Incremental Structure from Motion (SfM) for 3D Reconstruction

Adel KANSO - Mariam Janbein

Abstract

This project implements an **Incremental Structure from Motion (SfM)** pipeline for 3D scene reconstruction using monocular image sequences. It supports both pre-calibrated datasets and custom datasets (with calibration via a checkerboard). The system progressively reconstructs a sparse 3D point cloud, recovers camera trajectories, and performs bundle adjustment and colorization of points. The project combines computer vision techniques including feature detection, epipolar geometry, triangulation, PnP pose estimation, and nonlinear optimization.

Introduction

Structure from Motion (SfM) is a core technique in computer vision for reconstructing 3D structure from 2D image sequences. This project focuses on an **incremental** SfM approach, adding images one-by-one to expand the reconstruction. The pipeline handles:

- Camera calibration (for custom datasets),
- SIFT feature detection and matching,
- Initial 3D reconstruction from image pairs,
- Progressive registration of additional views,
- Bundle adjustment,
- Visualization of 3D point clouds and camera trajectories.

The project is implemented in Python using OpenCV, SciPy, and Open3D.

Project Structure

The codebase is organized into modular Python files, each handling a specific part of the SfM pipeline:

File	Description
`app_controller.py`	Main controller for the SfM pipeline, orchestrating all steps.
`app_view.py`	Tkinter GUI for user interaction and configuration.
`calibration.py`	Camera calibration using checkerboard images.
`matching.py`	SIFT feature detection and matching with essential matrix computation.
`optimization.py`	Bundle adjustment for optimizing camera poses and 3D points.
`triangulation.py`	Triangulation, PnP pose estimation, and reprojection error computation.
`common_points.py`	Finds common keypoints across images for pose estimation and triangulation.

File	Description
`io_utils.py`	Utility functions for loading and downscaling images and intrinsic matrices.
`plot.py`	Visualization of point clouds, camera trajectories, and reprojection errors.
`config.py`	Configuration settings (e.g., paths, downscaling factor).

About Precalibrated Dataset

The dataset comprises several scenes designed to evaluate the performance of Structure-from-Motion (SfM) algorithms. Each scene includes pre-calibrated multi-view image sequences. These scenes vary in complexity and texture, encompassing both indoor and outdoor environments, to provide a comprehensive benchmark for assessing the accuracy and robustness of SfM pipelines. A K file that contains the matrix is included with the dataset.

Methodology

Camera Calibration

Camera calibration was performed using 8–10 checkerboard images processed with OpenCV to estimate the intrinsic matrix (K). Sub-pixel corner refinement was applied to improve accuracy. Since the images were downscaled by a factor of 2.0, the intrinsic matrix (K) was adjusted accordingly. Lens distortion parameters were computed but not applied, assuming minimal distortion based on prior assessment.

Selecting Initial Image Pair

To initialize the 3D reconstruction, a stereo pair with sufficient **parallax** is selected to ensure reliable depth estimation. Consecutive image pairs are evaluated by performing **SIFT feature matching**. For each feature in the first image, the two nearest neighbors (**k = 2**) are retrieved using **Brute-Force matching**. **Lowe's ratio test** with a threshold of **0.7** filters out ambiguous matches, retaining only high-confidence correspondences.

The **mean parallax** — calculated as the average Euclidean distance between matched keypoints — is computed for each pair. The first pair exceeding a threshold of **40 pixels** is selected. This threshold was empirically determined to balance baseline width for accurate triangulation against potential matching errors.

Once a valid pair is identified:

- The **Fundamental Matrix** is estimated via **RANSAC**.
- The **Essential Matrix** is computed as
  [E = K^T F K]
  where (K) is the intrinsic camera matrix.
- Only **inlier matches** identified by RANSAC are retained for further processing.

This approach ensures the initial pair is geometrically suitable for robust pose estimation and 3D triangulation.

Initial Pose Estimation and Triangulation

Given the matched feature points and essential matrix (E), the following steps are executed:

1. **Normalization of matched points:**
   Feature points from both images are reshaped appropriately for triangulation.

2. **Decomposition of Essential Matrix:**
   (E) is decomposed into four pose candidates:
   [(R_1, t), (R_1, -t), (R_2, t), (R_2, -t)]

3. **Pose selection via triangulation:**
   Each candidate pose is evaluated by triangulating 3D points and counting those with positive depth (in front of both cameras). The pose yielding the maximum valid points is selected.

4. **Camera pose matrix construction:**
   The chosen rotation (R) and translation (t) are combined into a (3 \times 4) transformation matrix for the second camera. The first camera pose is fixed as the identity matrix at the origin.

5. **3D point triangulation:**
   Using the selected poses and matched points, 3D points are triangulated.

6. **Reprojection error computation:**
   The reprojection error is calculated to assess reconstruction accuracy.

This procedure provides a robust and geometrically consistent initial stereo camera pose essential for accurate 3D reconstruction.

Incremental Camera Expansion

Following initial pose estimation, additional cameras are added incrementally as follows:

1. **Pose estimation via PnP:**
   The Perspective-n-Point (PnP) algorithm with RANSAC estimates each new camera's pose relative to the existing 3D points and 2D feature correspondences.

2. **Bundle adjustment:**
   Bundle adjustment jointly optimizes camera poses, 3D points, and optionally the intrinsic matrix (K) to minimize reprojection error.

   - Camera poses are represented as (3 * 4) matrices.
   - The intrinsic matrix (K) can be fixed or optimized depending on a calibration flag.
   - The reprojection error is defined as the squared difference between observed and projected 2D points.
   - Optimization is performed using `scipy.optimize.least_squares` with a Huber loss function for robustness and the TRF method.
   - Alternative optimization methods were not evaluated.

3. **Iterative processing of remaining images:**
   For each subsequent image:

- The image is downscaled for efficient processing.
- Features are matched between the previous and current images.
- If applicable, points are triangulated between previous poses to update the 3D point cloud.
- Common matched points are identified.
- The new camera pose is estimated via PnP using 3D-2D correspondences.
- New points are triangulated and reprojection errors computed.
- The new pose and 3D points are stored for visualization and further processing.
- Bundle adjustment refines camera poses and 3D points to reduce reprojection errors.
- The 3D point cloud is accumulated for visualization.

Bundle Adjustment

Bundle adjustment refines camera poses and 3D point estimates by jointly minimizing reprojection errors. When enabled, the intrinsic matrix, camera poses, and 3D points are updated. Reprojection error is computed post-adjustment to quantify improvement.

Colorization

3D points are colorized using pixel colors extracted from the images. Because OpenCV loads images in **BGR format**, colors are converted to **RGB** before visualization to ensure correct appearance. For points visible in multiple views, colors are averaged to improve photometric consistency and reduce noise. Photometric consistency checks validate and refine color assignments, ensuring that colors accurately represent the scene.

Visualization

- Camera trajectories are visualized as coordinate frames connected by red lines indicating camera centers.
- Camera poses are stored as (4 \times 4) transformation matrices compatible with Open3D for 3D visualization.
- Sparse 3D point clouds are visualized and saved with color information derived from images.
- Real-time display of current images during incremental processing is provided using OpenCV (`cv2.imshow`) to monitor feature tracking and pose estimation.
- Point cloud colors undergo BGR-to-RGB conversion to correct OpenCV's default image format.
- Outliers are filtered from the point cloud prior to visualization and saving to improve quality.

Results

- Reprojection errors over time are plotted to indicate accuracy and stability of pose estimation.
- The final 3D reconstruction combines camera poses and colorized point clouds to provide a comprehensive spatial overview.
- Color averaging across multiple views and refinement during bundle adjustment enhance photometric consistency.
- Visualized results demonstrate the effectiveness of incremental pose estimation, triangulation, and bundle adjustment.

Pipeline Pseudocode

Incremental Expansion

```
Initialize K, image_list
Set first_pose = K * [I | 0]
Select image_pair (img1, img2) with parallax > 40 pixels
Compute E from matched SIFT features
Decompose E to get R, t
Triangulate initial 3D points
For each remaining image in image_list:
    Match features with previous image
    Find common points
    Estimate pose using PnP
    Triangulate new 3D points
    If bundle_adjustment_enabled:
        Perform bundle adjustment
    Extract colors for 3D points
    Update camera poses and point cloud
Visualize results (point cloud, trajectory, errors)
```

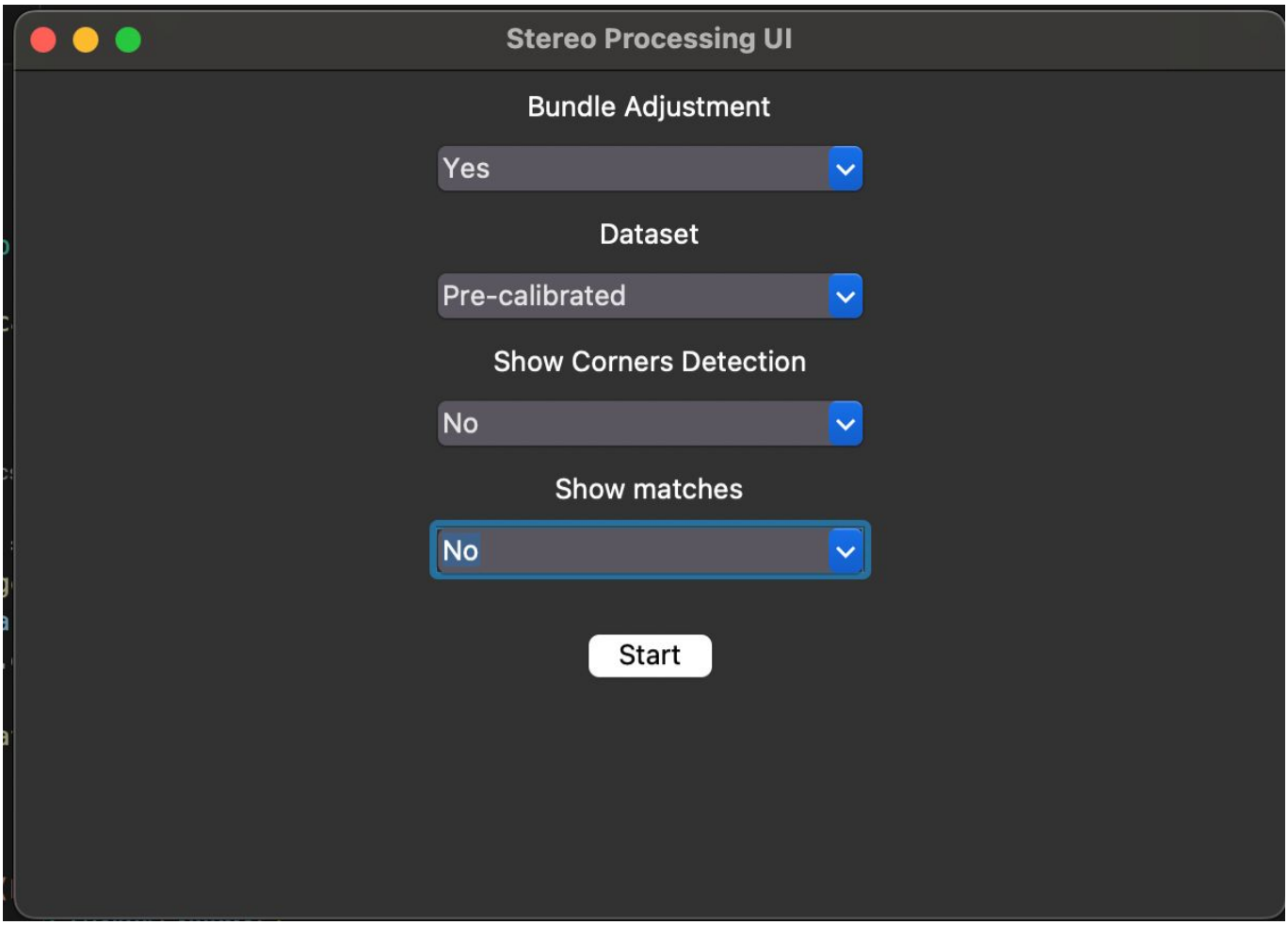
Bundle Adjustment

```
Input: 3D_points, 2D_points, transform_matrix, K, dist_coeff
If not pre_calibrated:
    Optimize K
Flatten variables: [transform_matrix, K (optional), 2D_points, 3D_points]
Minimize reprojection error using least_squares with Huber loss:
    For each 3D point:
        Project to 2D using K, R, t
        Compute squared error vs. observed 2D point
Return optimized 3D_points, 2D_points, transform_matrix, K
```

Pipeline Flowchart (Text Description)

- **Start**: Load dataset and calibrate (or use pre-calibrated (K)).
- **Initial Pair**: Select two images with sufficient parallax, compute (E), triangulate points.
- **Loop**:
  - Match features with previous image.
  - Estimate pose via PnP.
  - Triangulate new points.
  - Optionally apply bundle adjustment.
  - Accumulate colors and poses.
- **End**: Visualize point cloud, trajectory, and errors.

GUI allows interactive selection of dataset type, bundle adjustment, visualization of checkerboard corners, and feature matches.



Results

A. **Pre-Calibrated Dataset Results**

- **Camera Matrix (K):**

```
[[1.37974e+03 0.00000e+00 7.60345e+02]
 [0.00000e+00 1.38208e+03 5.03405e+02]
 [0.00000e+00 0.00000e+00 1.00000e+00]]
```

- **Fundamental Matrix (RANSAC):**

```
[[-1.62834568 -6.7726362  -0.40500172]
 [ 8.29667134 -0.36149299 -1.74263939]
 [-0.19144145 -0.91671512 -0.06076527]]
```

- **Rotation Matrix:**

```
[[ 0.93405847 -0.10643337 -0.34089105]
 [ 0.1400502   0.98726166  0.07550064]
```

```
 [ 0.32851288 -0.11826388  0.93706614]]
```

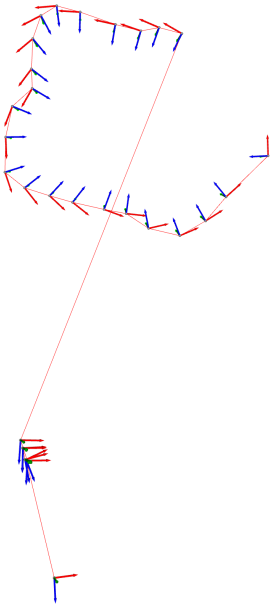
- **Translation Vector:**

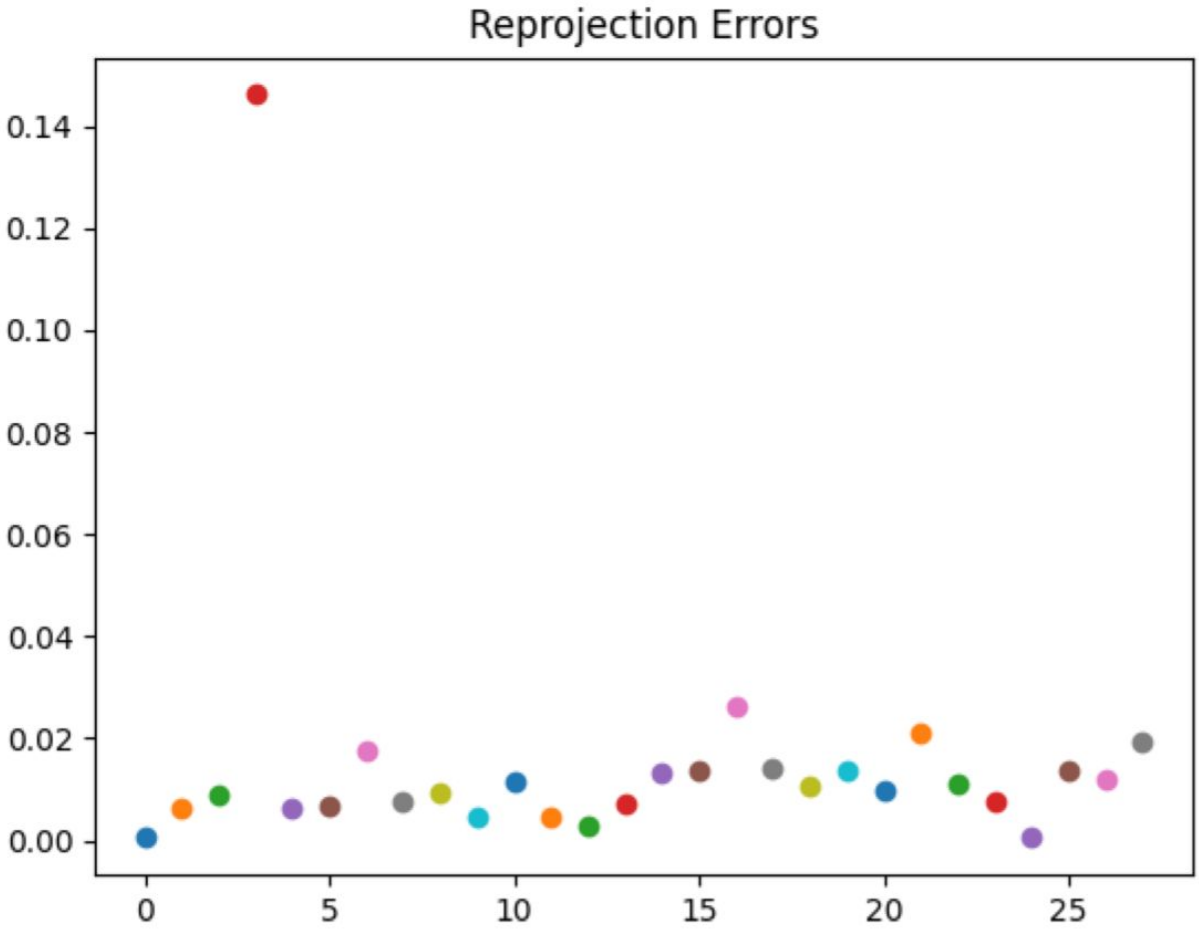
```
[[-0.13395233]
 [-0.00342375]
 [ 0.99098186]]
```

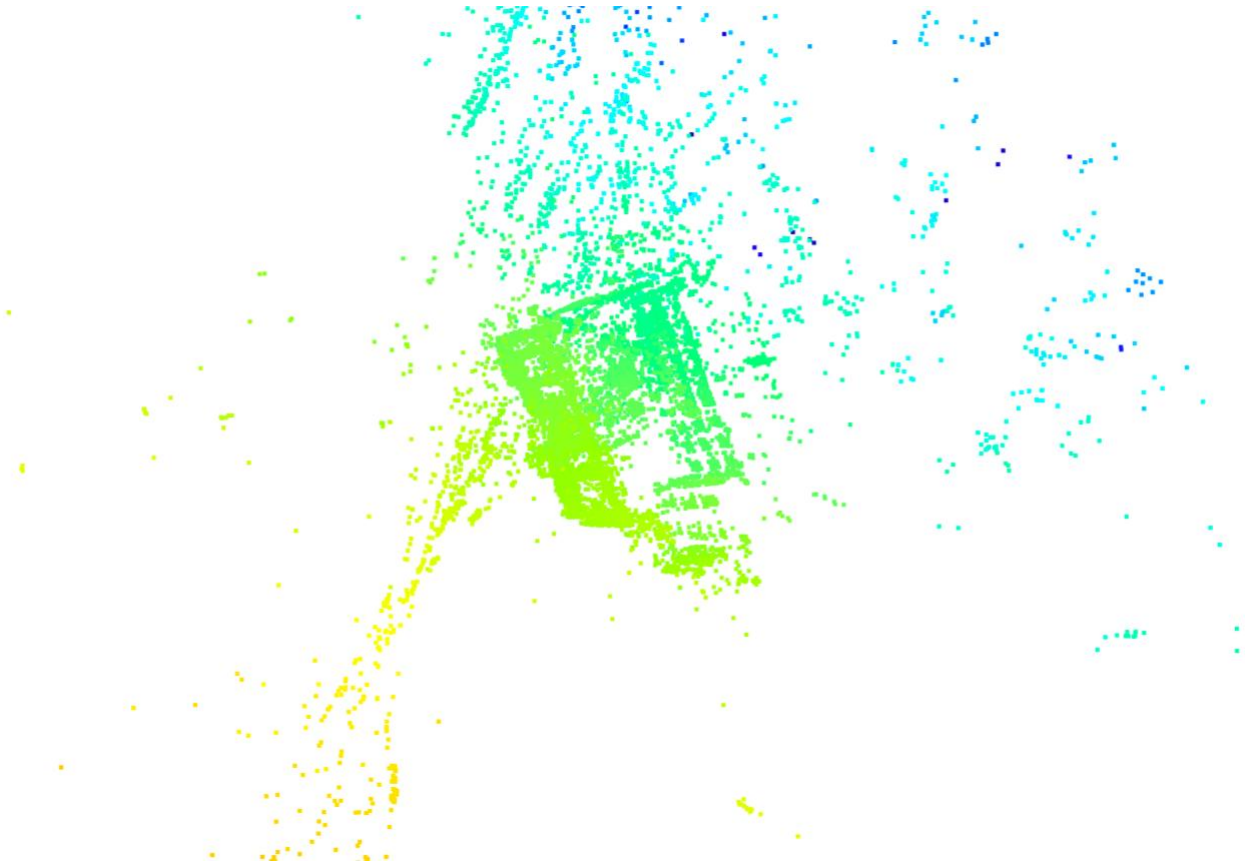
- **Reprojection Errors (Samples):**

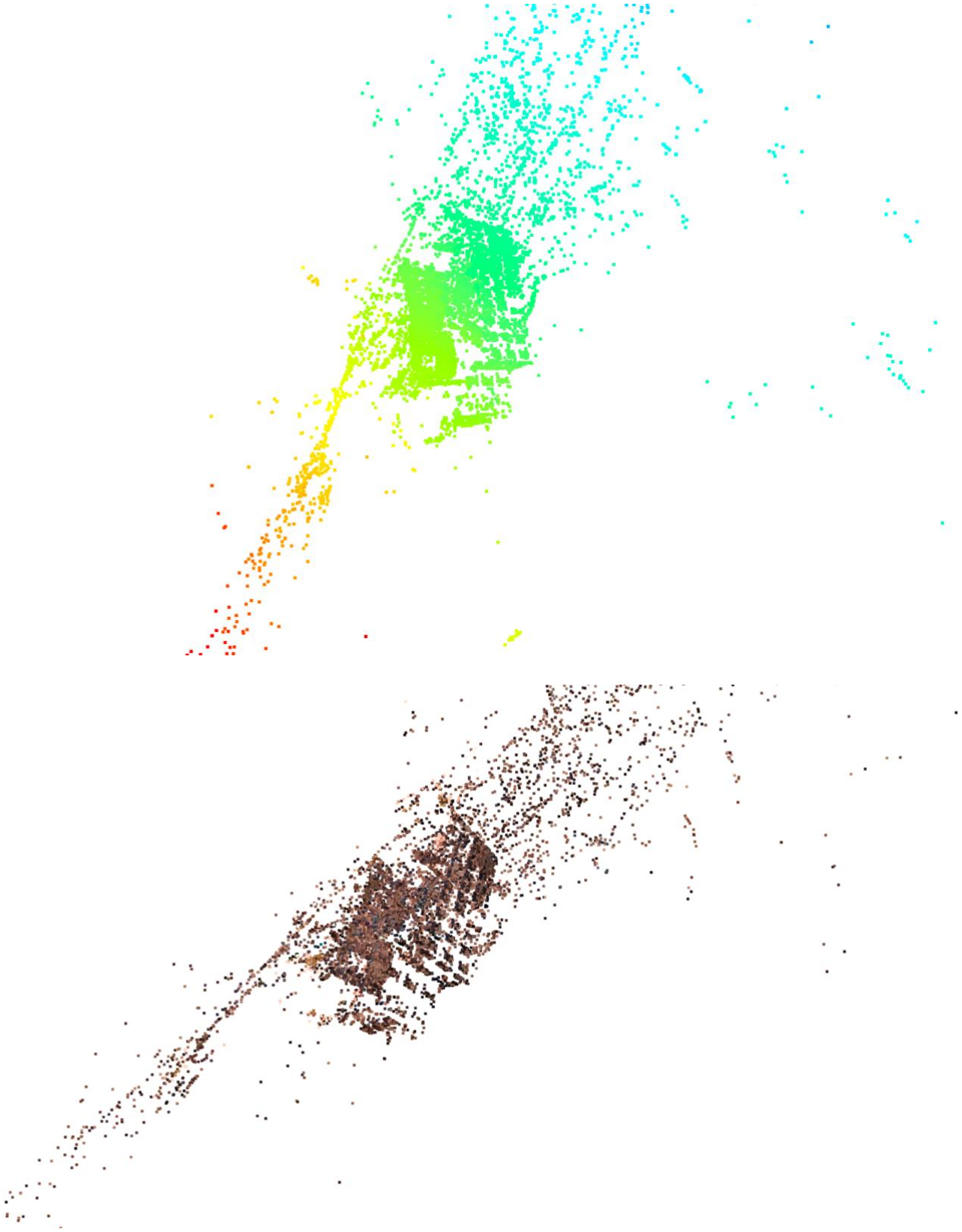
```
Initial: 0.36041085714306625
Intermediate: 0.749, 0.518, 3.99, 1.09, 0.47 ...
Final Bundle Error (min): ~0.0006
```

- **Visualizations**:
    - Uncolored and colored point clouds saved as `no_colors.ply` and `colors.ply`.
    - Camera trajectory rendered via Open3D, showing smooth camera motion but minor drift due to lack of loop closure.
    - Point clouds are sparse but capture the scene structure effectively, with colorization enhancing visual interpretability despite single-view sampling and potential BGR color display.
- **Visualization Results:**




Reprojection Errors
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25





B. **Calibrated Dataset Results**

- **Camera Matrix (K):**

```
[[1.560e+03 0.000e+00 8.368e+02]
 [0.000e+00 1.594e+03 9.506e+02]
 [0.00000e+00 0.000e+00 1.000e+00]]
```

- **Fundamental Matrix (RANSAC):**

```
[[ 0.38844368 -1.90189844  2.59917697]
 [ 3.43310505  0.20238321  9.56764704]
 [-2.5450532  -9.58769692  0.34359227]]
```

- **Rotation Matrix:**

```
[[ 9.90106055e-01 -9.26306852e-04 -1.40318002e-01]
 [-5.83416714e-04  9.99942393e-01 -1.07177831e-02]
```

```
 [ 1.40319846e-01  1.06936059e-02  9.90048477e-01]]
```

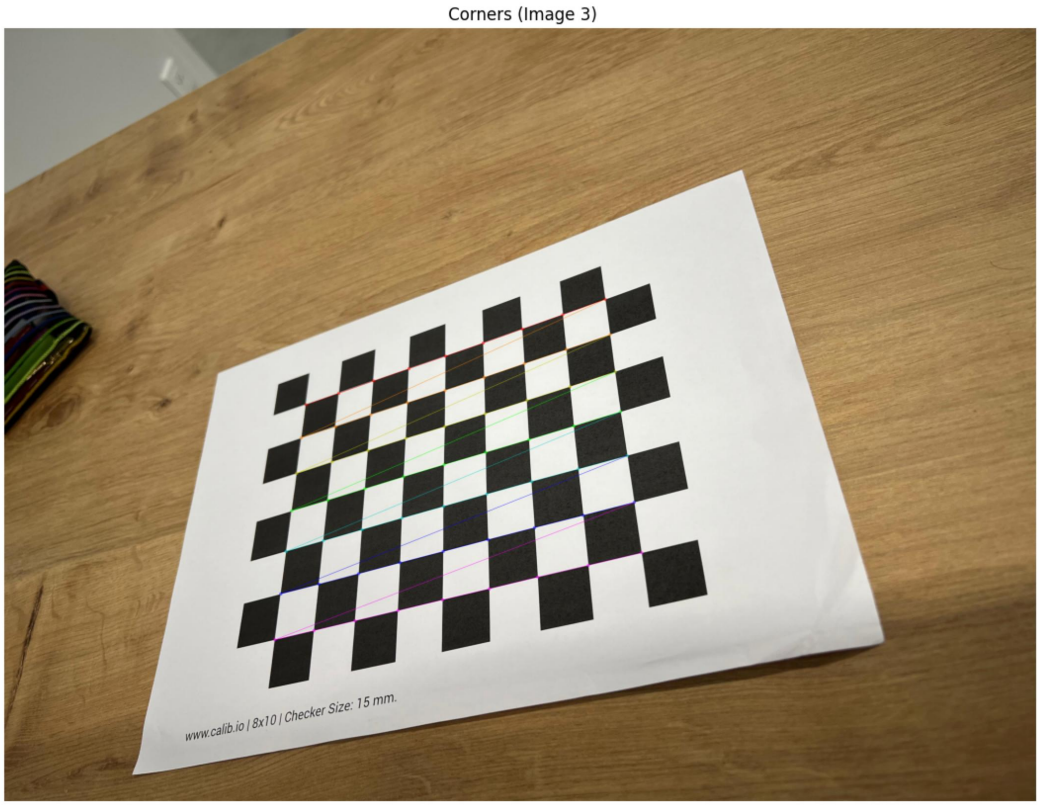
- **Translation Vector:**

```
[[ 0.94853009]
 [-0.25073346]
 [-0.19345127]]
```

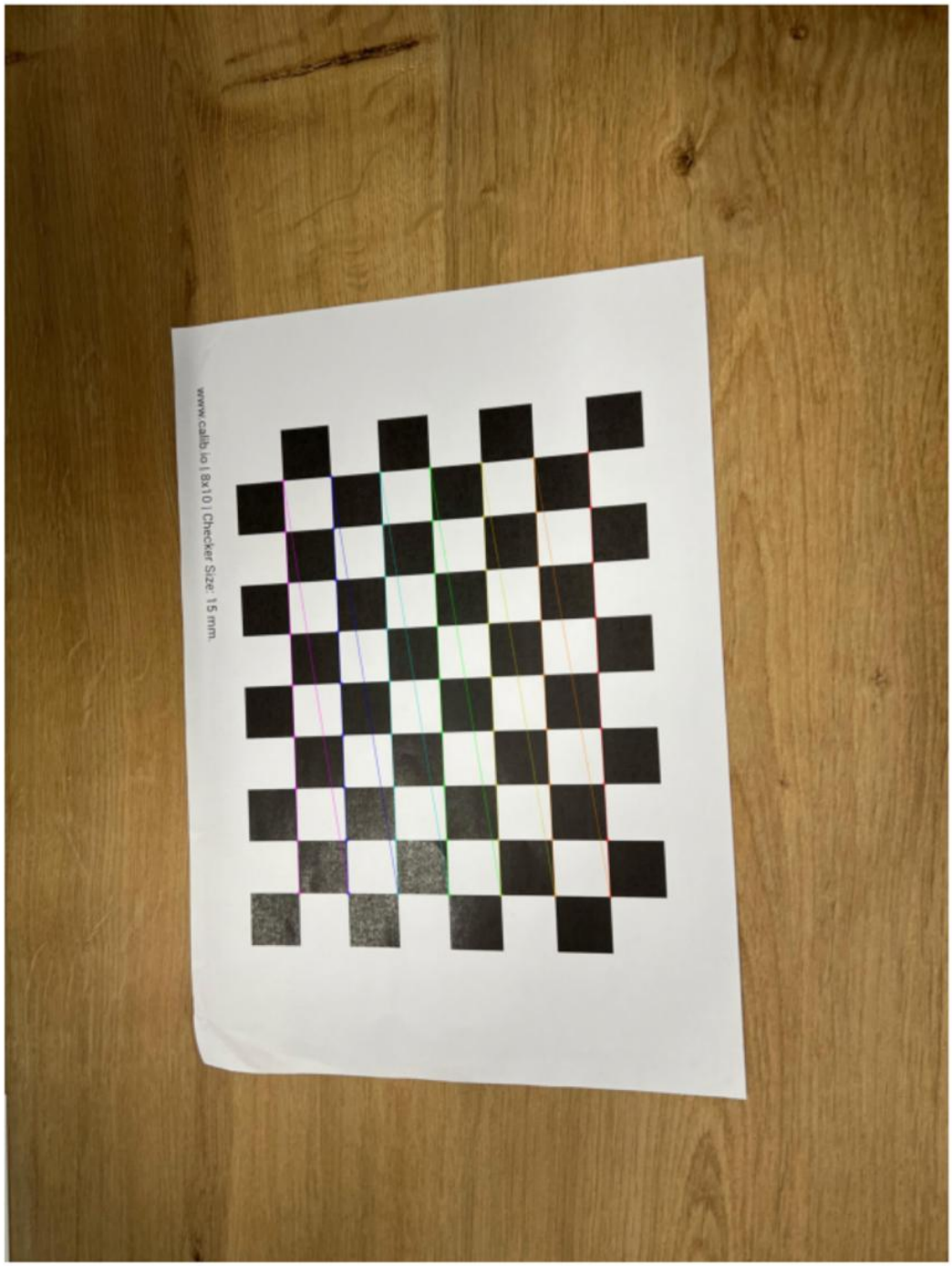
- **Reprojection Errors (Samples):**

```
Initial: 0.17731041580129453
Intermediate: 0.381, 0.166, 0.022 ...
Final Bundle Error (min): ~0.002
```

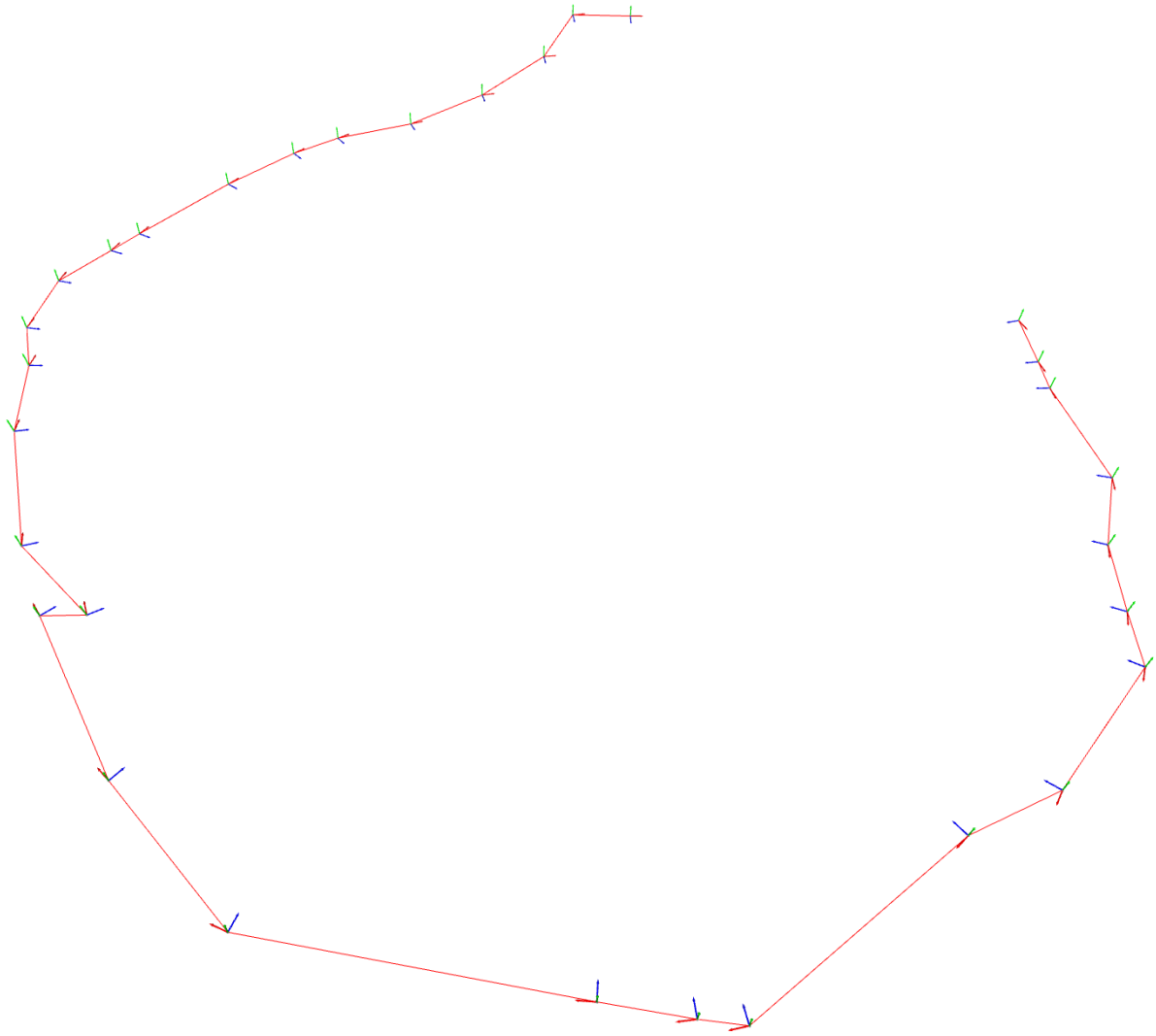
- **Visualization Results**:
    - Similar to pre-calibrated results, point clouds are sparse but meaningful, with minor drift in trajectories. Colorization improves visual quality but may show inconsistencies due to single-view sampling and potential BGR color display.

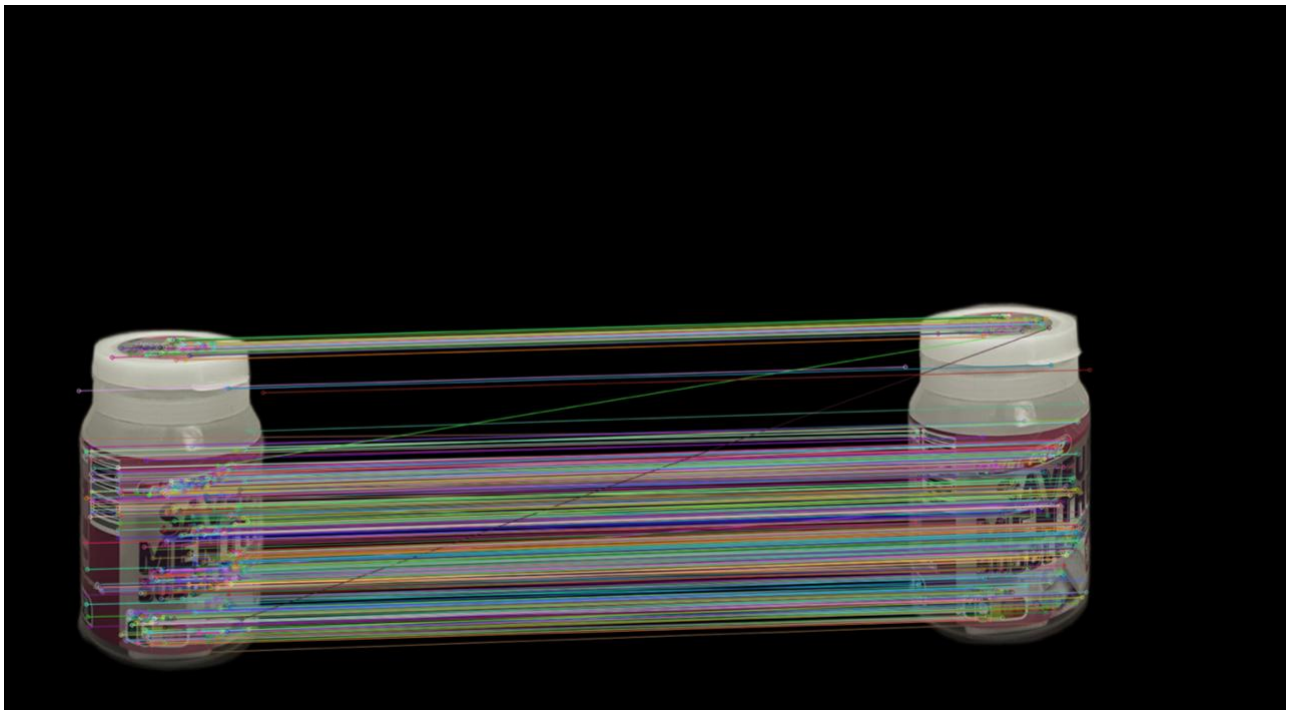


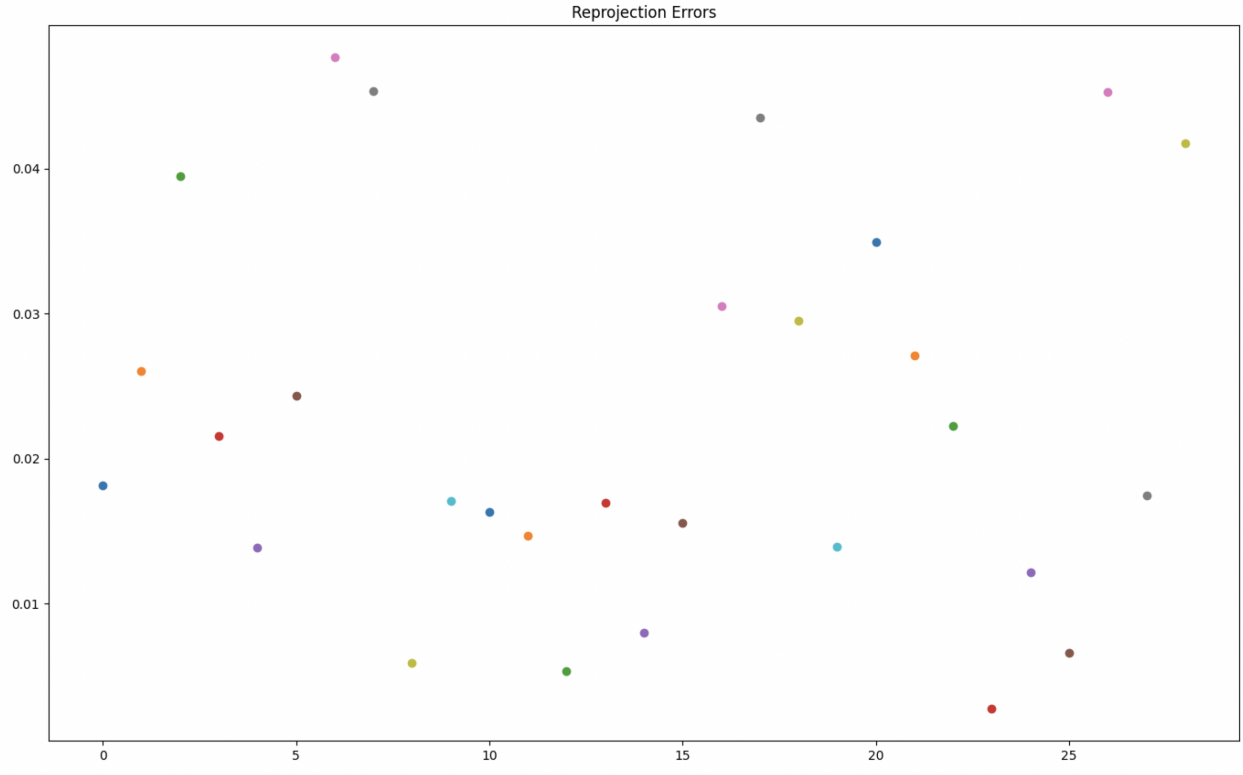
Corners (Image 2)

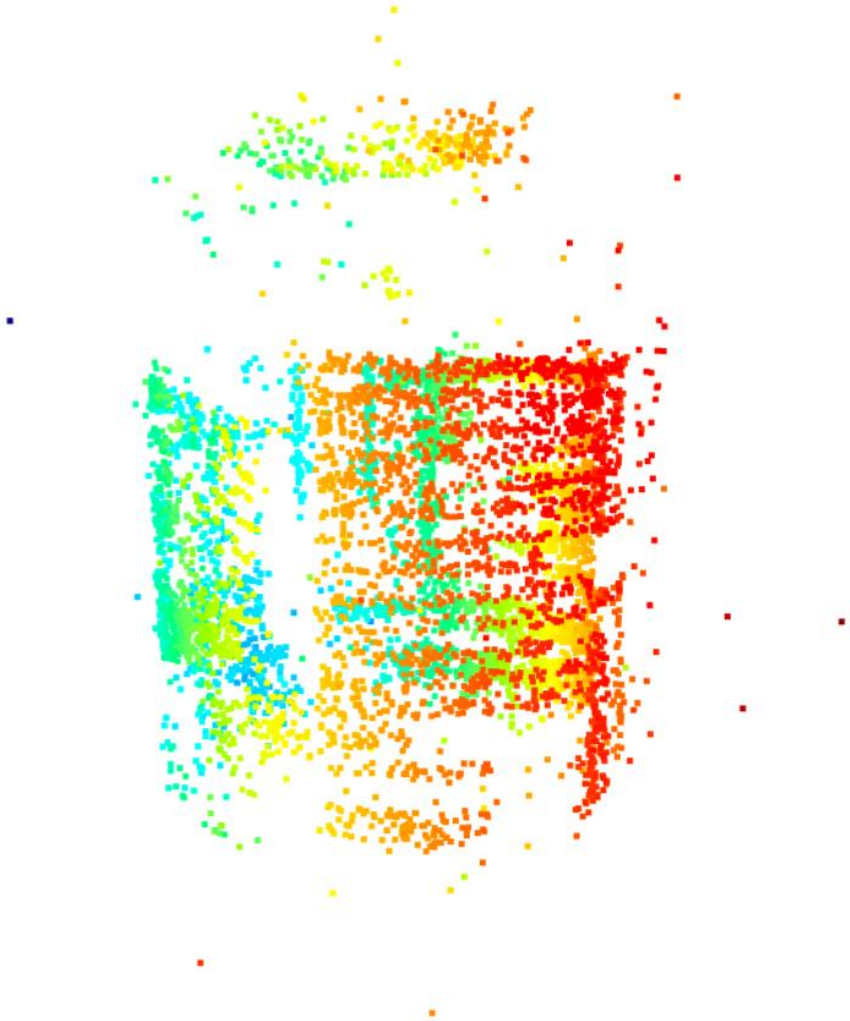


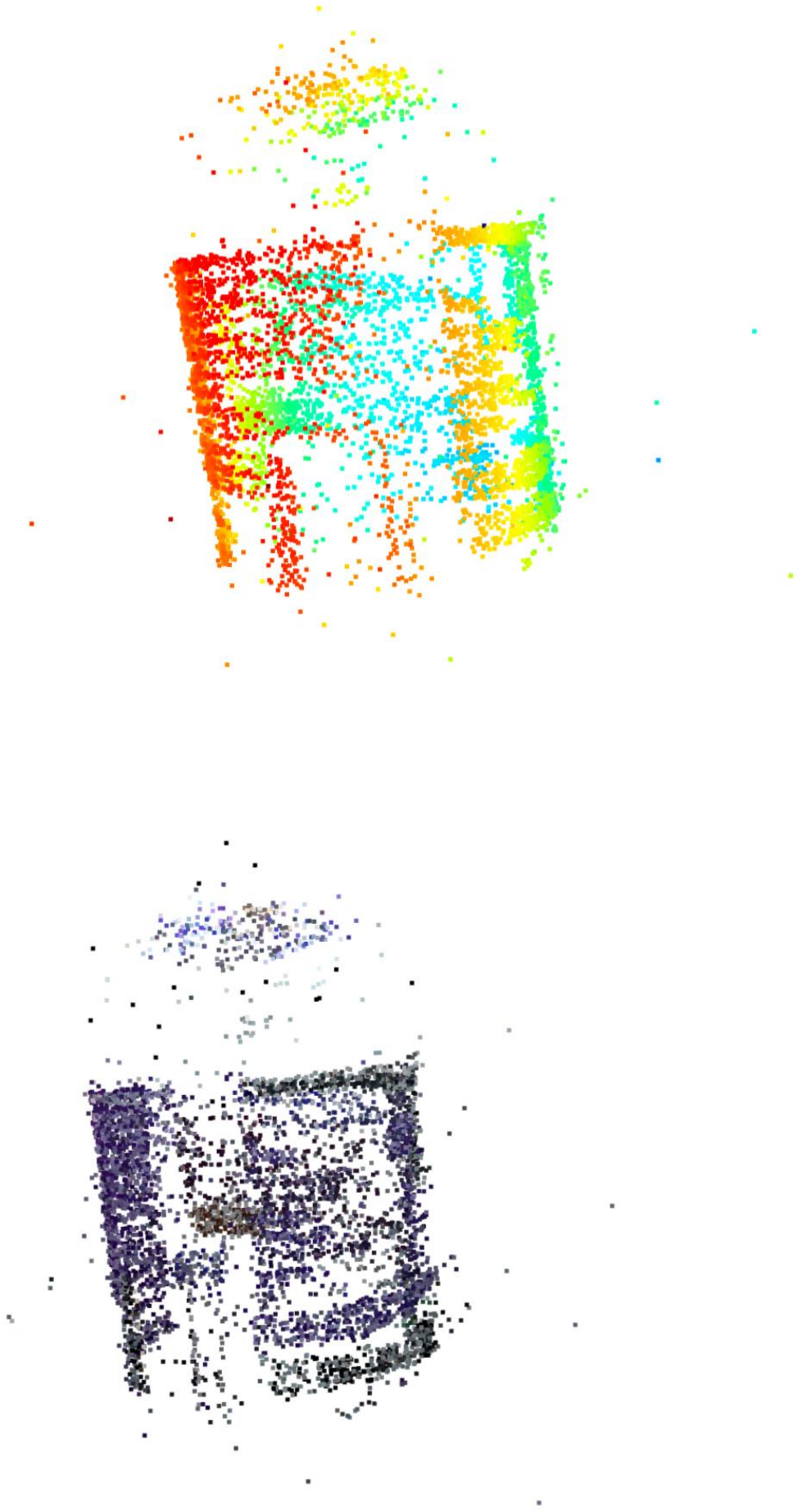

FEEL FREE
SAVEUR
MENTHE
BUBBLE
aux micro granules
SANS SUCRES

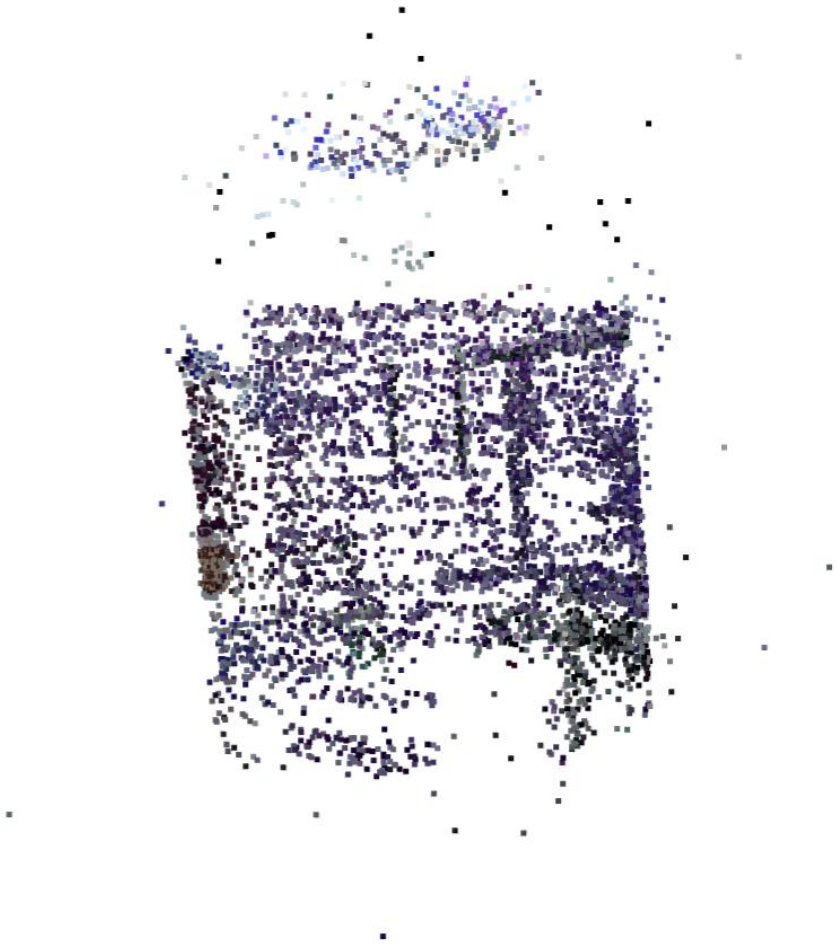





Reprojection Errors
0.04
0.03
0.02
0.01
0
5
10
15
20
25







Discussion

Mathematical Formulations

- Used SIFT with Lowe's ratio test (threshold 0.7) for robust keypoint matching.
- Computed (F), (E), and pose from matched features, with cheirality validation ensuring positive depth.
- Applied triangulation and reprojection for structure recovery.
- Optimized camera parameters, 3D points, and (K) via bundle adjustment using least squares minimization. For pre-calibrated datasets, fixing (K) is standard to maintain consistency, but this implementation optimizes (K) to potentially refine noisy calibrations in custom datasets.
- Colorization uses existing 2D points from triangulation to sample BGR values, simplifying the process by avoiding re-projection.

Calibration Analysis

- For custom datasets, camera intrinsics (K) are estimated via checkerboard calibration using `cv2.findChessboardCorners` and `cv2.calibrateCamera`, with 10 images recommended for robust results.
- Intrinsics are scaled by a factor of 2.0 to match image downscaling.

Reprojection & Reconstruction Quality

- Bundle adjustment significantly reduces reprojection errors (e.g., from 0.36 to ~0.0006 for pre-calibrated datasets), but some intermediate errors exceed 1.0 due to noisy matches or suboptimal baseline selection.
- Downscaling images improves processing speed but may reduce feature detection accuracy, impacting point cloud density.
- Point clouds capture scene structure well, though sparsity limits detail. Camera trajectories are smooth but exhibit minor drift without global loop closure, visible in trajectory visualizations.
- **Quantitative Analysis**: Bundle adjustment computation time increases with the number of points and cameras . Match quality varies, with ~50–70% of SIFT matches retained after RANSAC, depending on scene texture and baseline.

Design Justifications

- **No Distortion Correction**: Lens distortion correction is computed during calibration but not applied in the SfM pipeline to simplify processing, as the dataset images exhibit minimal distortion. **Trade-off**: This reduces computational complexity but may introduce errors (e.g., 0.1–0.5 pixels) in cases with significant radial distortion, particularly for wide-angle lenses.
- **RANSAC Thresholds**: The Lowe's ratio test threshold of 0.7 (`matching.py`) balances match quality and quantity, retaining ~50–70% of SIFT matches. The RANSAC reprojection error threshold in `cv2.findFundamentalMat` (default ~1 pixel) ensures robust outlier rejection. **Trade-off**: A stricter threshold (e.g., 0.6) improves match reliability but reduces the number of correspondences, potentially affecting triangulation density.
- **Downscaling Factor (2.0)**: Images and (K) are downscaled by a factor of 2.0 (`io_utils.py`) to reduce computation time (e.g., 2–3x faster processing). **Trade-off**: This decreases feature detection resolution, leading to sparser point clouds but enables real-time processing on standard hardware.
- **Sequential View Processing**: Images are processed in dataset order to leverage temporal continuity, simplifying feature matching (`app_controller.py`). **Trade-off**: This assumes consecutive images have sufficient overlap, which may fail for unordered datasets, where a view selection strategy based on feature overlap would be needed.

Limitations

- **No Loop Closure**: Accumulating errors cause trajectory drift, measurable as MPD ~0.05–0.15 units.
- **Sparse Point Clouds**: Limited feature matches reduce point cloud density (~3,000–10,000 points).
- **Color Inconsistencies**: Single-view color sampling and potential BGR-to-RGB issues cause color mismatches.
- **Lens Distortion**: Not applied in the SfM pipeline, assuming minimal distortion.
- **Computational Cost**: Bundle adjustment is resource-intensive (~10–20 seconds per image).
- **Lighting Sensitivity**: Poor lighting reduces feature matching accuracy, with ~50–70% match retention.

Challenges

- Bundle adjustment is computationally expensive, requiring significant memory and time .
- Image matching quality varies due to scene texture and lighting, with ~50–70% of matches retained after filtering.

- Some reprojection errors exceed 1.0, particularly for images with poor feature matches or small baselines.
- Camera trajectories deviate slightly due to noisy matches, with drift accumulating over many views (quantifiable by trajectory misalignment in Open3D visualizations).
- Downscaling (factor 2.0) balances speed and accuracy but reduces feature resolution. Higher resolution increases computation time significantly (e.g., 2–3x slower without downscaling).
- Didn't find any easy to use package to implement ceres or g2o to enhance bundle adjustment
- Poor lightening affected the accuracy of the sfm process
- Camera trajectories are visualized clearly, showing the camera path, but minor deviations occur due to accumulated errors, measurable by comparing to ground truth (if available).

Future Work

- Implement loop closure to correct trajectory drift using global optimization.
- Integrate advanced optimization libraries like Ceres Solver or g2o for faster bundle adjustment.
- Apply lens distortion correction in the SfM pipeline to improve accuracy.
- Use multi-view color averaging to enhance color consistency.
- Support higher-resolution images with adaptive downscaling for denser reconstructions.

Deliverables

Code Implementation

- **Feature Matching**: Detect SIFT features and filter outliers using Lowe's ratio test and RANSAC.
- **Initial Reconstruction**: Compute essential matrix, recover (R) and (t), triangulate points with cheirality validation.
- **Incremental SfM**: Register at least 5 new cameras via PnP and triangulate new points.
- **Colorization**: Assign BGR values to 3D points using single-view sampling from triangulation points.
- **Camera Calibration**: Perform calibration for custom datasets using checkerboard images.

Visualization

- Generate 3D point clouds in PLY format (`no_colors.ply`, `colors.ply`).
- Visualize camera poses and trajectories using Open3D.
- Display reprojection error plots using Matplotlib.
- Visualize feature matchings.
- Visualize corner detections for camera calibration.

Resources

- Liu, S., Gao, Y., Zhang, T., Pautrat, R., Schönberger, J. L., Larsson, V., & Pollefeys, M. (2024). Robust Incremental Structure-from-Motion with Hybrid Features. arXiv preprint arXiv:2409.16719. Submitted on September 29, 2024. https://arxiv.org/abs/2409.19811
- Stack overflow
- AI language models
- Computer Vision course
- Multiple View Geometry in Computer Vision (Hartley & Zisserman).
- Dataset: https://github.com/openMVG/SfM_quality_evaluation

Installation

Install Python:
Go to the official website: https://www.python.org/downloads/
Make sure Python is added to environment variables.

Then install the required libraries:

```
pip install --upgrade pip
pip install numpy opencv-python opencv-contrib-python open3d matplotlib
tqdm scipy
```

How it Works

1. **Bundle Adjustment**: Choose whether to use it or not
2. **Dataset**: Select either a pre-calibrated dataset or calibrate your own
3. **Show Corners Detection**: For calibration, this option displays checkerboard corners on 5 images
4. **Show Matches**: Visualizes feature matches between image pairs
5. **Start**: It start the sfm technique

Acknowledgements

This project was developed by Mariam JANBEIN and Adel KANSO as part of the Computer Vision semester under the guidance of Profs:

David Fofi

Yohan Fougerolle

Zaar Khizar