CV A2 - Report

Harsh Rajput - 2022201

1. 2D Geometry

Given Vector (In Homogeneous Form):

$$V = \begin{bmatrix} 3 \\ -1 \\ 4 \\ 1 \end{bmatrix}$$

Transformations (In Homogeneous Form):

1. A rotation of $-\frac{\pi}{6}$ about the Y-axis.

$$R_y(-\pi/6) = \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\pi/6) & 0 & \cos(-\pi/6) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. A rotation of $\frac{\pi}{4}$ about the X-axis.

$$R_x(\pi/4) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\pi/4) & -\sin(\pi/4) & 0 \\ 0 & \sin(\pi/4) & \cos(\pi/4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. A reflection across the XZ-plane.

$$R_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. A translation by $\begin{bmatrix} 1 & 0 & -2 \end{bmatrix}$.

$$t = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) Overall Coordinate Transformation Matrix

$$T = t \cdot R_{xz} \cdot R_x(\pi/4) \cdot R_y(-\pi/6)$$

$$=> t \cdot R_{xz} \cdot \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) & 0 \\ \sin(-\pi/6)\sin(\pi/4) & \cos(\pi/4) & -\cos(-\pi/6)\sin(\pi/4) & 0 \\ -\sin(-\pi/6)\cos(\pi/4) & \sin(\pi/4) & \cos(-\pi/6)\cos(\pi/4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$=> t \cdot \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) & 0 \\ -\sin(-\pi/6)\sin(\pi/4) & -\cos(\pi/4) & \cos(-\pi/6)\sin(\pi/4) & 0 \\ -\sin(-\pi/6)\cos(\pi/4) & \sin(\pi/4) & \cos(-\pi/6)\cos(\pi/4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$=> \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) & 1 \\ -\sin(-\pi/6)\sin(\pi/4) & -\cos(\pi/4) & \cos(-\pi/6)\sin(\pi/4) & 0 \\ -\sin(-\pi/6)\cos(\pi/4) & \sin(\pi/4) & \cos(-\pi/6)\cos(\pi/4) & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} \sqrt{3}/2 & 0 & -1/2 & 1 \\ 1/2\sqrt{2} & -1/\sqrt{2} & \sqrt{3}/2\sqrt{2} & 0 \\ 1/2\sqrt{2} & 1/\sqrt{2} & \sqrt{3}/2\sqrt{2} & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8660254 & 0 & -0.5 & 1 \\ 0.35355339 & -0.70710678 & 0.61237244 & 0 \\ 0.35355339 & 0.70710678 & 0.61237244 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Transformed Vector and Origin

Transformed Vector (In Homogeneous Form):

$$V' = T \cdot V = \begin{bmatrix} 1.59807621 \\ 4.2172567 \\ 0.80304313 \\ 1 \end{bmatrix}$$

Since Rotation and Reflection Would not affect origin and only translation would. Therefore origin (In Homogeneous Form)

$$O = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

is Mapped to

$$O' = T \cdot O = \begin{bmatrix} 1 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

(c) Axis and Angle of Combined Rotation

Rotation Matrix (In Homogeneous Form):

$$R_x(\pi/4)\cdot R_y(-\pi/6) = \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) & 0 \\ \sin(-\pi/6)\sin(\pi/4) & \cos(\pi/4) & -\cos(-\pi/6)\sin(\pi/4) & 0 \\ -\sin(-\pi/6)\cos(\pi/4) & \sin(\pi/4) & \cos(-\pi/6)\cos(\pi/4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In Non-Homogeneous Form:

$$R = \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(-\pi/6) \\ \sin(-\pi/6)\sin(\pi/4) & \cos(\pi/4) & -\cos(-\pi/6)\sin(\pi/4) \\ -\sin(-\pi/6)\cos(\pi/4) & \sin(\pi/4) & \cos(-\pi/6)\cos(\pi/4) \end{bmatrix}$$

$$= \begin{bmatrix} 0.8660254 & 0. & -0.5 \\ -0.35355339 & 0.70710678 & -0.61237244 \\ 0.35355339 & 0.70710678 & 0.61237244 \end{bmatrix}$$

The angle of rotation θ about this axis is given by:

$$\theta = \cos^{-1}\left(\frac{\text{trace}(R)-1}{2}\right) = \cos^{-1}\left(\frac{(\cos(-\pi/6)+\cos(\pi/4)+\cos(-\pi/6)\cos(\pi/4))-1}{2}\right) = 0.9363$$

The axis of rotation n is given by:

$$n = \frac{1}{2\sin\theta}\begin{bmatrix} R_{32}-R_{23} \\ R_{13}-R_{31} \\ R_{21}-R_{12} \end{bmatrix} = \frac{1}{2\sin\theta}\begin{bmatrix} \sin(\pi/4)-(-\cos(-\pi/6)\sin(\pi/4)) \\ \sin(-\pi/6)-(-\sin(-\pi/6)\cos(\pi/4)) \\ \sin(-\pi/6)\sin(\pi/4)-0 \end{bmatrix} = \begin{bmatrix} 0.81916073 \\ -0.52990408 \\ -0.21949345 \end{bmatrix}$$

(d) Verification Using Rodrigues' Formula

$$[n]_\times = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.21949345 & -0.52990408 \\ -0.21949345 & 0 & -0.81916073 \\ 0.52990408 & 0.81916073 & 0 \end{bmatrix}$$

Using Rodrigues' rotation formula :

$$R' = I + \sin\theta[n]_\times + (1-\cos\theta)[n]_\times^2$$

We Get

$$R' = \begin{bmatrix} 0.8660254 & 0. & -0.5 \\ -0.35355339 & 0.70710678 & -0.61237244 \\ 0.35355339 & 0.70710678 & 0.61237244 \end{bmatrix} = R$$

Rodrigues' rotation martix is Same as Combined rotation Matrix. Hence Verified

2. Image Formation

We start with the given image formation equation:

$$x = K[R|t]X$$

For two cameras C_1 and C_2, let their intrinsic parameter matrices be K_1 and K_2, respectively. For the first camera C_1, assuming it is aligned with the world coordinate frame:

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

Since $[I|0]X = X$, we get:

$$x_1 = K_1X$$

For the second camera C_2, its orientation is given by a pure rotation R, meaning there is no translation:

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

Since $[R|0]X = RX$, we get:

$$x_2 = K_2RX$$

Since C_2 is obtained by rotating C_1 by a *pure 3D rotation matrix* R, the world coordinate frame is rotated to align with C_2:

$$X_2 = RX_1$$

Since $X_1 = X_2$, this implies:

$$X = RX.$$

Since $x_1 = K_1X$, we solve for X:

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

Substituting this into the equation for x_2:

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

Rearranging:

$$x_1 = (K_1R^TK_2^{-1})x_2$$

Thus, we have:

$$H = K_1R^TK_2^{-1}$$

which shows that the image points x_1 and x_2 are related by the homography:

$$x_1 = Hx_2$$

1. H **dimension**

- K_1 is a 3×3 intrinsic matrix.
- R^T is a 3×3 rotation matrix.

- K_2^{-1} is a 3×3 inverse intrinsic matrix.
- Since the product of three 3×3 matrices is still 3×3, H is also 3×3.

2. **Invertibility of H**

- K_1 and K_2 are invertible because they are upper triangular matrices with nonzero diagonal entries (focal lengths).
- R^T is invertible because it is an orthogonal matrix ($R^T R = I$).
- The product of invertible matrices is also invertible, so H is invertible.
- Its inverse is:

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

Correctness

Substituting $H = K_1 R^T K_2^{-1}$ into $x_1 = H x_2$:

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

Since $x_2 = K_2 X$:

$$K_1 R^T K_2^{-1} K_2 X = K_1 R^T X$$

Using $R^T X = X$, we get:

$$K_1 X = x_1$$

which confirms the correctness.

Conclusion

The two image points x_1 and x_2 are related by the homography:

$$x_1 = H x_2, \quad H = K_1 R^T K_2^{-1}$$

where H is an invertible 3×3 matrix.

3. Camera Calibration Report

Overview

The objective of this project is to calibrate a camera using a chessboard calibration pattern. The process includes capturing images in various orientations, detecting chessboard corners, estimating intrinsic and extrinsic parameters, computing distortion coefficients, and evaluating the calibration accuracy.

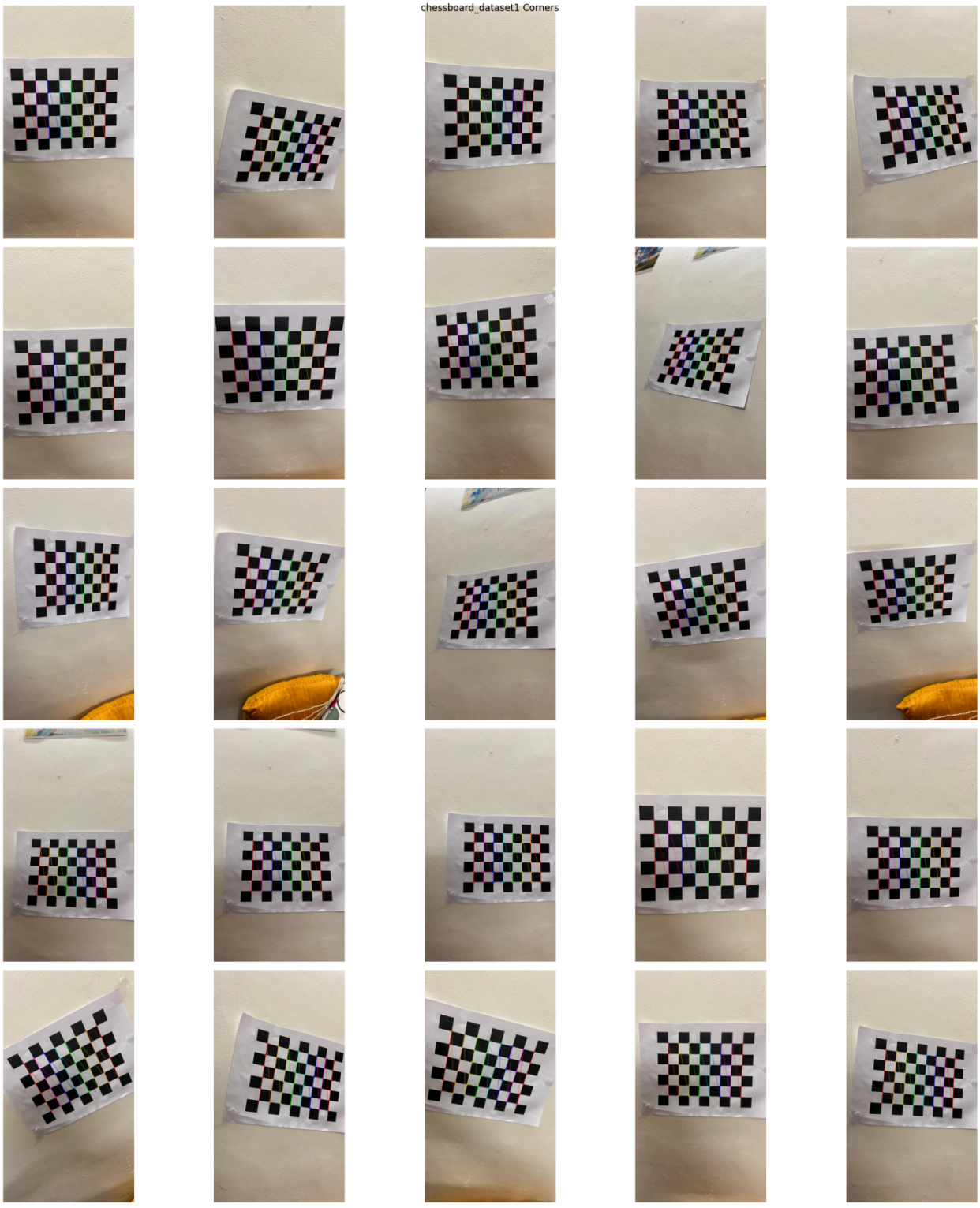
Step 0: Data Collection

Dataset Provided

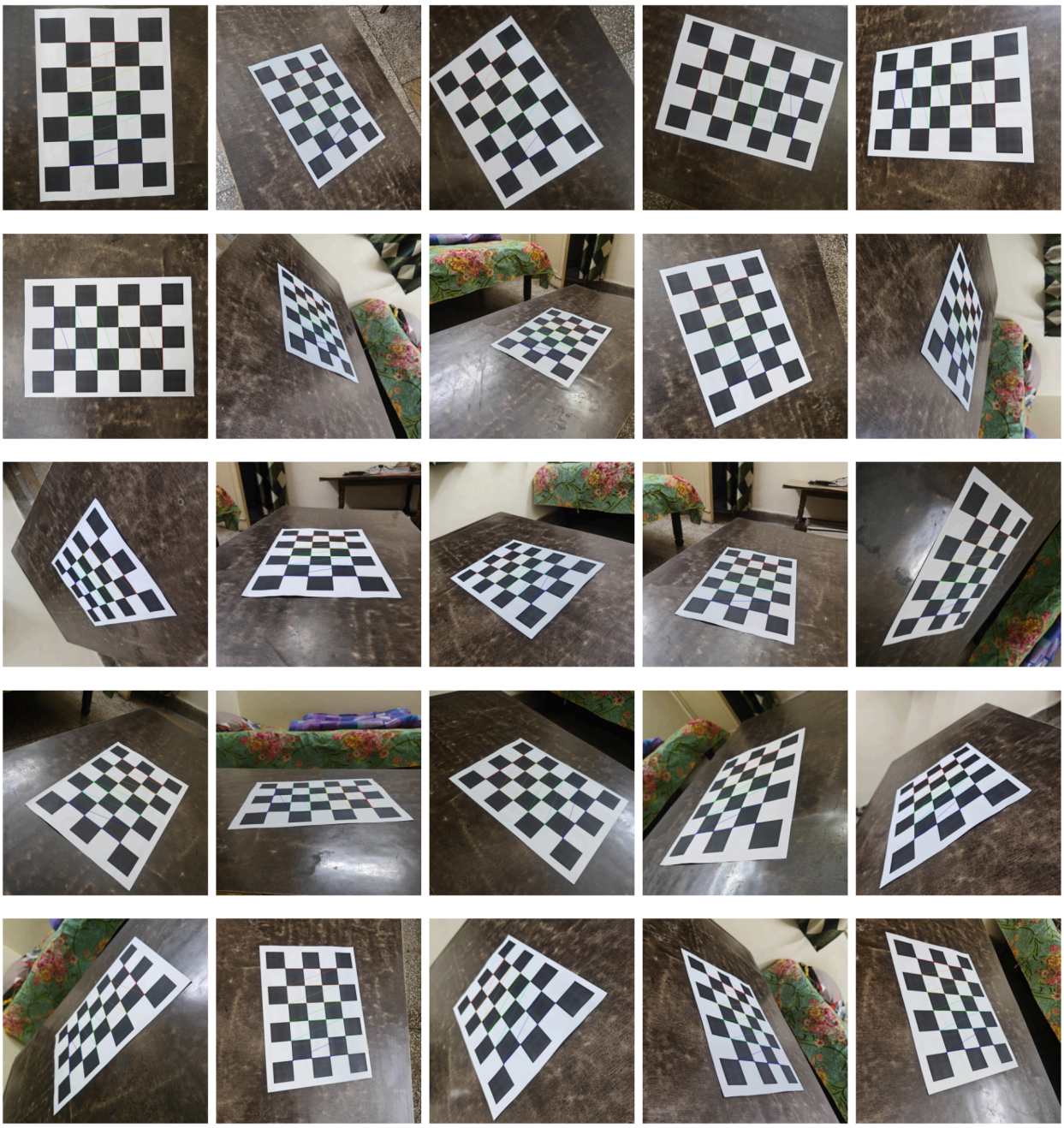
- 25 images from the dataset.

Self-Collected Dataset

- 25 images taken using a **mobile camera**.
- The camera was **stationary on a table**.
- A **printed chessboard pattern** was placed on a **hard, planar surface**.
- Images were captured from **various angles** to cover all degrees of freedom.


chessboard_dataset1 Corners

chessboard_dataset2 Corners



Step 1: Estimation of Intrinsic Camera Parameters

Dataset 1 (Provided)

```
print("fx, fy :", K1[0, 0], K1[1, 1])
print("skew :", K1[0, 1])
print("cx, cy :", K1[0, 2], K1[1, 2])
```

```
fx, fy : 955.9831309528636 956.7749752283094
skew : 0.0
cx, cy : 368.88206140847524 648.8286752813
```

Dataset 2 (Self Clicked)

```
print("fx, fy :", K2[0, 0], K2[1, 1])
print("skew :", K2[0, 1])
print("cx, cy :", K2[0, 2], K2[1, 2])
```

```
fx, fy : 1459.6591988782702 1458.1478364838722
skew : 0.0
cx, cy : 783.519188003288 738.2615409886507
```

Step 2: Estimation of Extrinsic Camera Parameters

Dataset 1 (Provided)

```
for i in range(2):
    print("Image ID:", i + 1)
    print("Rotation Matrix: \n", cv2.Rodrigues(rvecs1[i])[0])
    print("Translation Vector: ", tvecs1[i].flatten())
```

```
Image ID: 1
Rotation Matrix:
 [[ 0.01389117 -0.99696989 -0.076538  ]
 [ 0.98896274  0.0249913  -0.14604153]
 [ 0.14751179 -0.07366454  0.98631324]]
Translation Vector:  [ 3.04338964 -3.75226511 15.17125353]
Image ID: 2
Rotation Matrix:
 [[ 0.242051    0.89020748 -0.38592999]
 [-0.94345703  0.12309245 -0.3077939 ]
 [-0.22649536  0.43861018  0.86966716]]
Translation Vector:  [-2.70414834  4.09783203 15.03440681]
```

Dataset 2 (Self Clicked)

```
for i in range(2):
    print("Image ID:", i + 1)
    print("Rotation Matrix: \n", cv2.Rodrigues(rvecs2[i])[0])
    print("Translation Vector: ", tvecs2[i].flatten())
```

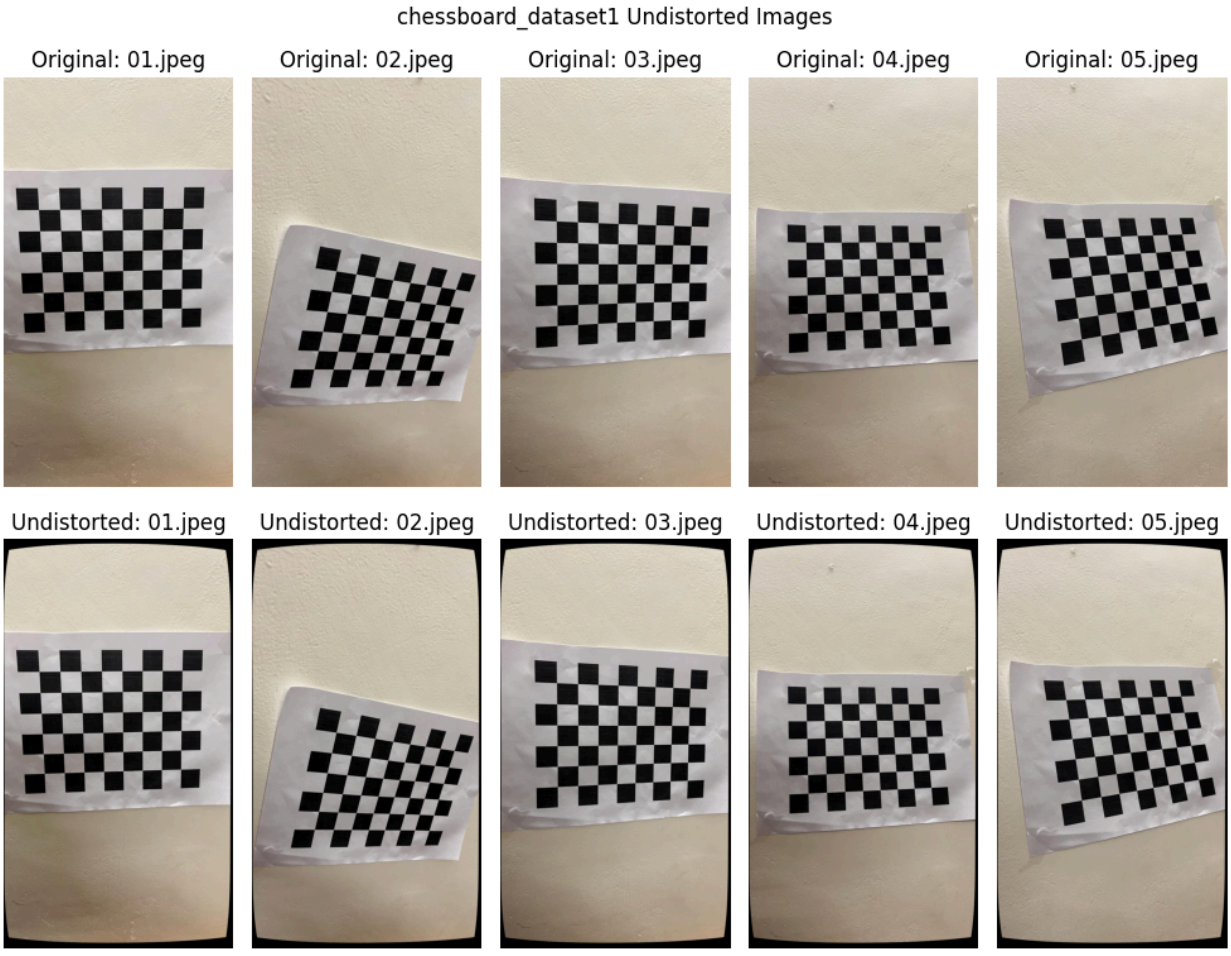
```
Image ID: 1
Rotation Matrix:
 [[ 0.99757039  0.0301895  -0.06278457]
 [-0.02862336  0.99925992  0.02569651]
 [ 0.06351387 -0.02383697  0.99769624]]
Translation Vector:  [-1.70164878 -2.47150213  8.05453054]
Image ID: 2
Rotation Matrix:
 [[ 0.77430167  0.53011168 -0.34559879]
 [-0.38743648  0.82892792  0.40344948]
 [ 0.50034977 -0.17849402  0.84722488]]
Translation Vector:  [-2.56072571 -1.45048045 11.69053884]
```

Step 3: Estimation of Distortion Coefficients

Dataset 1 (Provided)

```
print("Distortion coefficients: ", dist1)
undistort_images('chessboard_dataset1', K1, dist1, num_images=5)
```

```
Distortion coefficients:  [[ 0.21565351 -1.03904569  0.0027891   0.00682133  1.58635031]]
```



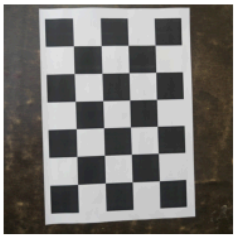
Dataset 2 (Self Clicked)

```
print("Distortion coefficients: ", dist2)
undistort_images('chessboard_dataset2', K2, dist2, num_images=5)
```

```
Distortion coefficients:  [[ 3.20436629e-01 -2.34227577e+00 -8.49016273e-03  4.82851853e-03
   6.49099743e+00]]
```

chessboard_dataset2 Undistorted Images

Original: 01.jpeg

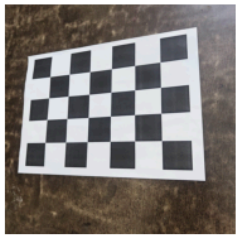


Original: 02.jpeg

Original: 03.jpeg

Original: 04.jpeg

Original: 05.jpeg



Undistorted: 01.jpeg

Undistorted: 02.jpeg

Undistorted: 03.jpeg

Undistorted: 04.jpeg

Undistorted: 05.jpeg

Observations:

- Straight lines near chessboard corners became straighter after distortion correction.
- Slight improvement in perspective correctness.

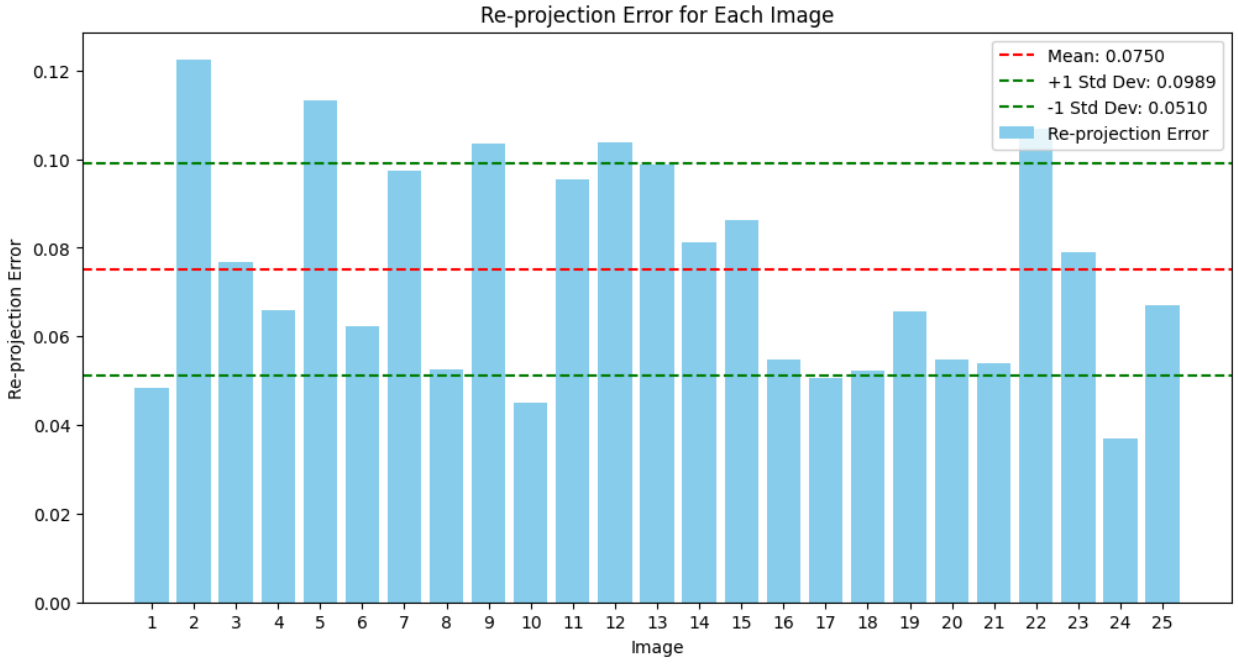
Step 4: Re-Projection Error Computation

The **re-projection error** is computed as the Euclidean distance between the detected chessboard corners and their re-projected locations.

Dataset 1 (Provided)

```
    print(errors1)
    print("Mean Reprojection Error:", np.mean(errors1))
    print("Standard Deviation of Reprojection Error:", np.std(errors1))
    Visualize_error(errors1)
```

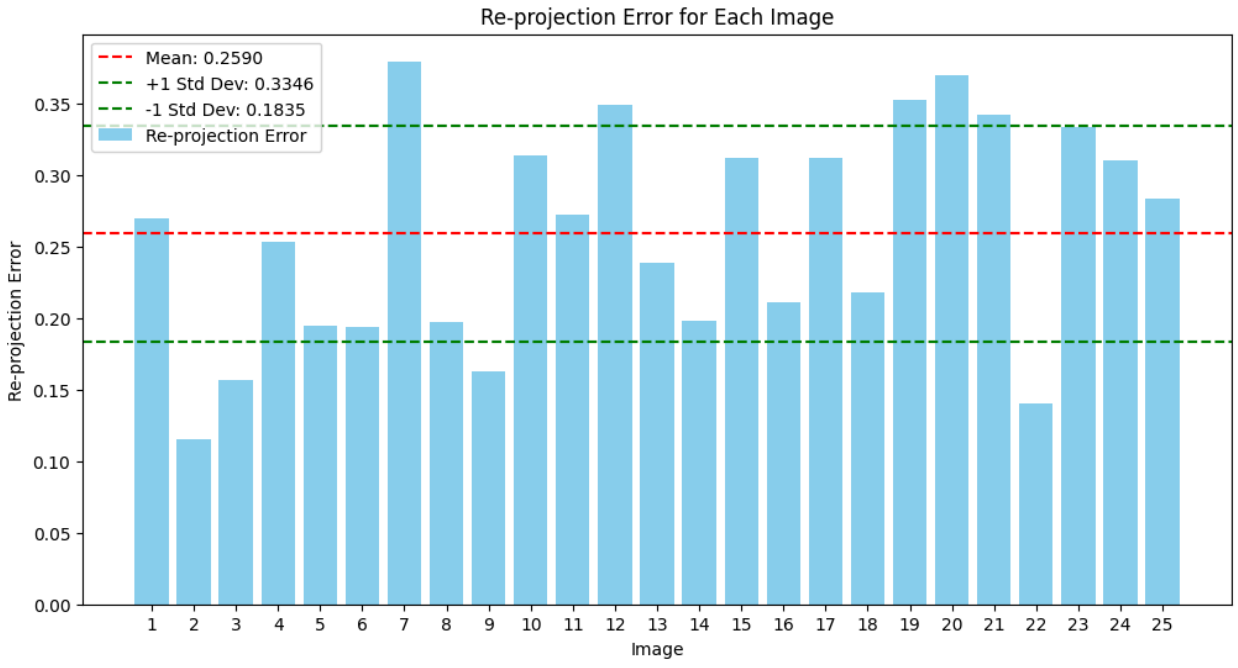
```
[0.04834708 0.12252298 0.0767687  0.06594681 0.11313981 0.06232144
 0.09731898 0.05247232 0.10364209 0.04492355 0.09541289 0.10373358
 0.09867753 0.08114601 0.08614456 0.05485417 0.05065939 0.0523607
 0.0657168  0.05471184 0.05399372 0.10678754 0.07906042 0.03701477
 0.06702289]
Mean Reprojection Error: 0.07498802315303951
Standard Deviation of Reprojection Error: 0.023956747838967504
```



Dataset 2 (Self Clicked)

```
print(errors2)
print("Mean Reprojection Error:", np.mean(errors2))
print("Standard Deviation of Reprojection Error:", np.std(errors2))
Visualize_error(errors2)
```

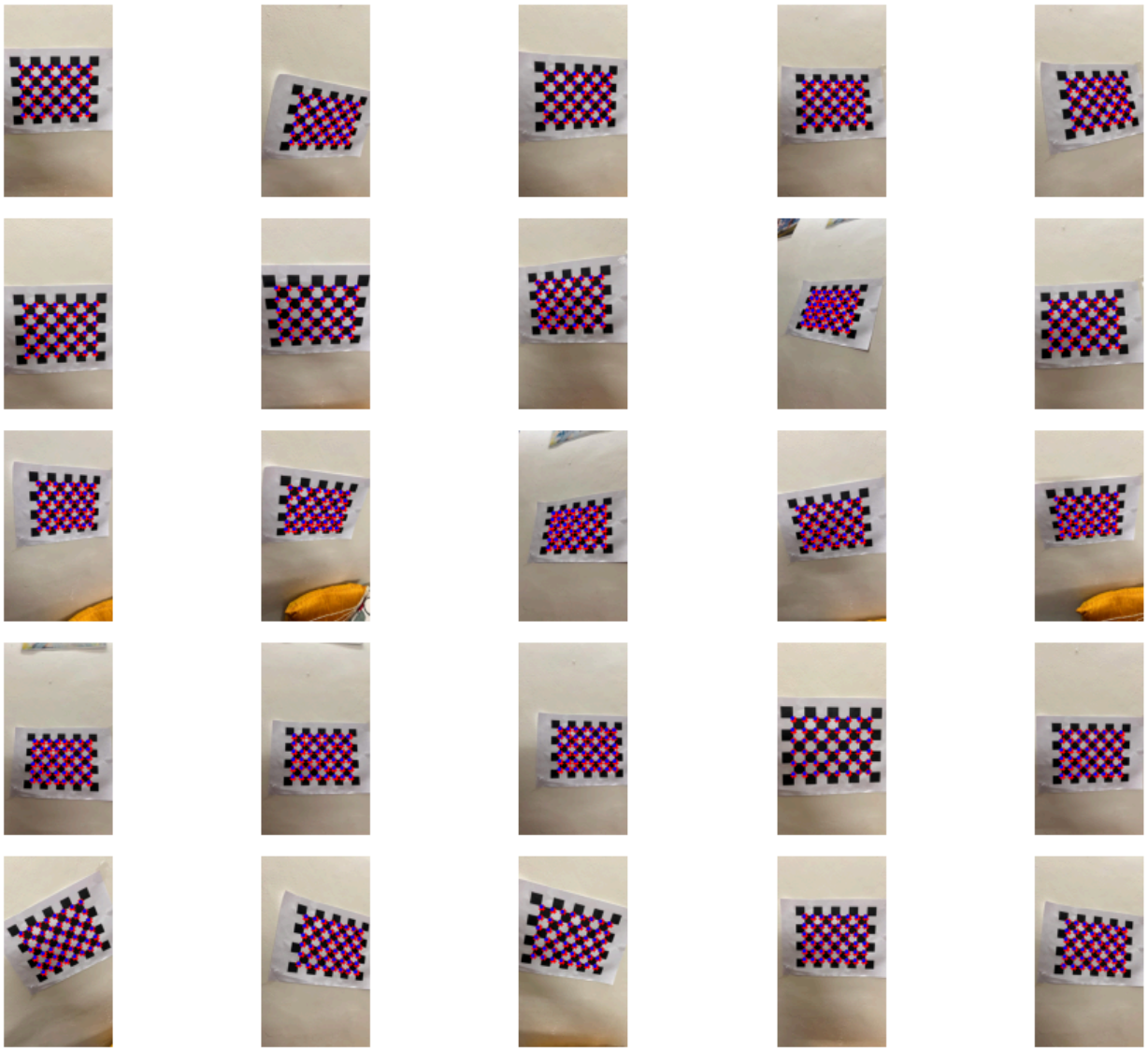
```
[0.26973014 0.11495426 0.15707367 0.25287859 0.19482172 0.19358058
 0.37930316 0.19710415 0.16288032 0.31342164 0.27195573 0.34869904
 0.23869214 0.19829061 0.31146482 0.21072385 0.31166255 0.21806065
 0.35199028 0.36993559 0.34166112 0.14035206 0.33341657 0.31040949
 0.28305135]
Mean Reprojection Error: 0.2590445629091397
Standard Deviation of Reprojection Error: 0.07554113205305335
```

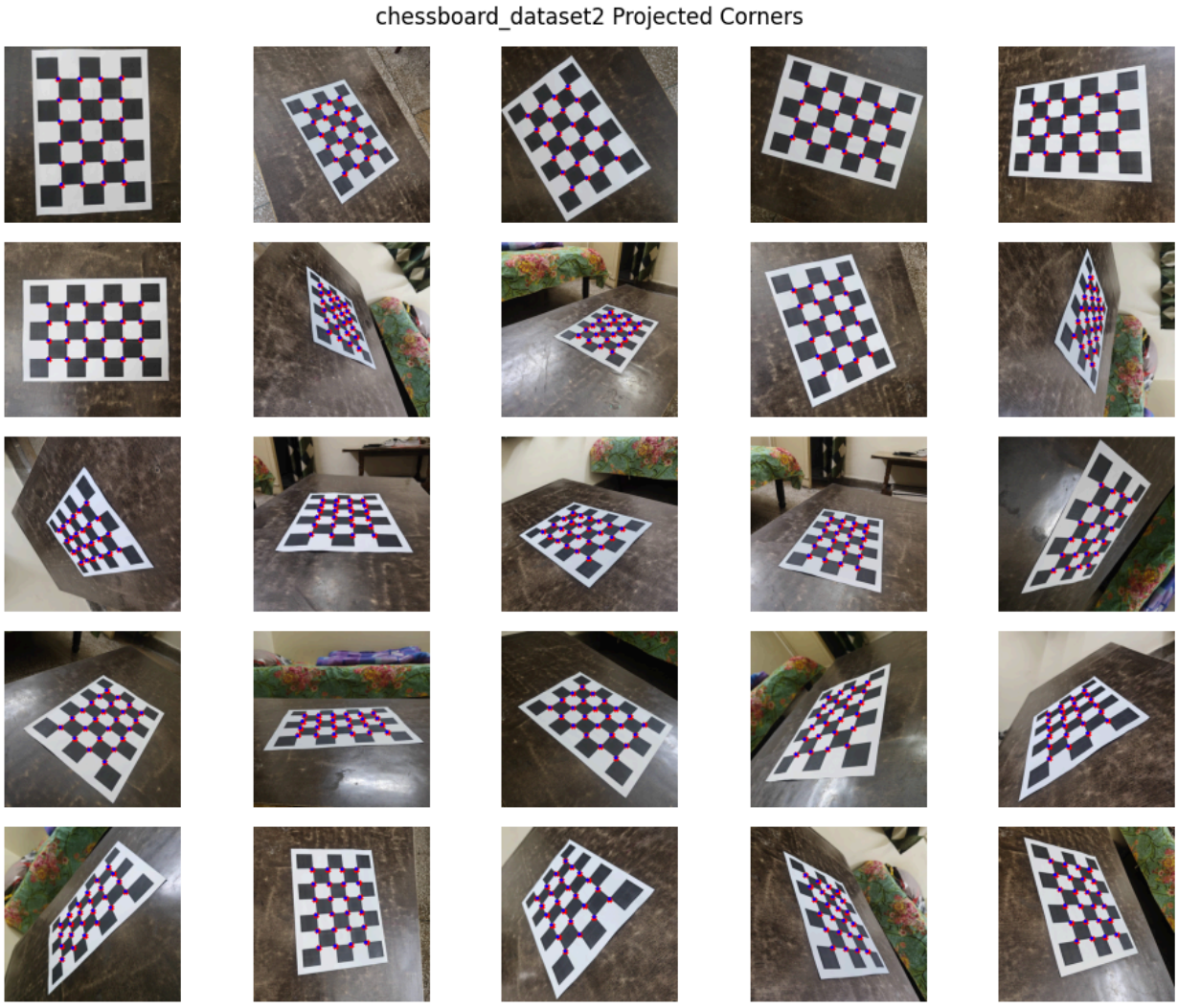


Step 5: Corner Detection and Re-Projection Visualization

- Chessboard corners detected using `cv2.findChessboardCorners()`.
- After calibration, the estimated corner positions were re-projected onto the original images.

chessboard_dataset1 Projected Corners





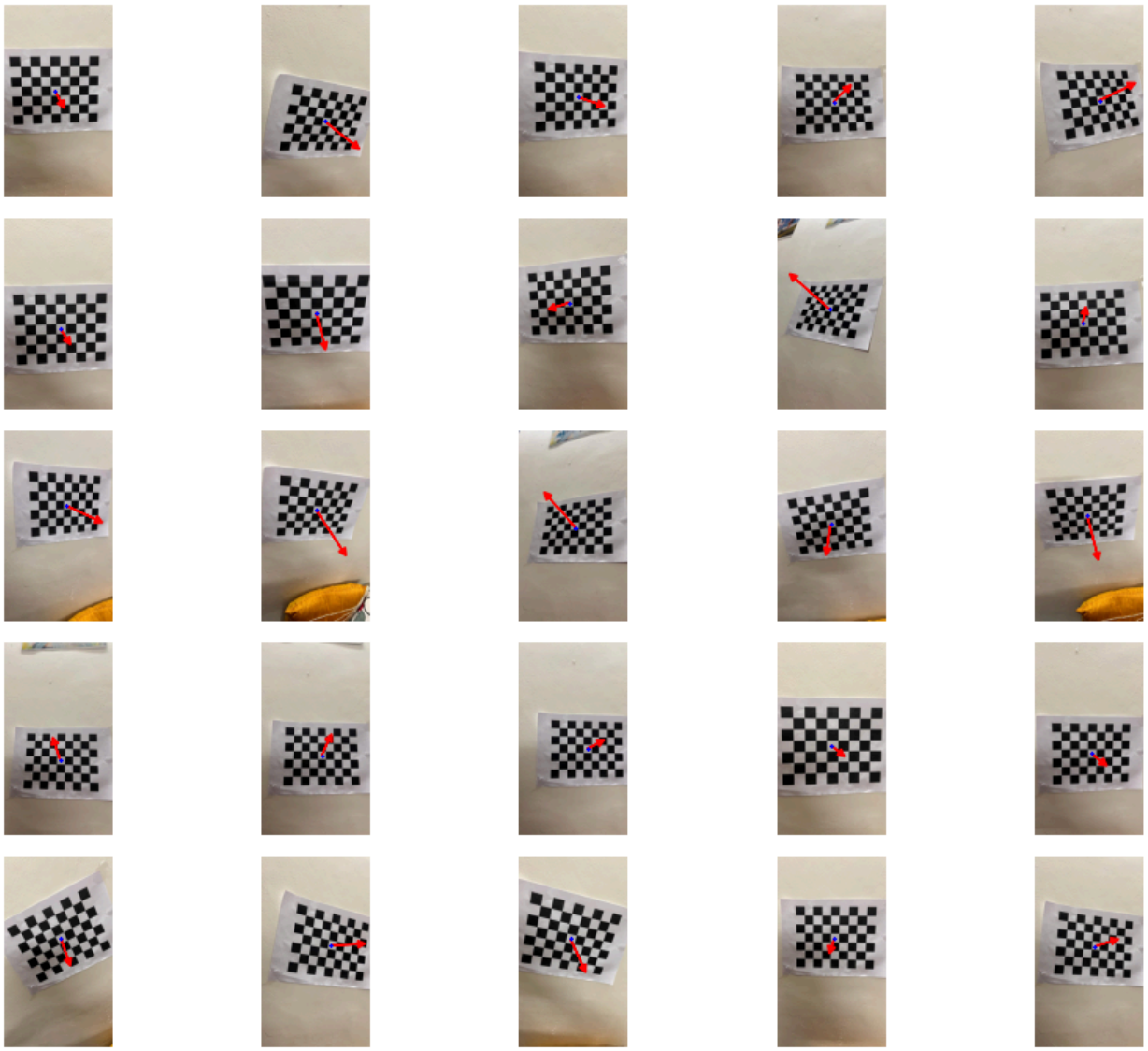
Observations:

- The re-projection error is smaller for images with good lighting and perspective.
- Corners closer to image edges have slightly higher errors.

Step 6: Computation of Checkerboard Plane Normals

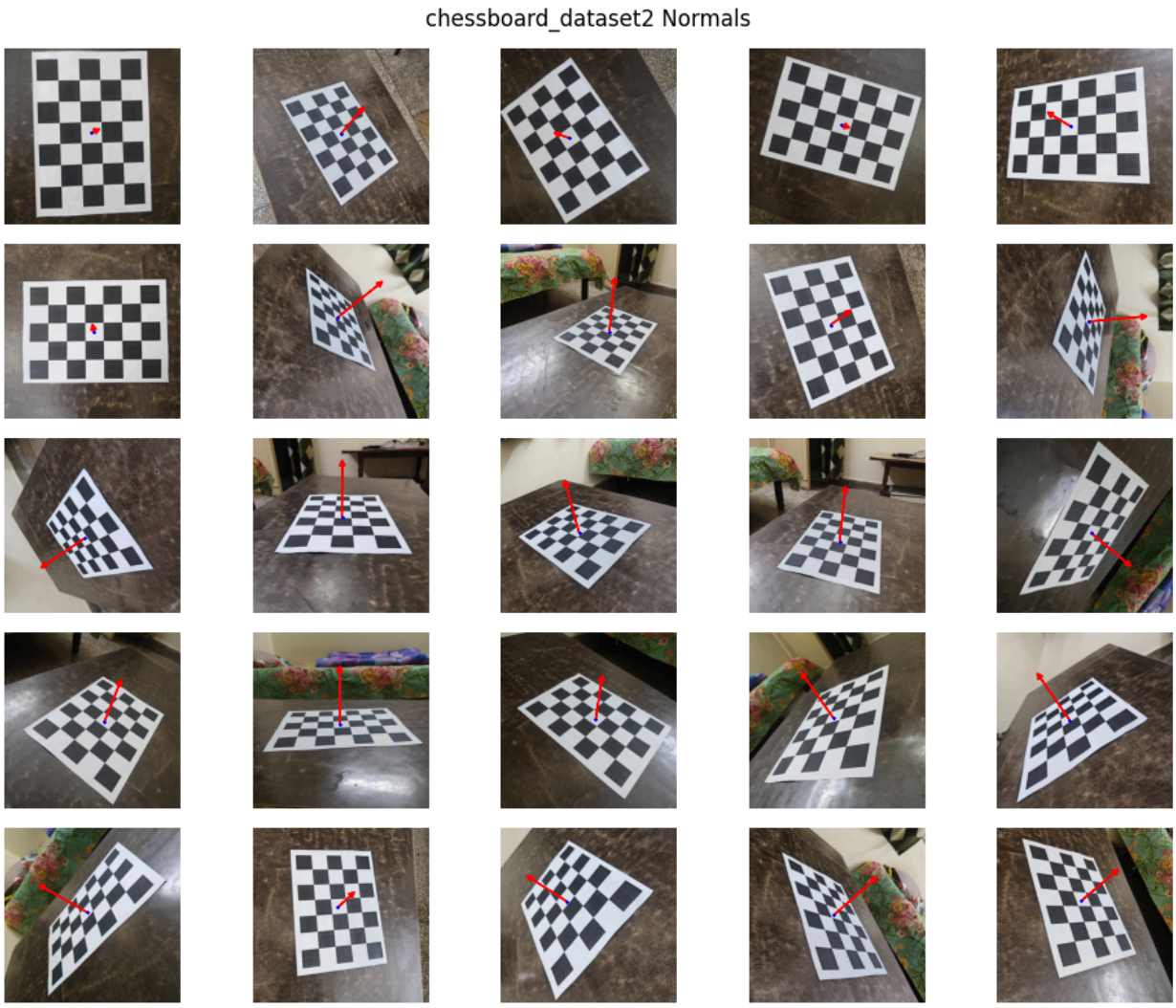
```
Normals for chessboard_dataset1:
01.jpeg: [-0.076538   -0.14604153  0.98631324]
02.jpeg: [-0.38592999 -0.3077939   0.86966716]
03.jpeg: [-0.27055961 -0.09590413  0.95791435]
04.jpeg: [-0.1653953   0.16256523  0.97273683]
05.jpeg: [-0.39753818  0.19242226  0.89718285]
06.jpeg: [-0.08367039 -0.12463089  0.98866901]
07.jpeg: [-0.10263416 -0.39925968  0.91107515]
08.jpeg: [ 0.18868403 -0.06357902  0.97997757]
09.jpeg: [0.45469862 0.40251032 0.79450274]
10.jpeg: [-0.02246819  0.12531503  0.99186255]
11.jpeg: [-0.39772957 -0.18595194  0.8984615 ]
12.jpeg: [-0.34642843 -0.52643739  0.77643481]
13.jpeg: [0.35757134 0.40152017 0.84316327]
14.jpeg: [ 0.04556911 -0.32149701  0.94581347]
15.jpeg: [-0.12740264 -0.50540119  0.85342733]
16.jpeg: [0.07218465 0.21920203 0.97300557]
17.jpeg: [-0.0964072   0.20029279  0.97498126]
18.jpeg: [-0.14471238  0.07615739  0.98653858]
19.jpeg: [-0.10791346 -0.08528305  0.99049558]
20.jpeg: [-0.13523341 -0.1051928   0.98521388]
21.jpeg: [-0.0966026  -0.27261542  0.95726108]
22.jpeg: [-0.35940042  0.0204804   0.93295867]
23.jpeg: [-0.16481583 -0.36095297  0.91790451]
24.jpeg: [ 0.03593448 -0.10987974  0.9932951 ]
25.jpeg: [-0.22305732  0.07012129  0.97228002]
```

chessboard_dataset1 Normals



```
Normals for chessboard_dataset2:
01.jpeg: [-0.06278457  0.02569651  0.99769624]
02.jpeg: [-0.34559879  0.40344948  0.84722488]
03.jpeg: [0.1772606  0.06259997 0.98217103]
04.jpeg: [-0.05963651 -0.02281741  0.99795934]
05.jpeg: [0.32333138 0.19720063 0.92550998]
06.jpeg: [0.01079596 0.04798995 0.99878947]
07.jpeg: [-0.71524066  0.57278574  0.40043388]
08.jpeg: [-0.08360447  0.83944486  0.53697544]
09.jpeg: [-0.27924782  0.19839354  0.93950022]
10.jpeg: [-0.93357093  0.08523494  0.34810964]
11.jpeg: [ 0.68213722 -0.46090917  0.56767205]
```

```
12.jpeg: [-0.00788593  0.89884224  0.43820137]
13.jpeg: [0.22458534 0.82935108 0.51160357]
14.jpeg: [-0.0933827   0.86497577  0.49304826]
15.jpeg: [-0.62256443 -0.52323433  0.58192729]
16.jpeg: [-0.25833456  0.64846575  0.71606943]
17.jpeg: [0.0066086  0.92901596 0.36998063]
18.jpeg: [-0.10341584  0.66774404  0.73717234]
19.jpeg: [0.5108402 0.7255826 0.4610555]
20.jpeg: [0.51079625 0.73741645 0.44193231]
21.jpeg: [0.75602538 0.43697875 0.48731427]
22.jpeg: [-0.22152599  0.2027451   0.9538452 ]
23.jpeg: [0.61041265 0.39287998 0.68778029]
24.jpeg: [-0.68626635  0.60352293  0.40595391]
25.jpeg: [-0.574008    0.48754023  0.65789007]
```



4. Panorama Generation

Overview

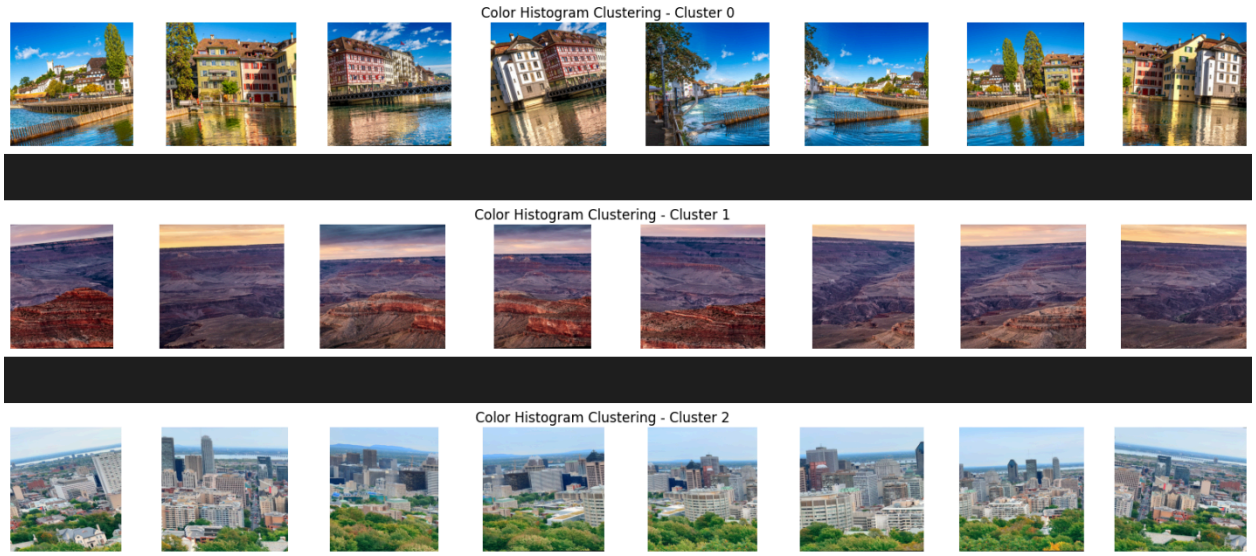
This project focuses on generating panoramic images from a set of unordered image data. The process includes clustering images into distinct sets using unsupervised techniques, detecting and matching features, estimating transformations, and finally stitching the images together to form complete panoramas.

Step 0: Dataset Preparation and Clustering

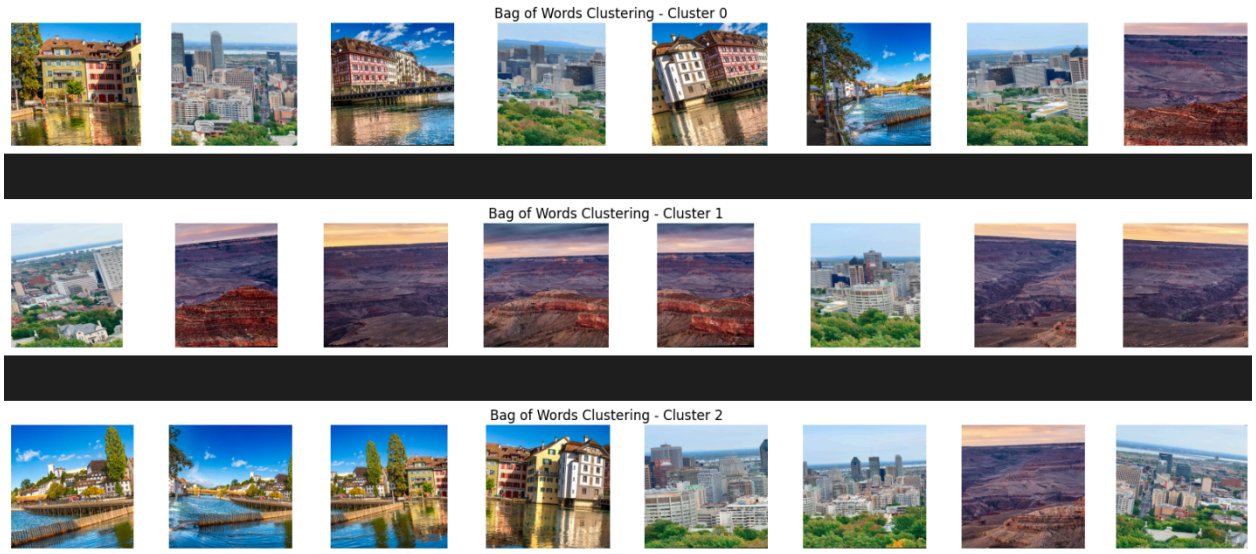
Objective: Separate the mixed dataset into three distinct image sets for panorama generation.

Clustering Approaches:

1. **Color Histograms**
    - Computed normalized histograms in RGB color space.
    - Flattened the histogram features for K-means clustering.



2. **Visual Bag of Words (BoW)**
    - Extracted SIFT features from all images.
    - Clustered descriptors using K-means to form a codebook.
    - Represented images using histogram of visual words.



Result:

After visual inspection:

- **Color HIstogram** provided better clustering results, more accurately grouping images by scene.

Step 1: Keypoint Detection using SIFT

Objective: Extract and visualize keypoints and descriptors from `image1` and `image2`.

- **Method:** SIFT
- **Library Used:** OpenCV (`cv2.SIFT_create()`)

Results:

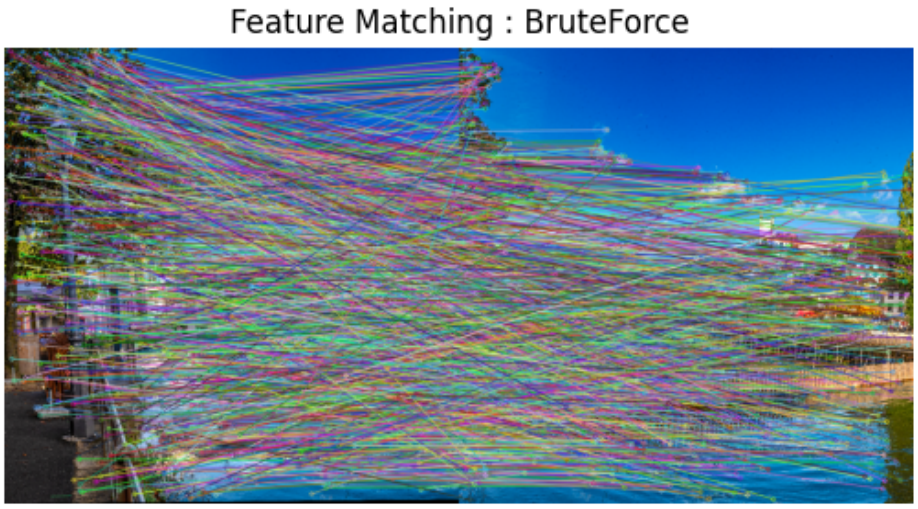
- Number of keypoints in image1: `5890`

- Number of keypoints in image2: `4510`

Step 2: Feature Matching

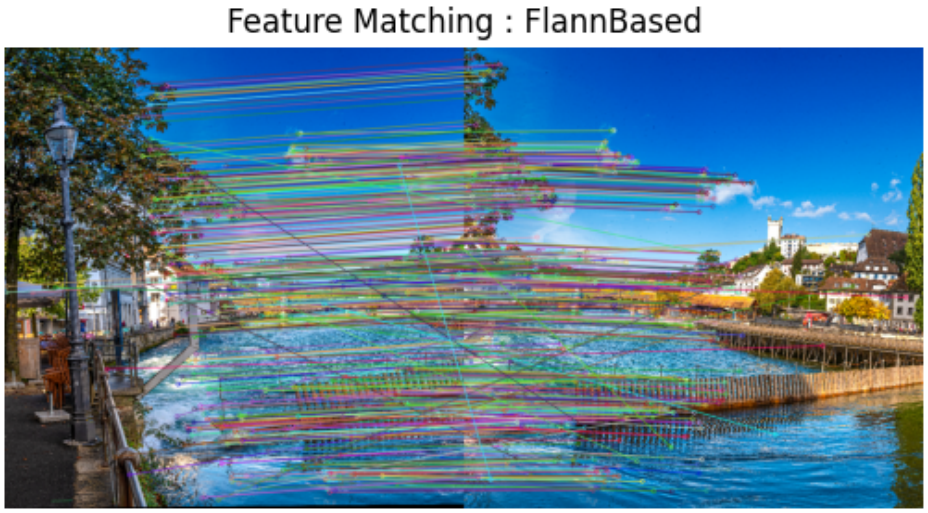
2a. BruteForce Matcher

- BruteForce is a simple algorithm that matches features by comparing all the descriptors of one image with all the descriptors of the other image.
- Matched features using L2 norm.
- Cross-checked matches for better accuracy.
- Number of matches using BruteForce: 1857



2b. FlannBased (Fast Library for Approximate NearestNeighbors) Matcher

- More efficient algorithm uses a hierarchical structure to speed up the matching process
- Number of matches using FlannBased: 686



Step 3: Homography Estimation using RANSAC

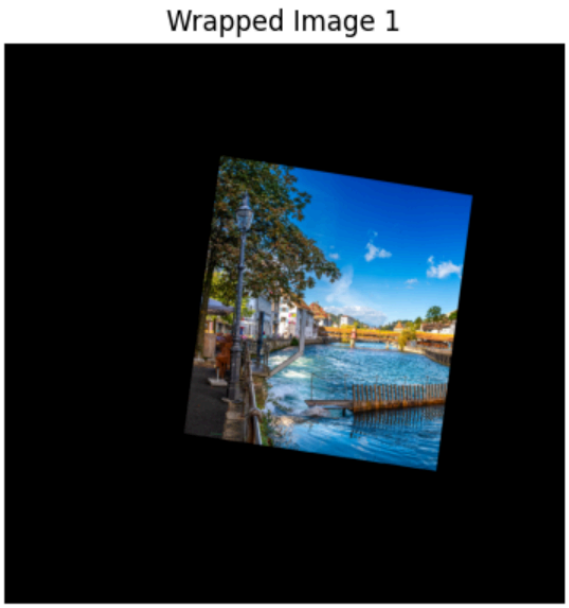
- Used FlannBased matched keypoints to compute the Homography matrix
- Applied `cv2.findHomography()` with RANSAC(Random Sample Consensus) to remove outliers.

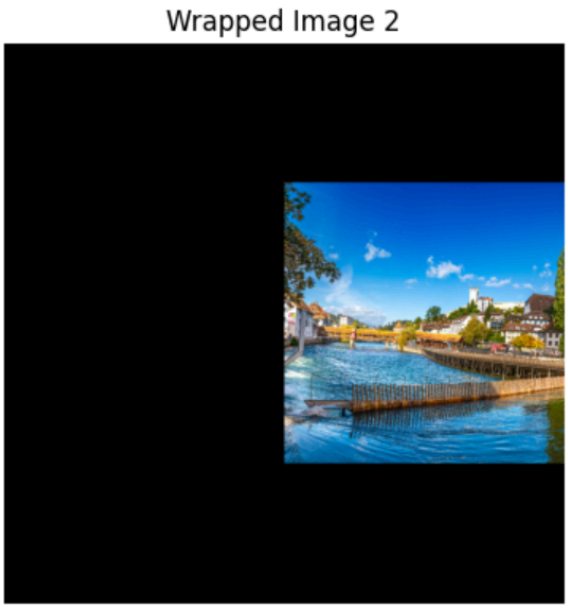
Output:

```
Homography matrix (FlannBased Matching) :
[[ 8.98903904e-01 -1.31281699e-01 -1.64422248e+02]
 [ 1.43239019e-01  9.90457145e-01 -6.94247988e+01]
 [-1.89979582e-07  1.06312086e-07  1.00000000e+00]]
Homography matrix saved to homography_matrix.csv
```

Step 4: Perspective Warping (5 points)

- Used the Homography matrix to warp the perspective of `image1` to align with `image2`





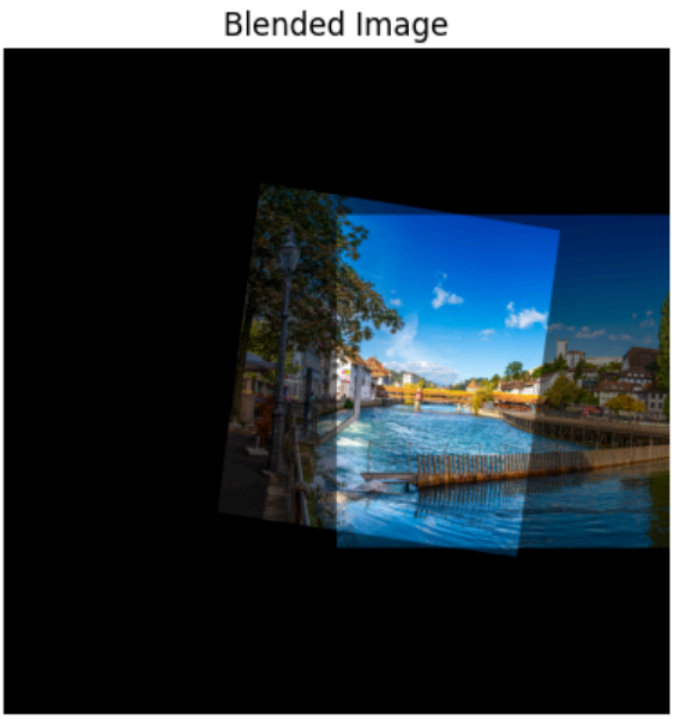
.

Step 5: Image Stitching

Uncropped and Unblended Panorama:

- Combined warped images into one canvas.
- No blending or cropping applied.

Image Blending:



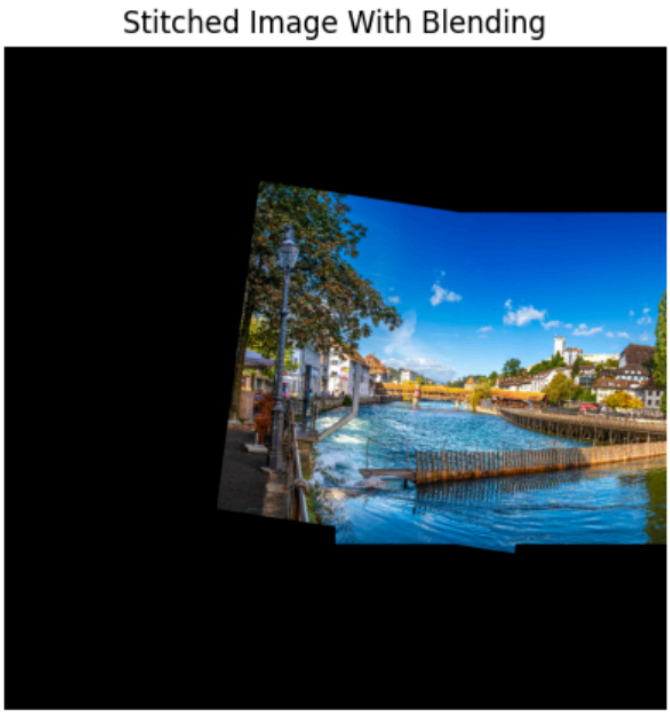


Image Cropping:.

Cropped

Cropped

Step 6: Multi-Stitching for All Image Sets

- Applied feature extraction, matching, homography estimation, warping, and stitching for all three clusters.

Final Panoramas:

Conclusion

- SIFT provided robust and reliable keypoint detection.
- FLANN provided fast and reasonably accurate matching.
- Homography estimation allowed for correct alignment.
- Final panoramas successfully demonstrate the complete workflow from clustering to stitching.

5. Point Cloud Registration

Overview

This project focuses on **3D point cloud registration** using the **Iterative Closest Point (ICP) algorithm** to estimate the **TurtleBot's trajectory**. The dataset consists of multiple sequentially recorded **.pcd (Point Cloud Data) files**, captured using a 3D LiDAR mounted on the TurtleBot. The primary goal is to align these point clouds and estimate the TurtleBot's movement through space.

Step 0: Dataset Preparation

- **Dataset Type**: Sequential 3D LiDAR point clouds.
- **Tool Used**: Open3D (`open3d`)
- **Number of Files**: `XX` point cloud files.

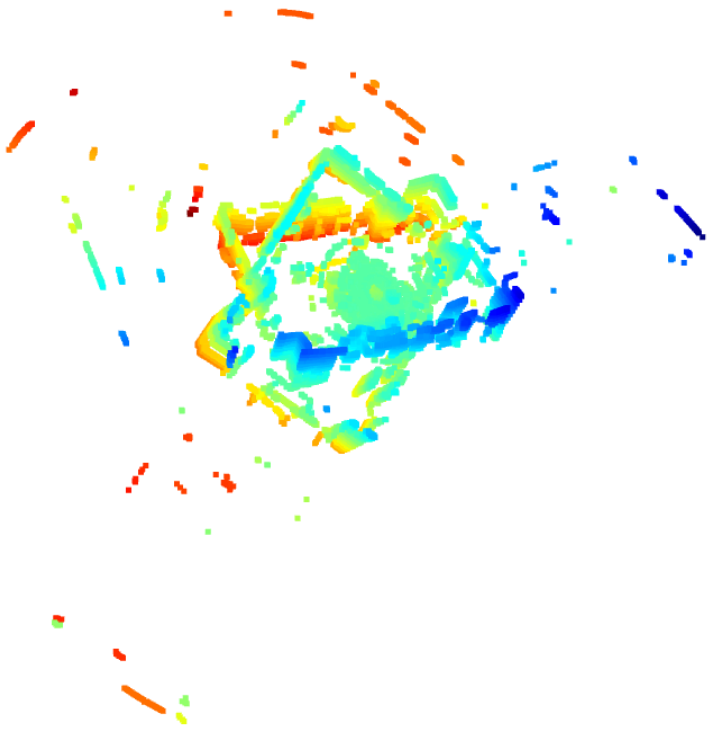
Step 1: Point-to-Point ICP on Two Consecutive Point Clouds

ICP Algorithm

- **Purpose**: Estimate a **transformation matrix (T-matrix)** that aligns two consecutive point clouds (0000.pcd and 0004.pcd).
- **Method Used**: **Point-to-Point ICP** (Iterative Closest Point).
  Software Used: Open3D's `registration_icp()`.
- **Initial Transformation Guess**: A random **valid orthonormal matrix** (not identical to the ground truth).

Results

```
Initial Transformation Matrix:
[[-0.78053528 -0.35709611 -0.51307606  0.67250961]
 [-0.61856367  0.55969998  0.55146615  0.16570671]
 [ 0.09024225  0.747809   -0.65775226  0.3504067 ]
 [ 0.          0.          0.          1.        ]]
ICP Learned Transformation Matrix:
[[-0.78588507 -0.57540184 -0.22648926  0.84310411]
 [-0.61679883  0.70329837  0.35345524  0.23980278]
 [-0.04408927  0.4174735  -0.90761887  0.00648642]
 [ 0.          0.          0.          1.        ]]
Initial Fitness: 0.206603497118296 37
Inlier RMSE: 0.10266657704530657
```



Step 2: FineTuning Hyperparameters

We performed multiple experiments with different hyperparameter settings to optimize performance. The following parameters were varied:

1. **Threshold values** (different distance thresholds for ICP).
2. **Initial Transformation Guess**:
    - **Random Orthonormal Matrix** (generated using `scipy.stats.ortho_group.rvs(3)`)
    - **RANSAC-based Initial Guess**
3. **Normal Estimation**: Used Open3D's `estimate_normals()` to get fpfh_features

Comparison Table

	Initial Transformation	Threshold	Fitness	Inlier RMSE
0	Random	0.05	0.009866	0.031929
1	Random	0.10	0.026424	0.062199
2	Random	0.20	0.287340	0.119322
3	Random	0.50	0.574241	0.223537
4	Random	1.00	0.743626	0.387675
5	RANSAC	0.05	0.984712	0.013113
6	RANSAC	0.10	0.997558	0.015169
7	RANSAC	0.20	0.999560	0.016227
8	RANSAC	0.50	0.999951	0.016983
9	RANSAC	1.00	1.000000	0.018004

Best Hyperparameters Selected with minimum RMSE:

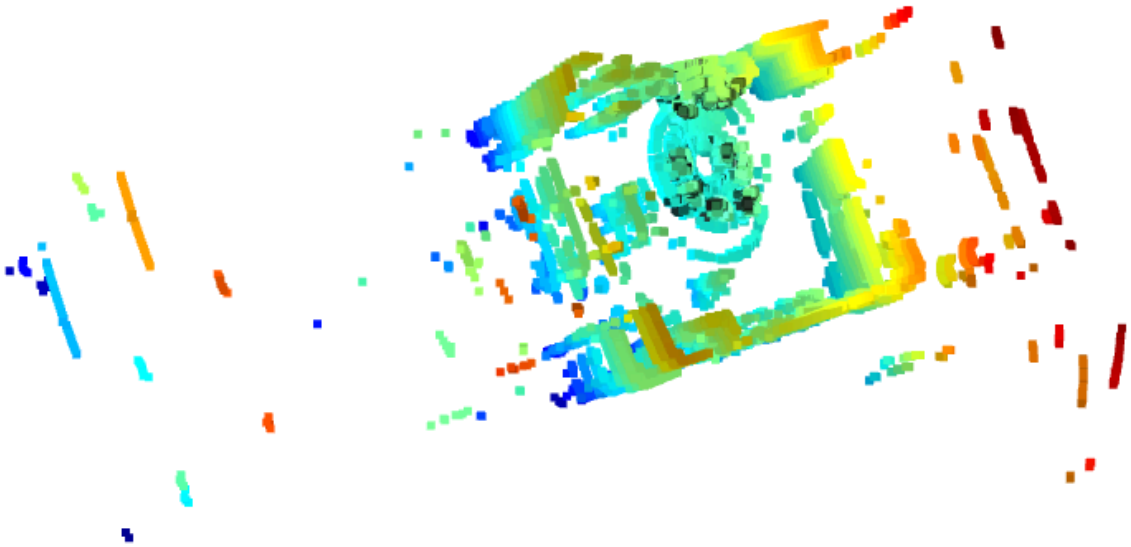
```
Best Hyperparameters:
Initial Transformation                                    RANSAC
Threshold                                                   0.05
Fitness                                                 0.984712
Inlier RMSE                                             0.013113
```

Step 3: Transforming the Source Point Cloud

- **Best Transformation Matrix**

```
Best Transformation Matrix:
[[-0.78627549 -0.5748775  -0.2264657   0.84352291]
 [-0.6163195   0.70371458  0.35346295  0.23888002]
 [-0.04383068  0.41749448 -0.90762174  0.00601855]
 [ 0.          0.          0.          1.        ]]
```

- **Visualization**: The transformed point cloud using the best T was plotted.



Observations:

- **Good alignment** was achieved with minimal drift.
- **Some minor misalignment** was still present in areas with fewer features.

Step 4: Full Registration and 3D Trajectory Estimation

ICP for All Point Clouds

- ICP was applied to all sequential point clouds to generate a global registered point cloud.

Estimated 3D TurtleBot Trajectory

- The trajectory was estimated by accumulating transformations from each pair of point clouds.