Structure-from-Motion in 2D
SSY098 Project III

Allegra Gasparetto

***Abstract*—This project implements a Structure-from-Motion pipeline in 2D to reconstruct a sparse representation of a planar scene from a set of images taken from different viewpoints and with different zoom and rotation conditions. Structure-from-Motion is used for reconstructing 3D geometry from multiple 2D images by analyzing the motion of features across views. In this project, the focus is on the 2D variant of Structure-from-Motion for planar scenes.**

In the suggested pipeline, SIFT features are extracted from each image and matched between image pairs. For each pair, a homography matrix is estimated that maps points from one image to the other. The point coordinates are first normalized. These normalized points are then used in the Direct Linear Transform algorithm to estimate the homography. A RANSAC-based method is applied to handle incorrect matches by repeatedly sampling minimal subsets of correspondences, computing candidate homographies, and selecting the one with the highest number of inliers. The global coordinate system is initialized using the image pair with the highest number of inliers, and the remaining images are added iteratively by composing homographies. Once all images are registered in the global coordinate system, keypoints are projected to generate a sparse 2D map of the scene. Finally, the accuracy of the reconstruction is evaluated by comparing the estimated sparse map with one generated using ground truth homographies, both visually and by computing the average residual error.

***Index Terms*—Structure-from-Motion, homography estimation, RANSAC, SIFT, 2D reconstruction**

I. Method

The goal of this project is to estimate a sparse 2D reconstruction of a planar scene using a Structure-from-Motion pipeline. The method consists of the following main steps: SIFT feature extraction, SIFT feature matching and homography estimation, global coordinate system initialization, iterative integration of remaining images, sparse map generation, and evaluation with ground truth data.

A. SIFT feature extraction

The process begins with converting each input image to grayscale using the `read_as_grayscale` function. This function apply `cv2.imread` with the `cv2.IMREAD_GRAYSCALE` flag to load the image as a 2D array, reducing the image to a single intensity channel instead of three (RGB), while preserving the essential structural details necessary for feature detection.

Next, feature extraction is performed by the `extract_sift_features` function, which applies the Scale-Invariant Feature Transform (SIFT) algorithm via `cv2.SIFT.create`. It detects local keypoints and computes their corresponding descriptors, vectors that delineate the local appearance around each keypoint, making them robust to changes in scale, rotation, and illumination.

Finally, the function `sift_features` iterates over a list of input images, applies SIFT feature extraction to each, and returns two lists: one containing the keypoints and another containing the descriptors for each image.

B. SIFT feature matching and homography estimation

Once features are extracted from two images, the next step is to establish correspondences between their keypoints. The function `match_descriptors` compares the SIFT descriptors from the source and target images to identify pairs of matching keypoints, using `BFMatcher` with L2 norm to find the two nearest neighbor matches for each descriptor in the source image. To ensure good matches and filter out ambiguous or incorrect matches, Lowe's ratio test is applied, ensuring a match is accepted only if the distance of the closest match is significantly smaller than that of the second-closest match. The function returns both the set of good matches and the complete set of matches.

Using the matching pairs, the 2D coordinates of the corresponding keypoints from the source and the target images are obtained with `extract_keypoint_matches`. This function iterates over the good matches and extracts the `.pt` attribute from each `cv2.KeyPoint`, converting the results into NumPy arrays, where each column represents the (x, y) coordinates of a matched keypoint. These correspondences serve as the input to estimate the homography matrix, which is the projective transformation that maps points from one image to the other.

To improve numerical stability in homography estimation, the matched 2D points are first normalized using the `normalize` function. This involves translating the points so their centroid is at the origin and scaling them such that the average distance from the origin becomes $\sqrt{2}$. The matrix `T` is the similarity transformation used to normalize 2D homogeneous points before estimating the homography. It is defined as:

$$T = \begin{bmatrix} s & 0 & -s \cdot c_x \\ 0 & s & -s \cdot c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where (c_x, c_y) are the coordinates of the centroid of the points, and $s = \frac{\sqrt{2}}{d}$ is the scaling factor, with d being the average distance of the points from the centroid. The `estimate_homography_dlt` function implements the Direct Linear Transform (DLT) algorithm for homography

estimation. It starts by normalizing both sets of corresponding points. Then, it constructs a linear system based on the geometric constraints that a homography imposes between two views. Given corresponding points in homogeneous coordinates,

$$\mathbf{P} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad \tilde{\mathbf{P}} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix},$$

and a homography matrix $H \in \mathbb{R}^{3\times3}$, the projective relationship is defined by:

$$\tilde{\mathbf{P}} \sim H\mathbf{P}$$

To eliminate the unknown scale, the cross product formulation is used:

$$\tilde{\mathbf{P}} \times (H\mathbf{P}) = \mathbf{0}$$

Expanding this yields:

$$\begin{bmatrix} y'(h_3^T\mathbf{P}) - w'(h_2^T\mathbf{P}) \\ w'(h_1^T\mathbf{P}) - x'(h_3^T\mathbf{P}) \\ x'(h_2^T\mathbf{P}) - y'(h_1^T\mathbf{P}) \end{bmatrix} = \mathbf{0}$$

This expression can be rearranged into a matrix-vector product:

$$\begin{bmatrix} \mathbf{0}^T & -w'\mathbf{P}^T & y'\mathbf{P}^T \\ w'\mathbf{P}^T & \mathbf{0}^T & -x'\mathbf{P}^T \\ -y'\mathbf{P}^T & x'\mathbf{P}^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix} = \mathbf{0}$$

Due to linear dependence among the three rows, only the second and third are typically retained to construct the constraint system. The following pair of equations is derived from the second and third rows of the expanded cross product:

$$\begin{aligned} -(xh_{21} + yh_{22} + h_{23}) + y'(xh_{31} + yh_{32} + wh_{33}) = 0 \\ (xh_{11} + yh_{12} + h_{13}) - x'(xh_{31} + yh_{32} + wh_{33}) = 0 \end{aligned}$$

This system is expressed in matrix form as:

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -w & y'x & y'y & y'w \\ x & y & w & 0 & 0 & 0 & -x'x & -x'y & -x'w \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \mathbf{0}$$

By stacking the above two equations for each point correspondence, a matrix $A \in \mathbb{R}^{2N\times9}$ is constructed:

$$A\mathbf{h} = \mathbf{0}$$

To find a non-trivial solution for $\mathbf{h}$, Singular Value Decomposition (SVD) is applied, yielding the solution corresponding to the smallest singular value. The resulting homography matrix is then denormalized back to the original coordinate system by inverting the earlier normalization transforms.

However, point correspondences may contain mismatches (outliers), which can distort the homography estimate. To robustly filter out these outliers, the `ransac_fit_homography` function employs the RANSAC (RANdom SAmple Consensus) algorithm. It repeatedly selects a minimal sets of four random point correspondences to estimate a candidate homography with DLT. For each candidate, the reprojection error for all correspondences is computed using `compute_reprojection_error`, which determines the Euclidean distance between the projected source points and their corresponding target points. Matches with errors below a defined threshold are considered as inliers. Finally, RANSAC selects the homography with the highest number of inliers.

The `pair_homographies` function automates this entire pipeline across all image pairs. For each image pair, it performs descriptor matching (`match_descriptors`), extracts the corresponding 2D keypoints (`extract_keypoint_matches`), and then estimates the homography (`ransac_fit_homography`). It stores: the best homography matrix for each pair, the corresponding inlier matches, and the number of inliers found.

C. Global coordinate system initialization

To build a global coordinate system, the image pair with the largest number of inlier correspondences is selected as the starting point. The first image in this pair is assigned the identity homography, $\mathbf{H}_{\text{ref}} = \mathbf{I}_3$, defining the global coordinate system's origin.

The second image is aligned relative to the first by applying the inverse of the estimated pairwise homography. The global homography for the second image is computed as the inverse of the homography mapping points from image i to image j, $\mathbf{H}_{ij}$. In this way, both images share a consistent coordinate system, serving as a stable basis for integrating the remaining images.

The global coordinate system initialization procedure is implemented within `compute_global_homographies`.

D. Iterative integration of remaining images

After initializing the global coordinate system with the first two images, the `compute_global_homographies` function incorporates the remaining images into the global map in a sequential manner.

At each iteration, the algorithm selects the image that has the largest number of homography inliers with images already integrated into the global coordinate system. The global homography for this new image is computed by composing the known global homography of a connected image with the inverse of the pairwise homography between them. Let I denote the image to be added and J the corresponding image already in the global coordinate system. Let $\mathbf{H}_{J,I}$ be the homography from I to J. The global homography $\mathbf{H}_I$ is then computed by composing the known global homography of image J with $\mathbf{H}_{J,I}$:

$$\mathbf{H}_I = \mathbf{H}_J \cdot \mathbf{H}_{J,I}$$

The homography $\mathbf{H}_I$ is normalized such that its bottom-right entry is 1 to maintain scale consistency. This integration process continues until all images are referenced in the global coordinate system.

E. Sparse map generation

Once all images are registered in the global coordinate system, the detected keypoints from each image are projected in it. This is achieved by applying the corresponding global homography to each keypoint location.

The function `keypoints_to_global` takes the list of keypoints and the dictionary of global homographies as input. For each image, it extracts the 2D keypoint coordinates, converts them into homogeneous coordinates, and applies the associated global homography. The result is then dehomogenized to obtain the corresponding Cartesian coordinates in the global system. The transformed keypoints from all images are concatenated to form a sparse 2D reconstruction of the planar scene.

To visualize this reconstruction, the function `sparse_map` is used. It plots the keypoints transformed in the global coordinate system from each image onto a common axis. This provides a visual representation of the reconstructed planar scene, showing how keypoints from different views align in the global coordinate system.

F. Evaluation with ground truth

To evaluate the accuracy of the estimated global homographies, both qualitative and quantitative comparisons are performed using ground truth data. The function `load_ground_truth` reads ground truth homography matrices. These homographies represent the transformation from each image to a common reference system, image 1. To compare all homographies within the coordinate system of a chosen reference image, each ground truth homography is adjusted by multiplying it with the inverse of the homography of that reference image. This effectively re-centers all transformations so they are expressed relative to the selected global coordinate system.

Using the re-centered ground truth homographies, the function `ground_truth_sparse_map` visualizes a sparse reconstruction of the scene. Each image's keypoints are projected into the global coordinate system using the ground truth homographies. This allows for a side-by-side comparison with the sparse map obtained from estimated homographies, highlighting alignment consistency and global map structure.

To quantitatively assess reconstruction accuracy, the function `compute_average_residual` calculates the average residual reprojection error. For each image, the keypoints are projected into the global coordinate system using both the estimated and ground truth homographies. The Euclidean distance between the resulting point sets is computed for all keypoints, and the mean residual across all images provides a scalar measure of geometric consistency between the estimated and true transformations.

II. Experimental Evaluation

The performance of the 2D Structure-from-Motion pipeline was evaluated on four datasets: `graf`, `wall`, `bark`, and `boat`. For each dataset, SIFT-based feature matching was followed by homography estimation using the RANSAC algorithm. The evaluation was based on both visual consistency of the reconstructed sparse maps and the quantitative average residual reprojection error.

Two key RANSAC parameters were tested: the maximum number of RANSAC iterations and the inlier threshold. Multiple values were evaluated empirically to identify the configuration yielding the most visually accurate reconstructions and the lowest residual errors. In particular, RANSAC iterations were tested from 10 up to 10000, while the inlier threshold values studied were between 0.01 and 3.

During experimentation, changes in the number of RANSAC iterations showed minimal impact on the quality of the reconstruction. Even with a small number of iterations, as 10, the algorithm often produced satisfactory homographies. Incrementing the number of iterations, above 1000, resulted in slightly lower residuals, but it also increased the computation times.

In contrast, the inlier threshold had the most significant impact on reconstruction quality. It directly affected the stability of the resulting homographies, the average residual error, and the choice of the reference image, which is automatically selected as the one yielding the highest number of inliers across all pairwise estimations. The inlier threshold effect on reconstruction quality varied between datasets.

A. `graf` dataset

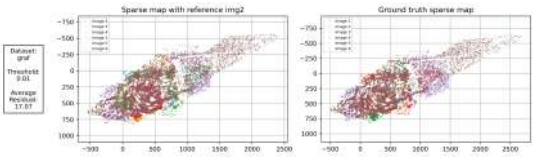
In the case of the `graf` dataset, the choice of reference image changed depending on the threshold value. As illustrated in Figure 1, for very low thresholds, such as 0.01, image 2 was selected as the reference, while slightly higher thresholds around 0.02 occasionally led to images 1, 3, or 4 being chosen. When image 5 was the reference, as illustrated in Figure 2, the average residual errors were consistently lower, around 1.5 pixels, indicating better reconstruction quality. Threshold values from 0.8 and upwards always resulted in image 5 as the reference, which suggests this range provides the most stable and accurate results, making it the preferred threshold choice for the graf dataset.

B. `wall` dataset

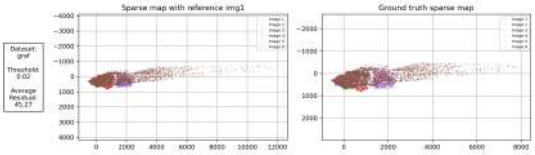
The `wall` dataset showed image 2 as the reference image across almost all threshold values. As illustrated in Figure 3 at an extremely low threshold of 0.001, image 1 or 3 were sometimes selected, and the residuals when using image 2 as reference were very high, above 10 pixels. As the threshold was adjusted, the average residual varied, with poor reconstructions for thresholds below 0.01 and above 3.0. The lowest residual error was observed at a threshold of 0.1, with similar results for thresholds near 1.0, as illustrated in Figure 4, confirming that image 2 is the preferred reference, and thresholds in the range of 0.1 to 1.0 offer a good balance between accuracy and stability for this dataset.

C. `bark` dataset

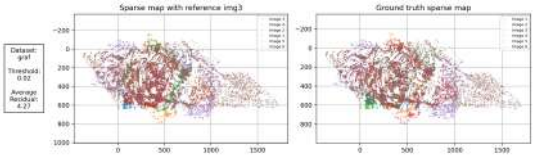
For the `bark` dataset, sensitivity to threshold variation was more pronounced. A illustrated in Figure 5, thresholds starting



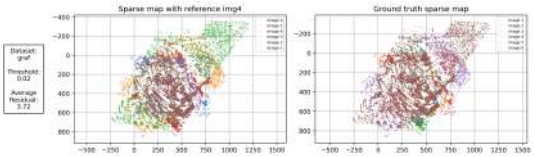
(a) Reference image 2



(b) Reference image 1

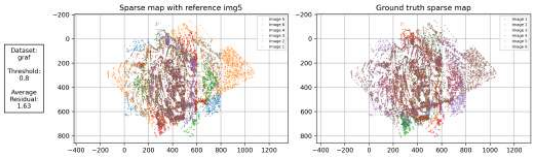


(c) Reference image 3

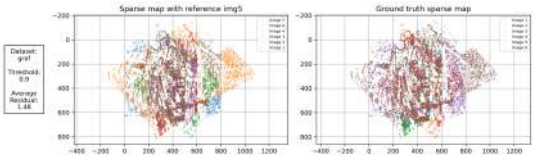


(d) Reference image 4

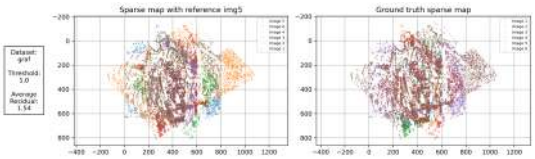
Fig. 1: Sparse maps for the `graf` dataset at small thresholds.



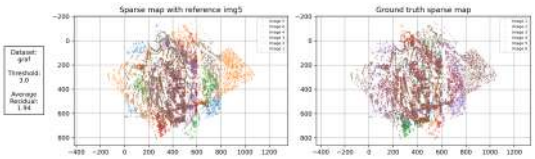
(a) Threshold 0.8



(b) Threshold 0.9

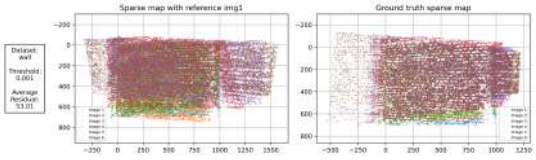


(c) Threshold 1.0

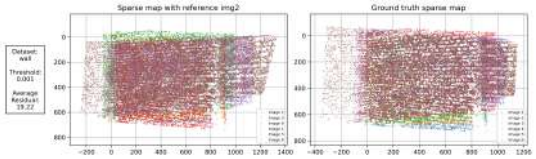


(d) Threshold 3.0

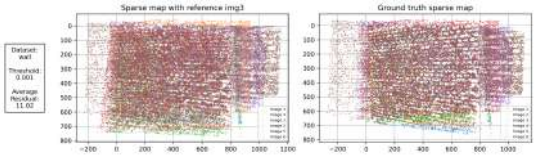
Fig. 2: Sparse maps for the `graf` dataset with reference image 5.



(a) Reference image 1

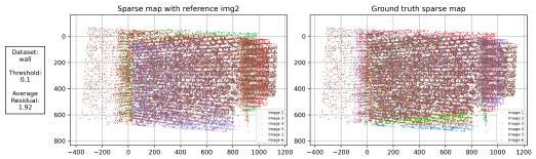


(b) Reference image 2

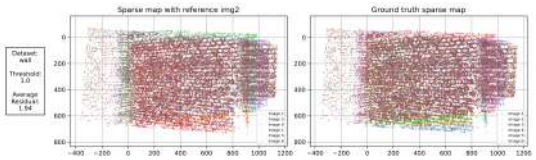


(c) Reference image 3

Fig. 3: Sparse maps for the `wall` dataset at threshold 0.001.



(a) Threshold 0.1



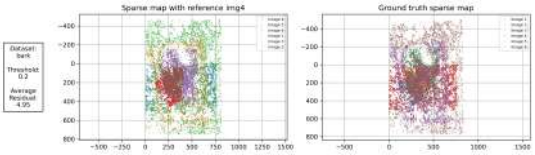
(b) Threshold 1.0

Fig. 4: Sparse maps for the `wall` dataset with reference image 2.

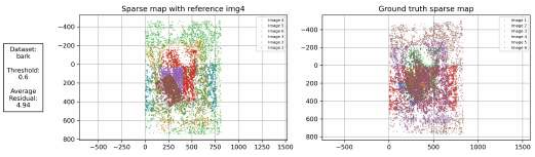
from 0.2 consistently resulted in image 4 being selected as the reference image. As illustrated in Figure 6, at thresholds of 0.1 and 0.01, image 2 was mostly chosen as the reference. At very low thresholds such as 0.001, the reference image alternated between images 1 and 5, which led to unstable reconstructions, as illustrated in Figure 7. When image 4 was selected as reference, residual errors remained stable around 5 pixels, with the lowest residual observed at a threshold of 0.6. These results indicate that for bark, image 4 paired with a mid-range threshold produces the most reliable reconstruction.

D. `boat` *dataset*

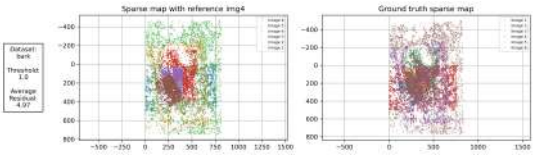
Finally, the `boat` dataset showed a strong dependence of the reference image on the threshold value. As illustrated in Figure 8, at thresholds of 1.4 and higher, image 1 was consistently chosen as the reference. Below 1.4, image 2 became the reference image, as illustrated in Figure 9. Residual errors were lower, approximately 8 pixels, when image 2 was



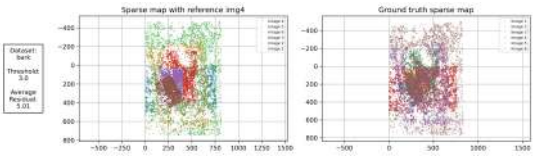
(a) Threshold 0.2



(b) Threshold 0.6

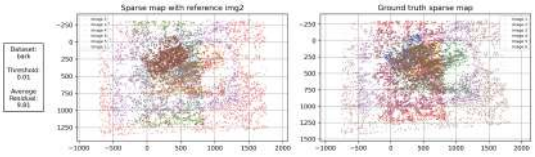


(c) Threshold 1.0

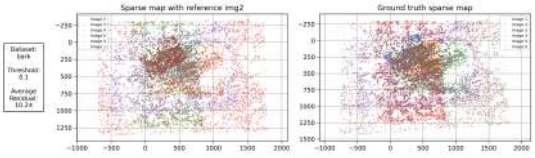


(d) Threshold 3.0

Fig. 5: Sparse maps for the `bark` dataset with reference image 4.



(a) Threshold 0.01



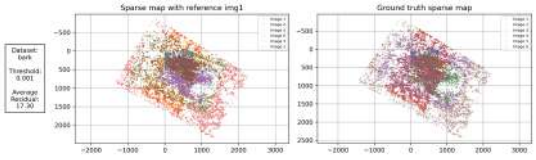
(b) Threshold 0.1

Fig. 6: Sparse maps for the `bark` dataset with reference image 2.

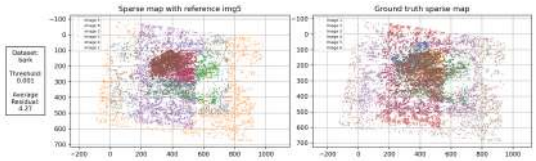
selected, and the best residual performance was achieved at a threshold of 1.0, making this the preferred setting for the boat dataset.

E. Best thresholds choice

Based on these results, a configuration with 1000 RANSAC iterations and thresholds tuned per dataset, as summarized in Table I, offered the best balance between accuracy and computational cost.

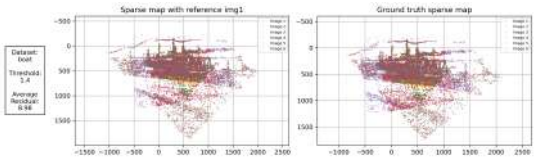


(a) Reference image 1

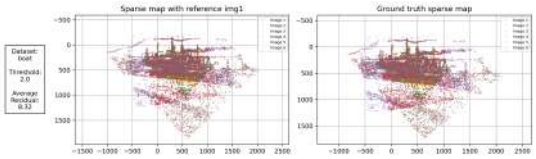


(b) Reference image 5

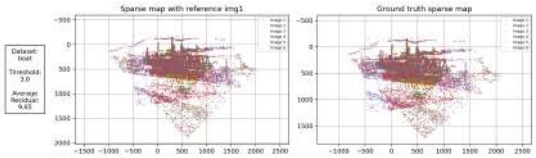
Fig. 7: Sparse maps for the `bark` dataset with threshold 0.001.



(a) Threshold of 1.4



(b) Threshold of 2.0



(c) Threshold of 3.0

Fig. 8: Sparse maps for the `boat` dataset with reference image 1.

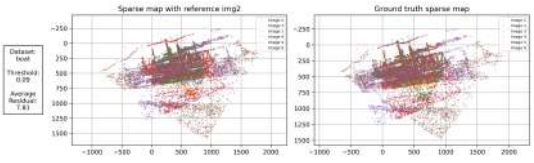
F. 2D sparse maps

For each dataset, the sparse 2D point map was constructed by projecting keypoints from each image into a global coordinate system defined by the reference image. Ground truth homographies, originally defined from the first image to the others, were re-referenced to match the coordinate system of the selected reference image. This allowed direct visual comparison between estimated and ground truth maps.

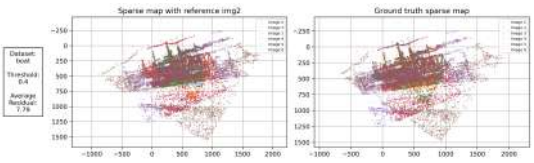
To quantitatively assess the reconstruction, the average

TABLE I: RANSAC Parameters for homography estimation

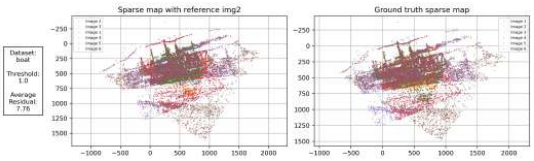
Dataset	Inlier threshold	RANSAC iterations
`graf`	0.8	1000
`wall`	0.1	1000
`bark`	0.6	1000
`boat`	1.0	1000



(a) Threshold of 0.09



(b) Threshold of 0.4



(c) Threshold of 1.0

Fig. 9: Sparse maps for the `boat` dataset with reference image 2.

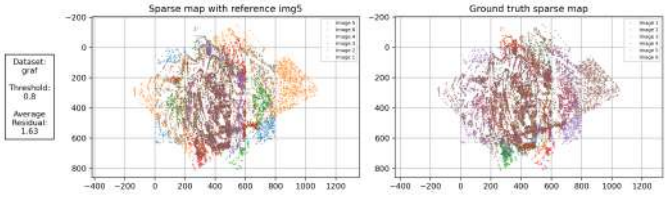


Fig. 10: Sparse map comparison for the `graf` dataset.

residual reprojection error was computed. This corresponds to the mean Euclidean distance between each sparse point in the estimated map and its counterpart in the ground truth map, aggregated across all images and matched keypoints. These results are summarized in Table II.

TABLE II: Average residual reprojection errors

Dataset	Avgerage residual (pixels)
`graf`	1.63
`wall`	1.92
`bark`	4.94
`boat`	7.76

Each evaluation is visually illustrated in Figures 10, 11, 12, and 13. These show the best reconstructions per dataset, with three panels: metadata and error statistics (left), sparse map from estimated homographies (middle), and the ground truth map (right).

III. Theoretical part

A. Bundle Adjustment in 2D

Bundle adjustment is a post-processing step used to improve the 2D Structure-from-Motion pipeline. This process refines both the global 2D point coordinates and the homographies,

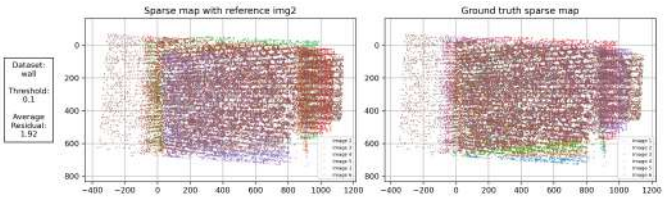


Fig. 11: Sparse map comparison for the `wall` dataset.

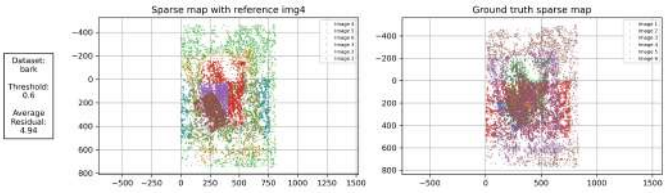


Fig. 12: Sparse map comparison for the `bark` dataset.

minimizing reprojection error between observed 2D features in each image and their corresponding projections.

Let $u_i = (x_i, y_i)$ denote the i-th point in global 2D space, and $H_j \in \mathbb{R}^{3\times3}$ be the homography mapping global coordinates to the coordinate system of image j. The projection of the i-th point into image j is computed by applying the homography H_j to u_i in homogeneous coordinates, $\tilde{u}_i = [x_i, y_i, 1]^\top$:

$$\hat{u}_{ij}^{(hom)} = H_j \tilde{u}_i = \begin{bmatrix} \hat{x}_{ij} \\ \hat{y}_{ij} \\ \hat{w}_{ij} \end{bmatrix}.$$

Then, the result is dehomogenized to obtain the Euclidean coordinates of the i-th point in image j:

$$\hat{u}_{ij} = \begin{bmatrix} \hat{x}_{ij}/\hat{w}_{ij} \\ \hat{y}_{ij}/\hat{w}_{ij} \end{bmatrix}.$$

The observed position of point i in image j is denoted by $u_{ij} = (x_{ij}, y_{ij})$. Therefore, the residual between the observed point u_{ij} and its projection $\hat{u}_{ij}$ defines the reprojection error:

$$r_{ij}(u_i, H_j) = \left\| \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} - \frac{1}{\hat{w}_{ij}} \begin{bmatrix} \hat{x}_{ij} \\ \hat{y}_{ij} \end{bmatrix} \right\|^2.$$

The equation that should be minimized during Bundle Adjustment in 2D is the sum of squared reprojection errors between the observed 2D points in the images and the corresponding

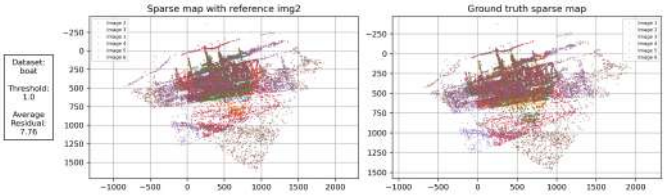


Fig. 13: Sparse map comparison for the `boat` dataset.

points projected from the global 2D coordinates the estimated homographies. The minimization objective is:

$$f(u, H) = \sum_{(i,j)} \|r_{ij}(u_i, H_j)\|^2 = \sum_{(i,j)} \left\| \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} - \frac{1}{\hat{w}_{ij}} \begin{bmatrix} \hat{x}_{ij} \\ \hat{y}_{ij} \end{bmatrix} \right\|^2.$$

B. Generative models

1) ELBO optimization: The goal in a variational autoencoder (VAE) is to generate new data samples that resemble those in the training set by learning the underlying data distribution $p(x)$. Since data x is often high-dimensional and complex, a latent variable z is introduced to capture its essential structure. This leads to the formulation $p(x) = \int p(x \mid z)\, p(z)\, dz$, where $p(z)$ is a simple prior distribution (typically a standard Gaussian), and $p(x \mid z)$ is the decoder network that generates x from z.

To train the model, the log-likelihood $\log p(x)$ needs to be maximized. However, computing this quantity requires integrating over all possible values of z, which is generally not posssible when z is high-dimensional and $p(x \mid z)$ is defined by a neural network.

To make the optimization tractable, an approximate posterior distribution $q(z \mid x)$ is introduced. This distribution, parameterized by the encoder network, estimates what the true posterior $p(z \mid x)$ might look like. Using this approximation, a lower bound on the log-likelihood, known as the Evidence Lower Bound (ELBO), can be derived:

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x \mid z)] - \mathrm{KL}(q(z \mid x) \,\|\, p(z)).$$

The first term, $\mathbb{E}_{q(z|x)}[\log p(x \mid z)]$, encourages the decoder to reconstruct the input x accurately from the latent code z. The second term, the KL divergence between $q(z \mid x)$ and the prior $p(z)$, regularizes the encoder by forcing its output distribution to remain close to the prior. This makes the latent space smooth and suitable for generative sampling.

2) VAEs and Diffusion probabilistic models: Both variational autoencoders (VAEs) and diffusion probabilistic models are generative models. However, they differ significantly in the encoding and decoding processes, as well as in the use of latent variables.

In VAEs, training involves maximizing the variational lower bound. Specifically, data is mapped into a low-dimensional latent space through a single-step encoder that approximates the posterior distribution $q(z \mid x)$. A sample from this distribution is then passed to a decoder, which reconstructs the input in one step via $p(x \mid z)$. The latent variable z is a compressed representation of the input, capturing its essential features.

In contrast, the training of diffusion models involves predicting and removing noise. Specifically, the data is progressively corrupted by Gaussian noise through a forward process that transforms the input into pure noise over many steps. The reverse process, which serves as the generative mechanism, is learned and consists of sequential denoising steps that aim to recover the original data. This reverse process is typically modeled by a neural network trained to predict the noise at each step.

While VAEs provide a structured latent representation producing low-quality content, diffusion models operate in the high-dimensional data space producing high-quality content. Howeve, diffusion models achieve higher sample quality at the cost of slower generation and greater computational complexity.

C. Triangulation

1) Recovering 3D points from 2D observations: Triangulation is used in Structure-from-Motion to estimate 3D points from multiple 2D observations. Given a set of 2D pixel positions $x_i \in \mathbb{R}^2$ observed in k images and the corresponding camera projection matrices $P^{(i)} \in \mathbb{R}^{3\times4}$, the goal is to reconstruct the 3D point $\hat{X}_i \in \mathbb{R}^3$ that best explains the observations.

Each 2D point x_i in an image corresponds to a ray in 3D space that originates from the camera center and passes through the pixel location. The intersection of these rays from different camera viewpoints defines the 3D point. However, due to noise and calibration errors, the rays do not intersect exactly. Triangulation therefore aims to find the 3D point that minimizes the reprojection error.

For each image i, the camera equation can be written as

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = P^{(i)} \begin{bmatrix} \hat{X}_i \\ 1 \end{bmatrix},$$

where λ_i is a scalar representing the depth. The projection matrix P is usually expressed as $P = K[R \mid t]$, where K is the intrinsic camera matrix containing the focal lengths and principal point, and $[R \mid t]$ are the extrinsic parameters representing the rotation R and translation t from the world coordinate system to the camera coordinate system. Alternatively, t can be written as $-RC$, where C is the position of the camera center in world coordinates.

To estimate $\hat{X}_i$, a common approach is to minimize the reprojection error $\|x_i - \hat{x}_i\|^2$, where the projected point $\hat{x}_i$ is defined as:

$$\hat{x}_i = \begin{bmatrix} \frac{P_1 \hat{X}_i}{P_3 \hat{X}_i} \\ \frac{P_2 \hat{X}_i}{P_3 \hat{X}_i} \end{bmatrix},$$

with $P_j \hat{X}_i$ denoting the j-th row of the projection matrix P applied to the homogeneous 3D point. Since the division by $P_3 \hat{X}_i$ introduces non-linearity, the minimization is typically performed using Gauss-Newton optimization with a fixed learning rate.

In practice, RANSAC is often used to increase robustness. A minimal set of observations (at least two) is used to triangulate a candidate 3D point, and its reprojection error is computed across all views. Observations with error below a threshold are considered inliers.

2) Ambiguity with two views and known relative pose: Considering the case of $k = 2$ images with known relative pose, represented by the projection matrices $P^{(1)}$ and $P^{(2)}$, the 2D image points x_1 and x_2 map to two rays in 3D, one

for each camera. Ideally, the 3D point lies at the intersection of these rays. However, in special cases, there may be infinitely many 3D points that project exactly to x_1 and x_2.

This occurs when the two rays are colinear, meaning they lie along the same line in space. In that case, any 3D point along the ray will be projected to the same pixel coordinates x_1 in the first image and x_2 in the second. Triangulation becomes ambiguous, and a unique 3D point cannot be determined.

3) Relative Pose without image observations: Although the relative pose, defined by a rotation R and translation t between the cameras, is known, it only describes how the cameras are positioned with respect to each other. It provides no information about the specific image points x_1 and x_2 observed in the two views. Therefore, it is not possible to derive the pixel coordinates x_1 and x_2 from the relative pose alone. These positions must be observed in the images and cannot be recovered from geometry alone.

IV. Use of LLMs

ChatGPT was used as a supplementary tool during the development and documentation of this project. Its use was limited to conceptual clarification, assistance in understanding and resolving errors, and language refinement. Specifically, the model contributed to the following:

- Reading images to grayscale: it was used to understand why grayscale is preferred for SIFT. It explained that grayscale improves speed, reduces memory use, and gives more consistent results since SIFT works on intensity gradients.
- Normalization: helped clarify how and why 2D points are normalized by moving their centroid to the origin and scaling them so their average distance to the origin is $\sqrt{2}$.
- Global coordinate mapping: it helped explain how to define a global reference frame by assigning the identity homography to the reference image and using the inverse of pairwise homographies to align the others.
- Debugging: it was used to understand and solve some coding or logic errors during implementation.
- Visualization improvements: it gave suggestions to improve the layout and clarity of plots and results.

References

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.