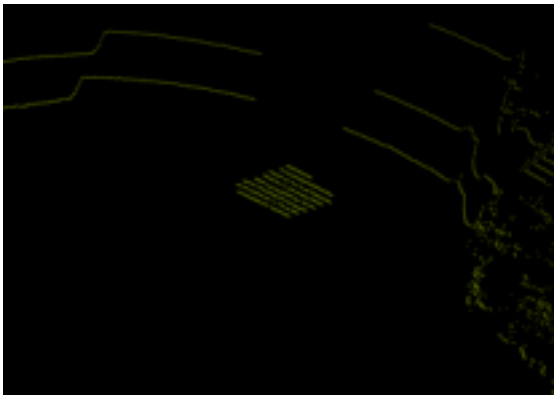
1. (10 points) Compute the chessboard plane normals $n^L_i$, $i \in \{1, . . . , 25\}$ and the corresponding offsets $d_i$, $i \in \{1, . . . , 25\}$ using the planar LIDAR points in each of the pcd files. You can estimate these by making use of singular value decomposition.
2. (10 points) Now that you have the plane normals $n^C_i$ and $n^L_i$ in camera and LIDAR frame of reference respectively for all the selected images, derive the set of equations that you would use to solve for estimating the transformation ${}^CT_L = [{}^CR_L | {}^Ct_L]$, i.e. the transformation from the LIDAR frame to the camera frame. Explain how the rotation and translation matrices are calculated. [*Hint*: You may refer to this thesis (Sec. 5) for deriving the necessary equations.]
3. (5 points) Using the above equations, implement the function that estimates the

transformation ${}^{C}T_{L}$. Recall that the rotation matrix has determinant +1. 4. (5 points) Use the estimated transformation ${}^{C}T_{L}$ to map LIDAR points to the camera frame of reference, then project them to the image plane using the intrinsic camera parameters. Are all points within the checkerboard pattern's boundary in each image?

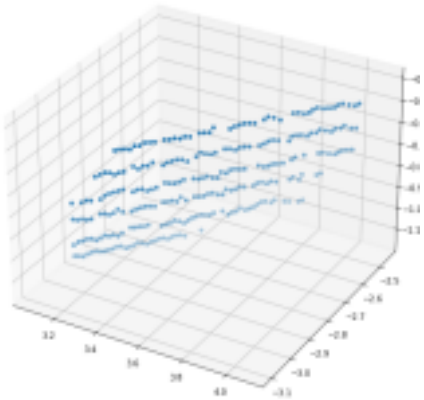
5. (10 points) Plot the normal vectors $n^{L}_{i}$, $n^{C}_{i}$, ${}^{C}R_{L}n^{L}_{i}$ for any 5 image and LIDAR scan pairs. Compute the cosine distance between the camera normal $n^{C}_{i}$ and the transformed LIDAR normal, ${}^{C}R_{L}n^{L}_{i}$ for *all* 38 image and LIDAR scan pairs, and plot the histogram of these errors. Report the average error with the standard deviation.



(a)



(b)



(c)

Figure 1: LIDAR-camera calibration. (a) Camera Frame. (b) Screenshot of LIDAR scan. The dense points in the center of the image are the points on the chessboard pattern. (c) Chessboard points extracted from the LIDAR Frame.

- **SOLUTIONS:::**

**Q1::**

**(a) :**

The coordinate transformation matrix for a rotation of $\theta$ about the Y-axis in three dimensions is given by:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

For $\theta = \pi/2$, the rotation matrix becomes:

$$R_y(\pi/2) = \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) \\ 0 & 1 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

**(b):**

We have the vector $\mathbf{v} = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$. After applying the rotation $R_y(\pi/2)$, the new coordinates $\mathbf{v}'$ are found by:

$$\mathbf{v}' = R_y(\pi/2) \cdot \mathbf{v} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 0 \cdot 5 + 0 \cdot 1 \\ 0 \cdot 2 + 1 \cdot 5 + 0 \cdot 1 \\ 0 \cdot 2 + 0 \cdot 5 + -1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ -2 \end{bmatrix}$$

The translation is then applied to this rotated vector by simply adding the translation vector

t= $\begin{bmatrix} -1 \\ 3 \\ 2 \end{bmatrix}$

to the origin $0$, yielding $t$ as the new position of the origin.

**(c)**

The rotation about the X-axis by $-\pi/2$ has its own rotation matrix:

$$R_x(-\pi/2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-\pi/2) & -\sin(-\pi/2) \\ 0 & \sin(-\pi/2) & \cos(-\pi/2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

The combined rotation matrix $R$ resulting from first rotating by $R_y(\pi/2)$ and then by $R_x(-\pi/2)$ is the matrix product:

$$R = R_x(-\pi/2) \cdot R_y(\pi/2)$$

The result is a 3x3 matrix, which is generally complicated to compute by hand. This matrix will have the information about the combined rotation's axis and angle encoded within it.

**(d)**

**Q2 :::**

Given a rotation R of angle θ about the axis u (a unit vector), then for any vector x, the rotated vector Rx is given by:

$$Rx = \cos(\theta)x + \sin(\theta)(u \times x) + (1 - \cos(\theta))(u \cdot x)u$$

The cross product $u \times x$ can be expressed as the matrix product $Ux$, where $U$ is the skew-symmetric matrix of $u$:

$$U = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$$

The term $(u \cdot x)u$ can be represented as the outer product $uu^T x$, where $uu^T$ is the matrix obtained by the outer product of $u$ with itself.

Now, we can rewrite the Rodrigues' formula using matrices:

$$R = \cos(\theta)I + \sin(\theta)U + (1 - \cos(\theta))(uu^T)$$

By expanding $\mathbf{u}\mathbf{u}^T\mathbf{x}$, we obtain:

- $(\mathbf{u}\mathbf{u}^T)\mathbf{x}$ multiplies each element of $\mathbf{x}$ by the corresponding element of $\mathbf{u}$ and sums the result for each component, which is equivalent to the projection of $\mathbf{x}$ onto $\mathbf{u}$ — in other words, $\mathbf{u} \cdot \mathbf{x}$.
- We then multiply this result by $\mathbf{u}$ again to return to a vector form, resulting in the vector $(\mathbf{u} \cdot \mathbf{x})\mathbf{u}$.
- Multiplying by the scalar $(1 - \cos(\theta))$ scales this vector accordingly.

Thus, $(1 - \cos(\theta))(\mathbf{u}\mathbf{u}^T)\mathbf{x}$ achieves the same result as $(1 - \cos(\theta))(\mathbf{u} \cdot \mathbf{x})\mathbf{u}$, confirming the equivalence of the two expressions. This step completes the proof that the given Rodrigues

## Q3 ::

. The equation provided, $x = K[R|t]X$

to Show -

Show that the homogeneous coordinate representation of image points $\mathbf{x}_1$ and $\mathbf{x}_2$ of $\mathbf{C}_1$ and $\mathbf{C}_2$ respectively, are related by an equation $\mathbf{x}_1 = \mathbf{H}\mathbf{x}_2$, where $\mathbf{H}$ is an invertible $3 \times 3$ matrix. Find $\mathbf{H}$ in terms of $\mathbf{K}_1, \mathbf{K}_2$ & $\mathbf{R}$.

- $x$ is the image point in homogeneous coordinates.
- $K$ is the intrinsic matrix containing parameters like focal length and optical center specific to the camera.
- $[R|t]$ are the extrinsic parameters, with $R$ being the rotation matrix and $t$ being the translation vector that relate the world coordinates to the camera coordinates.
- $X$ is the 3D point in the world coordinate system in homogeneous coordinates.

Given that camera C1 is aligned with the world coordinate system, its extrinsic matrix is the identity, $[I|0]$, and the equation for image formation simplifies to:

$$x_1 = K_1[I|0]X$$

For camera C2, which is rotated relative to C1 by a rotation matrix $R$, and assuming there is no translation between C1 and C2 (or the translation is not relevant for this part of the problem), the image formation equation is:

$$x_2 = K_2[R|0]X$$

To find the relationship between $x_1$ and $x_2$, we can express $X$ from the first equation and substitute it into the second equation. This process will reveal the matrix $H$ that relates $x_1$ and $x_2$.

Starting from $x_1 = K_1X$ and assuming $K_1$ is invertible, we can express $X$ as:

$$X = K_1^{-1}x_1$$

Now, let's substitute $X$ into the equation for $x_2$:

$$x_2 = K_2[R|0]K_1^{-1}x_1$$

Since the translation vector is zero, we can omit the translation part and the equation simplifies to:

$$x_2 = K_2RK_1^{-1}x_1$$

To express $x_1$ in terms of $x_2$, we simply invert the relationship:

$$x_1 = (K_2RK_1^{-1})^{-1}x_2$$

Using the property of matrix inversion for the product of matrices $(AB)^{-1} = B^{-1}A^{-1}$, we have:

$$x_1 = K_1 R K_2^{-1} x_2$$

Therefore, the matrix $H$ that relates $x_1$ to $x_2$ is given by:

$$H = K_1 R K_2^{-1}$$

This matrix $H$ is invertible as long as $K_1$, $R$, and $K_2$ are invertible, which is typically the case since $K_1$ and $K_2$ are intrinsic matrices of cameras (which are generally invertible), and $R$ is a rotation matrix (which is always invertible with the inverse being its transpose).

So the final homogeneous coordinate relationship between the image points $x_1$ and $x_2$ from cameras C1 and C2 respectively is:

$$x_1 = H x_2$$

where

$$H = K_1 R K_2^{-1}$$

Thus, $H$ is an invertible 3×3 matrix that relates the image points $x_1$ and $x_2$ in homogeneous coordinates from cameras C1 and C2, respectively. It encapsulates the intrinsic parameters of both cameras and the rotation between the two camera frames.

The invertibility of $H$ is ensured because:

- Intrinsic matrices $K_1$ and $K_2$ are invertible. These matrices are usually upper triangular with non-zero diagonal entries (focal lengths and skew coefficients), and hence they are non-singular and invertible.
- $R$ is a rotation matrix, which is always invertible, with the inverse being equal to its transpose. Since rotation matrices represent orthogonal transformations, they have determinant +1 or -1 and thus are guaranteed to be invertible.

This completes the derivation and explanation of how $H$ is found and what it represents in the context of two cameras with a rotational difference.

**Q4::** Answers in code

CODE::

```
#!/usr/bin/env python

import cv2
import numpy as np
import os
import glob

# Defining the dimensions of checkerboard
CHECKERBOARD = (6,9)
criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Creating vector to store vectors of 3D points for each checkerboard image
```

```
objpoints = []
# Creating vector to store vectors of 2D points for each checkerboard image
imgpoints = []


# Defining the world coordinates for 3D points
objp = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD[1], 3),
np.float32)
objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0],
0:CHECKERBOARD[1]].T.reshape(-1, 2)
prev_img_shape = None

# Extracting path of individual image stored in a given directory
images = glob.glob('./images/*.jpg')
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    # If desired number of corners are found in the image then ret = true
    ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD,
cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK +
cv2.CALIB_CB_NORMALIZE_IMAGE)

    """
    If desired number of corner are detected,
    we refine the pixel coordinates and display
    them on the images of checker board
    """
    if ret == True:
        objpoints.append(objp)
        # refining pixel coordinates for given 2d points.
        corners2 = cv2.cornerSubPix(gray, corners, (11,11),(-1,-1), criteria)
```

```
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2,
ret)

    cv2.imshow('img',img)
    cv2.waitKey(0)

cv2.destroyAllWindows()

h,w = img.shape[:2]

"""
Performing camera calibration by
passing the value of known 3D points (objpoints)
and corresponding pixel coordinates of the
detected corners (imgpoints)
"""
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)

print("Camera matrix : \n")
print(mtx)
print("dist : \n")
print(dist)
print("rvecs : \n")
print(rvecs)
print("tvecs : \n")
print(tvecs)
```

**Q5::** Answers in code

```
# %%
#%pip install numpy open3d

# %%
import numpy as np
import open3d as o3d
import os


# %%


# %%
# Path to the directory containing the .pcd files

# pcd_dir = 'path_to_your_pcd_files_directory'

pcd_dir = "./lidar_scans"

# Assuming the .pcd files are named in a sequential order
pcd_files = [f for f in os.listdir(pcd_dir) if f.endswith('.pcd')]

# Sort the files to maintain the sequence
pcd_files.sort()
```

```
# %%
# Let's list the contents of the 'camera_parameters' directory to see what files are there
camera_parameters_path = "./camera_parameters"
camera_parameters_files = os.listdir("./camera_parameters")

# %%
# Function to read normals from a text file
def read_normals(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        normals = np.array([list(map(float, line.strip().split())) for line in lines])
    return normals
# Initialize a dictionary to hold frame normals
frame_normals = {}

# Iterate over each folder in 'camera_parameters', read the normals, and store them in the dictionary
for folder_name in camera_parameters_files:
    if folder_name.endswith('.jpeg'):  # Ensure we're working with the correct directories
        normals_file_path = os.path.join(camera_parameters_path, folder_name, 'camera_normals.txt')
        if os.path.isfile(normals_file_path):  # Check if the normals file exists
            normals = read_normals(normals_file_path)
            frame_normals[folder_name] = normals

# For demonstration, let's print the normals for one of the frames
sample_frame = list(frame_normals.keys())[0]
frame_normals[sample_frame]
```

```
# %%
# Function to read normals from a text file
def read_rotation_matrix(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        normals = np.array([list(map(float, line.strip().split())) for line in lines])
    return normals
# Initialize a dictionary to hold frame normals
frames_rotation_matrix = {}

# Iterate over each folder in 'camera_parameters', read the rotation_matrix, and
store them in the dictionary
for folder_name in camera_parameters_files:
    if folder_name.endswith('.jpeg'):  # Ensure we're working with the correct
directories

        cam_normals_file_path = os.path.join(camera_parameters_path,
folder_name, 'camera_normals.txt')
        cam_rotation_matrix_file_path = os.path.join(camera_parameters_path,
folder_name, 'rotation_matrix.txt')
        cam_rotation_vector_file_path = os.path.join(camera_parameters_path,
folder_name, 'rotation_vectors.txt')
        cam_translation_vector_file_path = os.path.join(camera_parameters_path,
folder_name, 'translation_vectors.txt')

        if os.path.isfile(cam_rotation_matrix_file_path):  # Check if the normals
file exists
            rotation_matrix = read_rotation_matrix(cam_rotation_matrix_file_path)
            frames_rotation_matrix[folder_name] = rotation_matrix

# For demonstration, let's print the normals for one of the frames
```

```
sample_frame = list(frames_rotation_matrix.keys())[0]
frames_rotation_matrix[sample_frame]



# %%
camera_parameters_files

# %%
pcd_files

# for i in range(len(pcd_files)) :
#     print(pcd_files[i])

# %%
frame_normals

# %%

def compute_plane_normal_and_offset(points):
    # Subtract the mean to center the points
    points_centered = points - np.mean(points, axis=0)

    # Calculate the covariance matrix
    H = np.dot(points_centered.T, points_centered)

    # Perform SVD
    U, S, Vt = np.linalg.svd(H)

    # The normal to the chessboard plane is the last column of V
    normal = Vt[-1, :]

    # The offset can be found by projecting the mean of the points onto the
```

```
normal vector
    offset = np.dot(normal, np.mean(points, axis=0))

    return normal, offset




# %%

# # Storage for normals and offsets
# normals = []
# offsets = []

# # Process each .pcd file
# for idx, pcd_file in enumerate(pcd_files):
#     try:
#         # Load the point cloud data
#         pcd_path = os.path.join(pcd_dir, pcd_file)
#         pcd = o3d.io.read_point_cloud(pcd_path)

#         # Compute the normal and offset of the plane
#         normal, offset =
compute_plane_normal_and_offset(np.asarray(pcd.points))

#         # Store the results
#         normals.append(normal)
#         offsets.append(offset)

#         print(f'{idx+1} Processed file {pcd_file}: Normal - {normal}')
#         print(f'{idx+1} Processed file {pcd_file}: Offset - {offset}')
```

```
#     except Exception as e:
#         print(f'An error occurred while processing file {pcd_file}: {e}')

# # At this point, 'normals' and 'offsets' contain the plane parameters for all .pcd
files



# %%

# Storage for normals and offsets in a dictionary
LIDAR_plane_parameters = {}

# Process each .pcd file
for idx, pcd_file in enumerate(pcd_files):
    try:
        # Load the point cloud data
        pcd_path = os.path.join(pcd_dir, pcd_file)
        pcd = o3d.io.read_point_cloud(pcd_path)

        # Compute the normal and offset of the plane
        normal, offset =
compute_plane_normal_and_offset(np.asarray(pcd.points))

        # Store the results in the dictionary with the file name as the key
        LIDAR_plane_parameters[pcd_file] = {'normal': normal, 'offset': offset}

        print(f'{idx+1} Processed file {pcd_file}: Normal - {normal}')
        print(f'{idx+1} Processed file {pcd_file}: Offset - {offset}')

    except Exception as e:
        print(f'An error occurred while processing file {pcd_file}: {e}')
```

```
print(LIDAR_plane_parameters)

# At this point, 'plane_parameters' contains the plane parameters for all .pcd files keyed by their file names


# %%


# %%


# %% [markdown]
#

# %%



import numpy as np

def estimate_transformation(normals_L, normals_C, points_L, points_C):
    """
    Estimate the transformation from the LIDAR frame to the camera frame.

    Parameters:
    normals_L: List of normals in the LIDAR frame.
    normals_C: List of normals in the camera frame.
    points_L: List of points in the LIDAR frame.
    points_C: List of points in the camera frame.
```

```
Returns:
C_RL: Rotation matrix from LIDAR to camera frame.
C_tL: Translation vector from LIDAR to camera frame.
"""
normals_L = np.array(normals_L)
normals_C = np.array(normals_C)
points_L = np.array(points_L)
points_C = np.array(points_C)

# Compute cross-covariance matrix S
S = np.zeros((3, 3))
for n_L, n_C in zip(normals_L, normals_C):
    S += np.outer(n_L, n_C)

# Perform SVD on S
U, _, Vt = np.linalg.svd(S)

# Ensure a right-handed coordinate system
VU = Vt.T @ U.T
if np.linalg.det(VU) < 0:
    Vt[-1, :] *= -1
    VU = Vt.T @ U.T

C_RL = VU  # Rotation matrix

# Translate centroids
centroid_L = np.mean(points_L @ C_RL, axis=0)
centroid_C = np.mean(points_C, axis=0)
C_tL = centroid_C - centroid_L  # Translation vector

return C_RL, C_tL
```

```
def project_to_image(points_L, C_RL, C_tL, camera_intrinsics):
    """
    Project LIDAR points to the camera image plane.

    Parameters:
    points_L: List of points in the LIDAR frame.
    C_RL: Rotation matrix from LIDAR to camera frame.
    C_tL: Translation vector from LIDAR to camera frame.
    camera_intrinsics: Camera intrinsic parameters matrix.

    Returns:
    points_image: Points projected onto the camera image plane.
    """
    points_L_transformed = (points_L @ C_RL.T) + C_tL
    points_image = points_L_transformed @ camera_intrinsics.T
    points_image /= points_image[:, -1:]  # Homogenize
    return points_image[:, :2]  # Return only x, y coordinates


# It seems we have a NameError because numpy is not imported in this new
context.
# We need to import numpy again and try reading the camera parameters.

import numpy as np

# Redefine the reading functions with numpy imported
def read_camera_intrinsics(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        intrinsics = np.array([list(map(float, line.split())) for line in lines])
```

```
    return intrinsics

def read_distortion_coeffs(file_path):
    with open(file_path, 'r') as file:
        line = file.readline()
        coeffs = np.array(list(map(float, line.split())))
    return coeffs

# Attempt to read the camera intrinsic matrix and distortion coefficients again
camera_intrinsics = read_camera_intrinsics(camera_intrinsics_file)
distortion_coeffs = read_distortion_coeffs(distortion_coeffs_file)

camera_intrinsics, distortion_coeffs
```