Rotation Averaging for Map Merging and Trajectory Alignment in Global Structure-from-Motion

Elina Strömberg

Master's thesis
2025:E51

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Rotation Averaging for Map Merging and Trajectory Alignment in Global Structure-from-Motion

---

Student: Elina Strömberg
Supervisors: Viktor Larsson and Gabrielle Flood
Examiner: Professor Carl Olsson

Abstract

Merging of maps, such as 3D point clouds obtained from Structure-from-Motion or visual SLAM, can enhance robustness and expand covered areas in applications like multi-robot systems or augmented reality. When direct data fusion is difficult, an alternative approach is to align pose trajectories. This work investigates the use of rotation averaging as a method for indirect map merging, especially for sequential camera setups where loop closures are typically unavailable. Representing initially aligned data as graphs, with absolute rotations as nodes and relative rotations as edges, we incorporate two-view geometric constraints between sequence pairs using PoseLib, and jointly optimize using weighted edge constraints. Both quantitative evaluations and qualitative visualizations are presented. Results suggest that incorporating rotation averaging between sequences could possibly fine-tune alignment precision, although sensitivity to parameter choices and other issues remain. The study concludes with suggestions for further experiments to advance accuracy and robustness, eventually in practical map merging scenarios including integration with supplementary methods and full pose estimation frameworks involving translation.

Keywords: trajectory alignment, rotation averaging, map merging, COLMAP, GLOMAP, NetVLAD, PoseLib, Ceres, LaMAR dataset

POPULÄRVETENSKAPLIG SAMMANFATTNING

I många moderna tekniker med exempelvis robotar och förstärkt verklighet behövs tillförlitlig rörelsedata och kartor över omgivningen. Detta projekt fokuserar på att förbättra hur enhetens kamerariktningar, i en sekvens över tid, kan finjusteras efter sammanslagning av två eller fler sådana bildsekvenser, som initiellt roterats till att överlappa utan att deformeras. Genom att använda förhållanden mellan par av kameror, samt ett neuralt nätverk för effektivitet, ska den ursprungliga anpassningen förbättras med hjälp av så kallad rotation averaging, en metod med syfte att "balansera" uppskattningarna av kamerornas orienteringar.

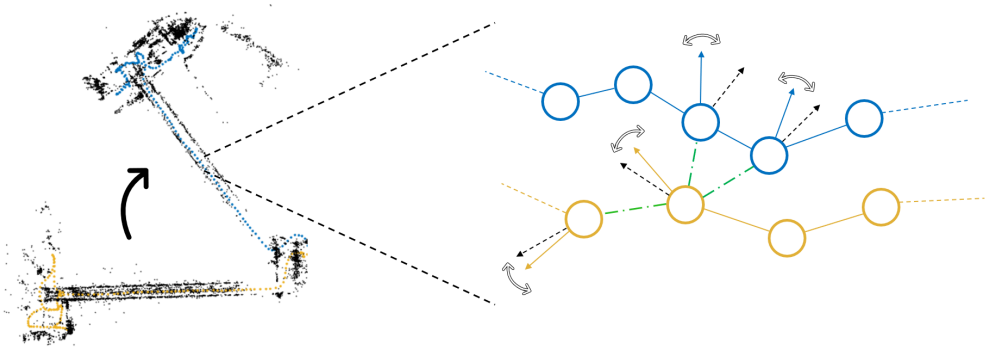
I takt med att självkörande bilar, robotar och förstärkt verklighet (AR) blir allt vanligare, behövs smarta metoder för att avgöra var en användare eller enhet befinner sig och dess rörelsemönster. Detta görs ofta genom att följa en bana (eng: *trajectory*) och samtidigt kartlägga omgivningen. Men vad händer när man vill slå samman data från flera olika tillfällen eller enheter, för att få en mer robust karta eller bana?

Att kombinera data från flera sessioner är ett växande behov, inte minst inom tillämpningar där data samlas in i omfattande mängder eller över tid (t.ex. crowdsourcing eller långvariga robotinsatser). Det möjliggör både utökad täckning av miljön och förbättrad tillförlitlighet. Vissa tillvägagångssätt för sådan anpassning (eng: *alignment*) av datasekvenser mot varandra, samt kartläggning av robotens omgivning, använder bland annat tredimensionella punktmoln (en samling av många små punkter som tillsammans visar formen på ett föremål eller omgivning). Dessa nyttjas delvis i detta projekt, men det huvudsakliga målet är att anpassa själva kamerorna (banorna) mot varandra.

Genomförandet delas översiktligt upp i två steg: först en stel anpassning av datan, för att sedan finjustera med hjälp av så kallad *rotation averaging*, en optimeringsmetod där vi försöker hitta en balans i hur kamerornas orienteringar överrensstämmer baserat på olika villkor och restriktioner mellan dem. Se figur P.1 nedan för en illustration.

Dessa villkor mellan kamerorna fås ur så kallad tvåbildsgeometri och bidrar med information om hur två kameror är orienterade i förhållande till varandra. Till följd av brus, extremvärden, felaktiga kamerapar, etc. kommer alltså inte geometrin mellan ett par nödvändigtvis stämma överens med ett annat, och därav görs den avvägning som visualiseras till höger i figuren nedan. Vi kombinerar även ramverket med ett neuralt nätverk för att effektivt hitta bildpar för tvåbildsgeometrin.

Sammanfattningsvis visar projektet på en möjlighet att förbättra noggrannheten i kameraorienteringar efter att två bildsekvenser har förts samman, vilket i förlängningen ska kunna bidra till mer exakta och robusta kartor.



Figur P.1: Illustration av de två huvudsakliga utföranden: 1. Inledande stel rotation (vänster): Två sekvenser bestående av ca 100 kameror vardera (blå respektive gula punkter) från två AR-sessioner tagna i en och samma korridor, och deras tillhörande punktmoln (svarta punkter). Anpassa den ena sekvensen till den andra (samt deras punktmoln) med hjälp av en stel rotation. Detta motsvarar att placera dem i ett gemensamt koordinatsystem och tar oss en god väg mot den slutliga lösningen. 2. Finjustering (höger): Zooma in på en del av sekvenserna, här fem respektive fyra kameror vardera. Finjustera kameraorienteringar baserat på villkor mellan kamerapar (illustrerade av gröna, "alternerande streck-punkt" kanter i grafen), både inom och mellan sekvenser.

Acknowledgements

I would like to express my gratitude to Viktor Larsson and Gabrielle Flood for the opportunity to undertake this project and for their continuous guidance throughout the supervision. I am grateful for your valuable feedback, insightful suggestions and consistent encouragement. I would also like to thank Professor Carl Olsson for acting as the examiner on this thesis.

Additionally, I am deeply thankful to my family for their unconditional love and unwavering support throughout this journey.

Contents

1

Introduction

1.1 Background

Pose estimation of cameras and mapping of scene points are two central components of computer vision, commonly known as Structure-from-Motion (SfM) or visual SLAM (Simultaneous Localization and Mapping). Increasing amounts of data, e.g. from crowd-sourcing, allows for building more robust systems by fusing information across multiple sessions. Such map merging and trajectory alignment play a key role in achieving optimal performance, whether in augmented reality, for autonomous vehicles or enabling a robot to revisit known environments, as well as other contemporary subjects.

Pioneering map merging rely mostly on geometric and probabilistic methods such as Iterative Closest Point and Procrustes, or optimization of pose graphs. Many recent approaches lean more towards deep learning. However, traditional concepts may still be incorporated, while using frontier mechanisms for other sub-components of the pipeline, thus maintaining foundational reliability while targeting wider applications.

A common optimization technique is *Rotation Averaging*, as cameras are usually defined decomposed into their orientational and positional components. It was introduced by Govindu in the early 2000s and has since expanded, now existing many variants depending on context and complexity of the problem.

1.2 Project Purpose & Formulation

In this project, it was investigated how rotation averaging could be used in merging of maps and trajectories, by leveraging estimated/computed relative rotations between/within trajectories as edge constraints in a view graph, where optimizing absolute rotations.

The main objectives are to:

- Transform two trajectories from separate coordinate systems into a common reference frame.
- Gain insight into possibilities of fine-tuning the camera orientations of the pre-aligned sequences (with merged SfM maps) using rotation averaging.
- Explore how it can be coupled with existing renown frameworks, for other parts of the pipeline.

In other words, if having initially aligned two sequences using a common transformation for all poses, there may be some remaining inconsistency between computed and actual rotations, unless working with fixed sensor rigs that will not capture different viewpoints between sessions. The goal is to introduce some flexibility to the system, individually adjusting rotations using reliable two-view geometry from PoseLib. We try for two different initializations, one of them being with rotation from Procrustes which uses scene points and the other being found with camera poses and estimated relative rotations.

The trajectories' images are passed through SfM software, here COLMAP or GLOMAP, to obtain reconstructed scene points as well as keypoint matches, input to PoseLib for accessing reliable two-view geometry. NetVLAD, a network layer (integrated to any convolutional neural network), is used to improve the efficiency of searching for image pairs between sequences which can be a time-consuming step in SfM.

The task becomes particularly challenging in long-term operations, where lighting, viewpoint, and appearance can vary significantly over time. We utilize the LaMAR dataset which addresses these concerns. Additionally, many related applications follow sequential patterns, where lacking loop closures, thus this dataset being a suitable choice for reflecting real-case scenarios.

1.3 Thesis Structure

Following is the report structure: In Chapter 1, the project is introduced, outlining its objectives and scope. Chapter 2 presents related theory and foundational concepts, with certain background under Appendix A. The chapter concludes by formulating the problem addressed in the project. Chapter 3 describes the dataset and provide details about the implementation. This is followed by explanation of the experiments, performance metrics along with qualitative assessment of the results in Chapter 4, with comments and interpretations in Chapter 5, where also discussing potential future improvements. Finally, the key outcomes of the project are summarized in Chapter 6, which concludes the report.

2

Theory

In some of the following sections, the reader is assumed to be somewhat familiar with computer vision-related naming conventions and concepts, and is otherwise referred to Appendix A.1, or other more thorough material.
Some fundamental theory closer related to the project will nevertheless be presented in this chapter, as well as general surface-level descriptions of two of the programs and a toolbox that were used. The last section includes a definition of the problem formulation.

2.1 Rotation Representations

Different rotation conventions may be chosen depending on user preferences, application contexts, and the specific advantages each representation may offer.

2.1.1 Special Orthogonal Group SO(n)

The main components of interest used in this project are three dimensional camera rotations

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.1}$$

belonging to the *Special Orthogonal Group* $SO(3)$, where

$$SO(n) := \left\{ R \in \mathbb{R}^{n \times n} \;\middle|\; \underbrace{R^T R = I}_{(*)}, \underbrace{\det(R) = 1}_{(**)} \right\}.$$

A practical meaning of the orthogonality $(*)$ will be the preserving of geometries, such as distances and angles. There is also the applicable benefit of the invertibility (having $R^{-1} = R^T$), facilitating back-and-forth mapping, which is useful where transformations have to be undone.
The determinant constraint $(**)$ ensures the absence of reflections (flipping the orientation about some axis), so that $SO(n)$ will consist of proper rotations. The *Special Orthogonal Group* is also called the *Rotation Group*.

2.1.2 Quaternion Representation

A *quaternion* q is defined as

$$q = q_w + q_x i + q_y j + q_z k$$

where $q_w \in \mathbb{R}$ is the scalar part and $q_x, q_y, q_z \in \mathbb{R}$ form the vector part, fulfilling

$$i^2 + j^2 + k^2 = ijk = -1 \tag{2.2}$$

where i, j, k are imaginary units extended into 3D space and can be interpreted as directional axes.[1]

[1] As is often recounted, Sir William Rowan Hamilton discovered quaternions on October 16, 1843, reportedly carving the fundamental identity (2.2) into the stone of Brougham Bridge in Dublin. This occurred before vectors existed in their modern form, and their development was in part influenced by quaternions. Interestingly, the term *scalar*, in its usual mathematical context, is said to have originated from quaternion theory.

A quaternion can be implicitly written on vector form as

$$q = [q_w, q_x, q_y, q_z] \tag{2.3}$$

or

$$q = [q_x, q_y, q_z, q_w] \tag{2.4}$$

depending on convention. It is important that the quaternion is normalized, to represent a true rotation. Two benefits over rotation matrices are the fewer parameters as well as there only being a unit norm constraint

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1.$$

See Appendix A.2 for how one can convert between the quaternion and matrix representations.

2.1.3 Axis-Angle Representation

One can represent a 3D rotation using a unit vector $\mathbf{n} = (x, y, z)$ around which a rotation of angle θ occurs, where the *axis-angle representation* is given as

$$\omega = \theta \mathbf{n}.$$

It is sometimes also called the *rotation vector*. The rotated vector ω_{rot} when rotating ω around axis $\mathbf{n}$ by θ is obtained by Rodrigues' formula as

$$\omega_{\text{rot}} = \omega \cos\theta + (\mathbf{n} \times \omega) \sin\theta + \mathbf{n}(\mathbf{n} \cdot \omega)(1 - \cos\theta) \tag{2.5}$$

and (2.5) can also be reorganized and used for obtaining the corresponding rotation matrix and vice versa.

2.2 Map Merging

A *map*[2] can have multiple different meanings in the field of computer vision, here it is given by a point cloud (set of scene points) obtained through cameras. The merging of maps refer to building a model of the environment by fusing two or more SfM reconstructions (or constructed from other vSLAM methods). This goes hand-in-hand with trajectory alignment, connecting (sequential) sets of cameras, and one can opt for optimizing either based on scene points (as for Procrustes) or cameras (as for rotation averaging), here also trying a combination of them both.
There are many different tools for map merging, such as the open-source framework maplab [1, 2], which uses mapping and localization with visual-inertial odometry (based on ROVIO [3]) and incorporates for instance pose graph relaxation[3] with loop closures.
A short out-take of other methods include iterative closest point (ICP) - often as a refinement step or as the robuster, trimmed version (TrICP) [5] - or merging based on occupancy grids [6]. It is common practice to include techniques such as Kalman filtering (EKF-SLAM) [7, 8, 9], expectation maximization (EM), or other methods, such as with the Hough transform [10, 11]. Pipelines like [12] also uses NetVLAD. In [9, 13] they use tricks such as decoupling the Jacobian into blocks of camera parameters and 3D points and as in this project, they also use Procrustes as a baseline method, described below.

2.2.1 Procrustes Transformation

One may use Procrustes transformation to rotate, translate and scale one object to another.[4] Given two sets of scene points $\mathbf{X} = [X_1, X_2, \ldots, X_N] \in \mathbb{R}^{N \times d}$ and $\mathbf{Y} = [Y_1, Y_2, \ldots, Y_N] \in \mathbb{R}^{N \times d}$, the goal is to find R^*, t^* and s^* that solve

[2] Not to be confused with the later use of the word *map* (written in italic) throughout the remainder of the report.
[3] Relaxation refers to a set of techniques that simplify a difficult problem into a similar one. Pose graph optimization (PGO) is a vSLAM method which is in some ways similar to rotation averaging, see e.g. [4] for a tutorial deriving the non-linear least squares from a maximum likelihood approach with Gaussian noise and solving iteratively, alternatively optimizing using manifold properties.
[4] Most of the following steps were procured from instructions provided in a lecture in the course *Medical Image Analysis (FMAN30)* at LTH.[14]

$$\arg\min_{R,t,s} \sum_{i=1}^{N} ||Y_i - t - sRX_i||^2, \tag{2.6}$$

that is, aligning the first point cloud to the second. This is done by first defining the centroids

$$\bar{\mathbf{X}} = \sum_{i=1}^{N} \frac{X_i}{N} \quad \text{and} \quad \bar{\mathbf{Y}} = \sum_{i=1}^{N} \frac{Y_i}{N} \tag{2.7}$$

and subtracting them to the points. For simplified notation, denote

$$\tilde{X}_i = X_i - \bar{\mathbf{X}} \quad \text{and} \quad \tilde{Y}_i = Y_i - \bar{\mathbf{Y}}.$$

This corresponds intuitively to eliminating translation from the optimization. Let $H = \sum_{i=1}^{N} \tilde{X}_i \tilde{Y}_i$ and use Singular Value Decomposition (SVD) to get $H = U\Sigma V^T$.[5] The optimal rotation, scale and translation are then given as

$$R^* = U \operatorname{diag}\left(\left[1, 1, \det\left(UV^T\right)\right]\right) V^T, \tag{2.8}$$

$$s^* = \frac{\sum_{i=1}^{N} \tilde{Y}_i^T R^* \tilde{X}_i}{\sum_{i=1}^{N} ||\tilde{X}_i||^2} \tag{2.9}$$

and

$$t^* = \bar{\mathbf{Y}} - s^* R^* \bar{\mathbf{X}}. \tag{2.10}$$

See Appendix A.3 for a derivation of the above. In the following, we denote $R_P = R^*$.

2.3 Rotation Averaging

The so called *relative* rotation between two cameras $P_i = \begin{bmatrix} R_i & t_i \end{bmatrix}$ and $P_j = \begin{bmatrix} R_j & t_j \end{bmatrix}$ is defined as

$$R_{ij} = R_j R_i^T.$$

It can be regarded as representing the value of an edge $(i, j) \in \mathscr{E}$ connecting nodes i and j in a view graph $\mathscr{G}$, see figure 2.5 (where instead using the naming Q and S to distinct between sequences; the notation with R above being intended as a general description). *Estimated* relative rotations are denoted as $\hat{R}_{ij}$. Now overlooking any necessary translation, the relative rotation essentially describes how to rotate camera P_i to get P_j, and under ideal conditions it should hold that

$$R_{ij} R_i = R_j, \quad \forall (i, j) \in \mathscr{E}.$$

Rotation Averaging is the process of solving

$$\arg\min_{R_1,\ldots,R_n} \sum_{(i,j)\in\mathscr{E}} d\left(\hat{R}_{ij}, R_j R_i^T\right)^p \tag{2.11}$$

which intuitively gives motivation for its name, since finding some "balance" between multiple cameras' rotations. A comprehensive tutorial is found in [15], where (2.11) is known as *Multiple* rotation averaging. One should choose the distance metric $d(\cdot,\cdot)$ as best suited, depending on the used rotation representation. In our case we use the Frobenius norm[6]. The minimization in (2.11) is a non-linear (and non-convex) least squares problem, since here choosing $p = 2$, although an L_1 minimization might be robuster.

Note that rotation averaging only affects the orientation, and should therefore be performed in combination with subsequent *Translation Averaging*. Rotation averaging is a *global* method, meaning it estimates all rotations simultaneously to minimize overall inconsistency, rather than refining them sequentially or locally.

[5] U and V are unitary matrices and Σ is diagonal, containing the singular values, which can be said to represent the magnitude of each dimension's contribution to the structure of the data.

[6] $||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$.

2.4 Solving the Non-Linear Least Squares

There exists various methods for solving the rotation averaging problem (2.11), see Appendix A.5 for some common practice. The approach of Levenberg-Marquardt [16, 17] was mostly relied on for this project.

2.4.1 Levenberg-Marquardt

The following is mainly described as of the optimization toolbox *Ceres*, section 3.5.2, and some background from pedagogic explanations in [18]. Levenberg-Marquardt is a trust region method closely related to the method of Gauss-Newton, which uses a Taylor approximation to solve in a *linear* least squares sense.

The problem (2.11) essentially is about minimizing the sum of a set of residuals $r_i(x)$, that is

$$\arg\min_x \sum_i r_i(x) = \arg\min_x ||r(x)||^2. \tag{2.12}$$

Gauss-Newton linearizes these residuals around an initial (and later updated) point x_0

$$r(x) = \begin{pmatrix} r_1(x) \\ \vdots \\ r_n(x) \end{pmatrix} \approx \underbrace{\begin{pmatrix} r_1(x_0) \\ \vdots \\ r_n(x_0) \end{pmatrix}}_{r(x_0)} + \underbrace{\begin{pmatrix} \nabla r_1(x_0)^T \\ \vdots \\ \nabla r_n(x_0)^T \end{pmatrix}}_{J(x_0)} \underbrace{(x - x_0)}_{\Delta x}$$

where $J(x_0)$ is the Jacobian of the vector r. This breaks down to solving the approximated *linear* least squares problem

$$\arg\min_{\Delta x} ||r(x_0) + J(x_0)\Delta x||^2 \tag{2.13}$$

with solution[7]

$$\Delta x = -\left(J(x_0)^T J(x_0)\right)^{-1} J(x_0)^T r(x_0).$$

The Levenberg-Marquardt method adds a penalty to (2.13) for large steps Δx, with the aim to mitigate instability too far from the minimum.[8] Thus, instead solve

$$\arg\min_{\Delta x} \left(||r(x_0) + J(x_0)\Delta x||^2 + \lambda ||\Delta x||^2 \right). \tag{2.14}$$

The user can choose either an *exact*[9] or *inexact step* version of Levenberg-Marquardt in Ceres, where the former uses a Cholesky or QR decomposition and the latter solves (2.14) iteratively, see Appendix A.4.

Worth mentioning is that a good initialization is crucial for Levenberg-Marquardt since the method will *locally* search for a solution and face the risk of getting stuck in suboptimal minima, many of which exist due to the non-convexity of the optimization (since $SO(3)$ is non-convex).[10,11] In this project, we initialize by first applying R_P, section 2.2.1, or the so called global rotation R_g, section 3.3, to one of the sequences, thus aiming to circumvent the most difficult global optimality issues.

[7] From the normal equations.

[8] The damping, or relaxation, parameter λ acts as a Lagrange multiplier controlling the trust region size. A large λ corresponds to a small trust region (not trusting the model much), and vice versa. With $\lambda \longrightarrow \infty$, Levenberg-Marquardt becomes gradient descent. The damping parameter is adjusted automatically by Ceres for optimal performance.

[9] Note that the solution will nevertheless *not* be exact per se, due to the regularization term.

[10] A convex set is a subset $C \subseteq \mathbb{R}^n$ of a vector space where, for any two points in the set, the line segment connecting them lies entirely within the set. That is, C is convex if for any $x, y \in C$ then $\lambda x + (1-\lambda)y \in C$ for all $\lambda \in [0,1]$.

[11] To avoid this, one could apply convex relaxation methods such as semidefinite programming, such as in [19] and [20].

2.4.2 Robustifiers

To dampen the effect of outliers, it is common practice to use *robustifiers*. One often used is the Huber loss function, defined as

$$\rho_H(r(x)) = \begin{cases} r(x)^2, & r(x) \leq \delta_H \\ 2\delta_H \left(\sqrt{r(x)^2} - \delta_H \right), & r(x) > \delta_H \end{cases}$$

with squared residuals as in (2.12) and $\delta_H > 0$. We also use Cauchy loss, defined as

$$\rho_C(r(x)) = \frac{\delta_C^2}{2} \ln \left(1 + \left(\frac{r(x)}{\delta_C} \right)^2 \right) \tag{2.15}$$

with $\delta_C > 0$.

2.5 Structure from Motion

For this project, a map (point cloud) was created in either COLMAP or GLOMAP and may look like in figure 2.1.

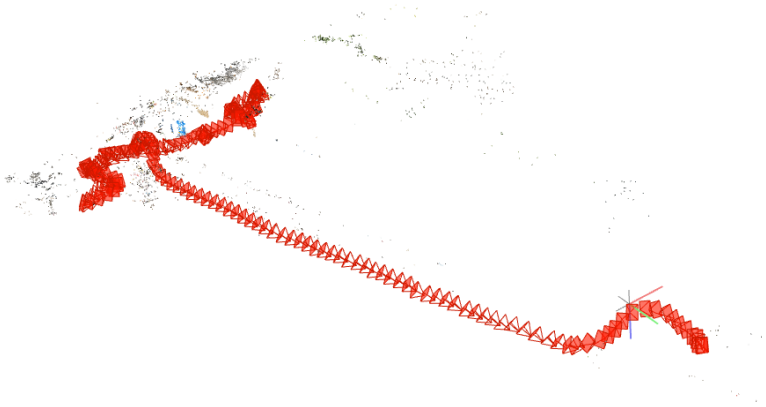


Figure 2.1 A camera trajectory from the LaMAR dataset (red) and its 3D reconstruction (sparse point cloud) viewed in the COLMAP GUI.

2.5.1 COLMAP

The popular open-source software COLMAP, based on the paper *Structure-from-Motion Revisited* [21], adopts an *incremental* SfM approach - successively adding more images to the reconstruction - and follows the steps as illustrated in figure 2.2. See next section for a *global* approach, considering all cameras at once.
The components of the *first* part of the COLMAP pipeline, the **Correspondence Search**, is here described on a surface-level in order to somewhat understand the distinction to GLOMAP in next section. A summary of the next part, the **Incremental Reconstruction**, is left in Appendix A.6 for the interested. The user input is a set of (partially overlapping) images. Again, see [21] or specifications for details on the below.

Feature Extraction Initially, SIFT (scale-invariant feature transform) [22] is used to detect local potential features, with associated keypoints or salient points (coordinates) tagged with descriptors[12] or "fingerprints/appearance". They chose SIFT for its robustness. Alternatively, the user can input other externally precomputed features of choice.

Matching Feature correspondences between images are then searched for by comparing descriptors using some similarity metric, and outputting a set of potentially overlapping image pairs and associated feature correspondences. COLMAP offers six different matching techniques. In this project we use *exhaustive*, *sequential* and *custom* matching.

[12] SIFT uses 128 dimensional element feature vectors for describing the gradients around each keypoint. Details in [22].

Two-view Geometric Verification The previous step only take feature *appearances* into account and no actual locations. This third stage of the process verifies potentially overlapping image pairs by using projective geometry to ensure that matched features correspond to the same scene point. Transformations like homographies or epipolar geometry are estimated to describe the geometric relationship between images, with robust methods like RANSAC addressing outliers. Images are considered geometrically verified if a valid transformation successfully maps a sufficient number of features between them. The outcome is a *view graph* $\mathcal{G}_{\text{sfm}}$, where images serve as nodes, and verified image pairs with their geometric relations form the edges. This view graph is the input to the next stage, the incremental reconstruction, where each node is considered one at a time. For GLOMAP, the above steps in correspondence search are the same, but the view graph instead enters a **Global Reconstruction**.

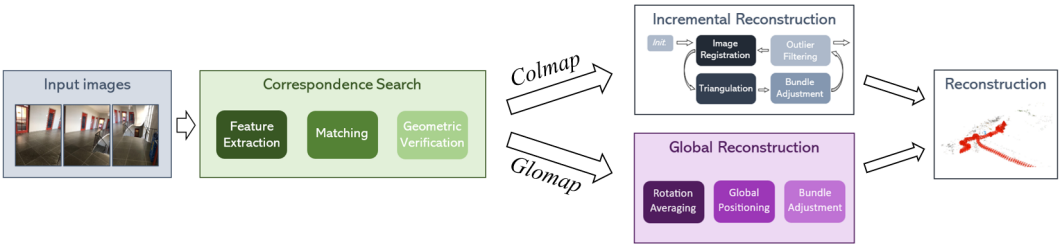


Figure 2.2 A simplified illustration of the COLMAP vs GLOMAP pipelines. Figure adapted from [21] and [23].

2.5.2 GLOMAP

Whilst incremental approaches excel in accuracy and robustness, global methods offer significantly greater scalability and efficiency. Nevertheless, COLMAP's global successor GLOMAP [23], see figure 2.2, retain the superior speed without compromising the results. For the LaMAR dataset - used in this project, see section 3.1 - the reconstruction process for GLOMAP is nearly 30 times (28.6x) faster than for COLMAP[13], and providing better recall[14] and AUC[15].
GLOMAP is also more closely related to this project since using rotation averaging, on the view graph $\mathcal{G}_{\text{sfm}}$ obtained as described above, solved with their implementation of the **L1-IRLS** algorithm, see Appendix A.5.2. That is, all nodes (images) are optimized at once by considering all edges (two-view geometries).
After obtaining the camera orientations, the positions should be found, which has traditionally been done through translation averaging. This however suffers from multiple limitations.[16] Instead, they perform the so called *Global Positioning* which is "joint global triangulation and camera position estimation" [23], where optimizing over image rays instead of relative translations, with one benefit being bounded errors, also that it escapes other degeneracy-caused outlier issues, which would be especially important in this project context since working with sequential data. Something similar to global positioning could be used as inspiration for position retrieval in future endeavors around global trajectory alignment.

2.6 Image Retrieval

The process of searching and retrieving relevant images from a database based on a user query is referred to as image retrieval. Instead of performing an exhaustive search (all images against each other) for the feature matching in COLMAP/GLOMAP - which is a time-consuming part of the process - the command alternative `matches_importer` was employed for doing custom matching.
Here, a slightly adjusted version of the NetVLAD implementation in hloc (the hierarchical localization toolbox) [24, 25] was used, see section 3.5.3. A database of images is input to an "offline" image retrieval

[13] Note however that the reconstruction is in general no time-consuming part of the pipeline.
[14] Recall$= \frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$ and can be seen as how many relevant instances were retrieved.
[15] AUC is given by integrating the ROC (receiver operating characteristic) curve, which is given as the true positive rate against the false positive rate. Intuitively, it explains how well a model is at ranking correct answers above incorrect ones, with AUC $\in [0, 1]$ and AUC $= 0.5$ corresponding to random guessing.
[16] 1. SfM suffers from scale ambiguity. 2. Known camera intrinsics are needed to get rotation and translation from the two-view geometry, otherwise risking introducing large errors. Although, this would be no issue here since having the K matrix. 3. Degeneracy (large variance along the viewing direction) for near co-linear camera motion.

step, yielding a text file containing image pairs, such as in figure 2.3, and an HDF5 (Hierarchical Data Format version 5) file[17], for quick retrieval in "online" execution. In summary, NetVLAD was used in this project to facilitate more rapid progress of the matching process.

```
662202068948.jpg 85475170711.jpg
662202068948.jpg 85475770441.jpg
662202068948.jpg 85476370172.jpg
```

Figure 2.3 Excerpt of a `pairs-netvlad.txt` output file, with left and right column images from the two different sequences.

2.6.1 NetVLAD

NetVLAD, presented in [26], is a neural network layer designed for large-scale visual place recognition, specifically in the context of image retrieval. It extends the Vector of Locally Aggregated Descriptors (VLAD)[18] [27] approach by integrating it with a convolutional neural network (CNN), enabling end-to-end learning. The following short summary is aimed for the interested reader to get a feel for how the layer works.

2.6.1.1 The NetVLAD Layer Some related former methods rely on the two following procedures:

1. Explicitly extract handcrafted local descriptors, such as SIFT.

2. A pooling step to aggregate the descriptors into a fixed-length representation (with e.g. bag-of-features (BoF)[19] or VLAD).

Apart from the obvious drawback of the modularity of this approach, these methods lack adaptability to varying environments/lighting/etc. since being learnt on the task of matching *local* image patches. Other work had explored CNNs, but merely as black-box descriptor extractors rather than learning a representation explicitly optimized for place recognition. NetVLAD instead built on fusing these ideas, where the above items were replaced with:

1. Use a CNN (left part of figure 2.4) and crop it at the last convolutional layer, viewing it as a dense descriptor extractor. It will simply be a feature map. This output of the last convolutional layer is an $H \times W \times D$ map that can be considered as a set of D-dimensional descriptors, extracted at $H \times W$ spatial locations.

2. A new pooling layer inspired by VLAD. Pools extracted descriptors into a fixed image representation. This is the NetVLAD layer (right part of figure 2.4).

Simply put, a query image I_q is used to visually search a (large) image set I_{all}, possibly geotagged[20]. The result from passing through the architecture is a fixed size vector

$$f_\theta(I_q) = NetVLAD(CNN(I_q))$$

parametrized with a set of parameters θ, learnable via backpropagation. As a similarity metric, they chose to sort the images based on the Euclidean distance as

$$d_\theta(I_i, I_j) = ||f_\theta(I_i) - f_\theta(I_j)||$$

[17] HDF5 files are named as either `.h5` or `.hdf5` and are designed for storing and organizing large, complex datasets efficiently.
[18] VLAD is a feature encoding method that aggregates local image descriptors by computing the residuals between each descriptor and its nearest k-means cluster center (visual word), then summing these residuals per cluster to form a compact global image representation.
[19] also known as bag-of-visual-words (BoVW)
[20] Their means of evaluation was based on comparing GPS measurements for Google Street View Time Machine panoramas (where imprecise locations only added, by introducing some noise into training). This is an example of a suitable way to gather appropriate data tailored for the task. The query image I_q will have an unknown location, to be found and evaluated against each image I_i in the geotagged database.

and then searching for the optimal explicit feature map f_θ. See [26] for all details, and other concepts such as soft-assignment and intra-normalization. PCA[21]-based dimensionality reduction is applied to enhance efficiency.

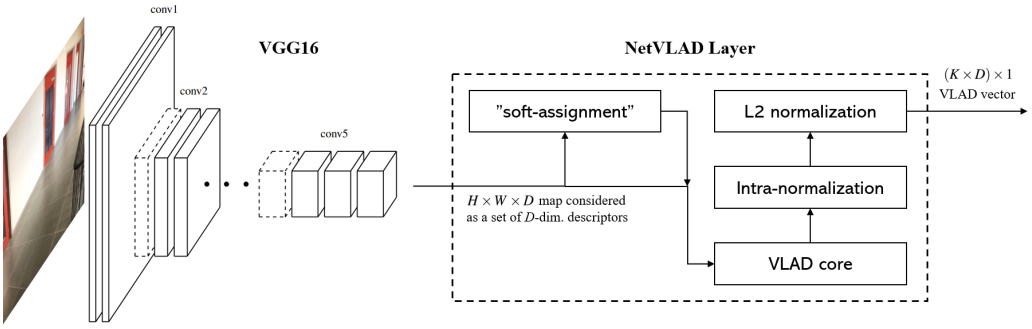


Figure 2.4 Visual of the NetVLAD layer, and how, intuitively, it is plugged into the CNN architecture. In hloc, they implement VGG16 as the CNN backbone, with PyTorch. A query image (from one of the sequences) is input to the CNN, which is cut off before any fully connected layers or softmax (and where dashed lines indicate max pooling). The output map, interpreted as a set of descriptors, is passed to the NetVLAD layer, comparing against images in the other sequence.

[21] Principal component analysis (PCA) uses SVD, where the eigenvectors are the so called *principal components* - ordered by the amount of variance they capture - that span the new, lower-dimensional and orthogonal space.

2.7 The Problem

Having presented basic theory about the main components of the project, the problem formulation is now provided under this section. It is straightforwardly given as in (2.16) by including both sequences, and their relative rotations both within and between, in a common optimization. In contrast to the view graph $\mathscr{G}_{sfm}$ above, we now only consider rotations and not complete cameras with translation included. Since starting from an initialization that has already been optimized to a certain extent, using R_g or R_P, again see sections 3.3 and 2.2.1, we can expect it to be possible to solve the problem using standard methods such as Levenberg-Marquardt. That is, since not starting from a view graph based on solely two-view geometry as in e.g. GLOMAP, with no prior knowledge about absolute rotations.

Denote $\mathscr{V}_s$ as the set of absolute rotations (vertices/nodes) of sequence $s \in \{1,2\}$. Let $Q_k \in \mathscr{V}_1$ and $S_m \in \mathscr{V}_2$, with relative rotations within each sequence (*intrasequential*) $Q_{kl} \in \mathscr{E}_1$ and $S_{mn} \in \mathscr{E}_2$. Note that when implementing the intrasequential constraints, edges are defined from lower to higher indexed nodes. The optimal relative rotations between sequences (*intersequential*) are $R_{ij} = S_j Q_i^T$, note however that they will be estimated from two-view geometry with PoseLib given a (hopefully correct) image pair from COLMAP/GLOMAP, aided by NetVLAD, where $\hat{R}_{ij} \in \mathscr{E}_{betw}$.
The different naming with Q,R,S is simply for clearer distinction. As input for the optimization, the residuals for each sequence and those representing the edges in between were stacked in a combined objective function, and solving

$$\underset{Q_1,\ldots,Q_N,S_1,\ldots,S_M}{\arg\min} \left(\alpha \left(\underbrace{\sum_{(k,l)\in\mathscr{E}_1} ||Q_{kl} - Q_l Q_k^T||^2}_{\text{Seq. 1}} + \underbrace{\sum_{(m,n)\in\mathscr{E}_2} ||S_{mn} - S_n S_m^T||^2}_{\text{Seq. 2}} \right) + \beta \underbrace{\sum_{(i,j)\in\mathscr{E}_{\text{betw}}} w_{ij} ||\hat{R}_{ij} - S_j Q_i^T||^2}_{\text{Between}} \right). \tag{2.16}$$

The parts in (2.16) representing sequences 1 and 2 will pull the solution towards the initialization, whereas the part representing the edges in between will push it towards new, adjusted solutions. This can be viewed as fine-tuning the globally aligned rotations, to independently find better agreement with the correct orientations.
In this project, a *weighted* rotation averaging was performed, namely multiplying each intersequential residual $r_i(x)$ by a weight w_{ij} (see the last summation). Additionally, the sequences are multiplied by a constant weight α[22] and β between. How these are chosen will affect whether to account more for the intra- or intersequential constraints. The nodes $\mathscr{V}_1, \mathscr{V}_2$, edges $\mathscr{E}_1, \mathscr{E}_2, \mathscr{E}_{\text{betw}}$ and weights α, β and w_{ij} make up the view graph $\mathscr{G}$, see illustration 2.5.

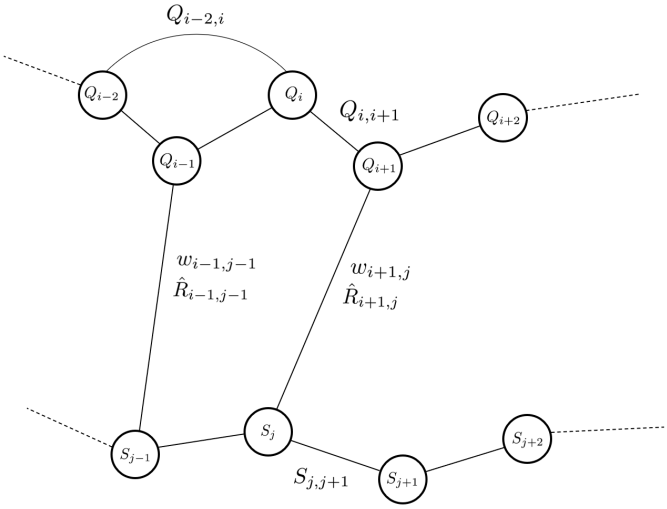


Figure 2.5 A conceptual illustration of the view graph $\mathscr{G}$. Weights α and β are omitted.

[22] Alternatively, α could be given as two separate weights, e.g. if one sequence would consist of multiple already aligned ones, thus endorsing a stronger weight assignment.

3

Implementation

3.1 Dataset

The LaMAR dataset, part of *LaMAR: Benchmarking Localization and Mapping for Augmented Reality* [28], was used. While intended for AR scenarios, it was well-suited for the problem at hand. The dataset emerged from acknowledging the need for capturing diverse environments/conditions in large-scale, addressing the benefit of data that contains structural changes over time.
Three locations were covered, CAB, HGE and LIN, out of which only the former was used, which takes place in a multi-floor office building, both indoor and outdoor and covering 1.2 ha. The scenes were captured by Microsoft HoloLens 2 and Apple iPad Pro[1] by different participants during over 100 sessions of 5 minutes each, as well as Lidar scans.

In LaMAR, they perform an automated procedure for aligning raw sensor data to a common frame, given by first aligning the lidar scans to each other using their images and point clouds. LaMAR's ground truth (GT) generation relies on finding a rigid transformation for each sequence, using techniques and tools such as NetVLAD (and other image retrieval systems).[2] Their steps are followed by a final joint optimization of all aligned poses, where both using, and using inspiration from, COLMAP. However, no specifics about the steps for creating the (highly accurate[3]) pseudo-GT, nor information about the benchmark, is provided here, instead see [28]. The GT is referred to as *map*, see plots in figure 3.1. The corresponding unaligned *raw* sequences were downloaded. These have larger image sets that include the *map* sequence's cameras, but not existing in a common coordinate frame. See figure 3.2 for an example comparing sequence pairs *map/map* and *raw/raw*. Other sequence pairs with the most overlap are shown under Appendix C (only for *map/map*).

Each *raw* sequence's folder and the one for *map* contains multiple different text files, where `trajectories.txt` (containing poses; quaternions and camera centers[4]) and `sensors.txt` (camera type, image width/height and camera intrinsics) are used here. Important to note is one distinction from COLMAP, that follows the Capture format, with sensor-to-world transformations in `trajectories.txt`, whereas LaMAR uses the Kapture format with world-to-device. See figure 3.3. Another note is that each *raw* sequence's folder is downloaded separately, and contains all images in one common subfolder. Whereas, *map* is contained in one single zip, but might have separate image subfolders for sequences. For instance, all files for the first *raw* sequence in figure 3.2 is found in the folder `ios_2021-06-02_14.48.13`, with corresponding GT *map* sequence images in `ios_2021-06-02_14.48.13_000` and `ios_2021-06-02_14.48.13_003`, although in this case it could have been natural to omit the second part due to the gap as seen in the plots.

[1] Only the iOS data were used and thus avoiding the need for local rig alignment required for the HoloLens device's four cameras.
[2] They also use e.g. HF-Net [29], SuperPoint [30] and SIFT for local features, inlier and pose consensus voting schemes, resectioning with a locally optimized RANSAC [31] and robust and Lie algebraic PGO. Ratio-of-pixels is used to compute overlap and a variant of ICP as well as bundle adjustment for refinement.
[3] They restrict to keep pose estimates that fall within a 10 cm threshold with 99.7% confidence. This implies a standard deviation of $\sigma = 3.33$cm, for a normal distribution, leading to an exclusion of only 0.8% of the frames.
[4] Note: `[tx, ty, tz]` in `trajectories.txt` refers to *camera centers*, not translations, as per the Kapture format.

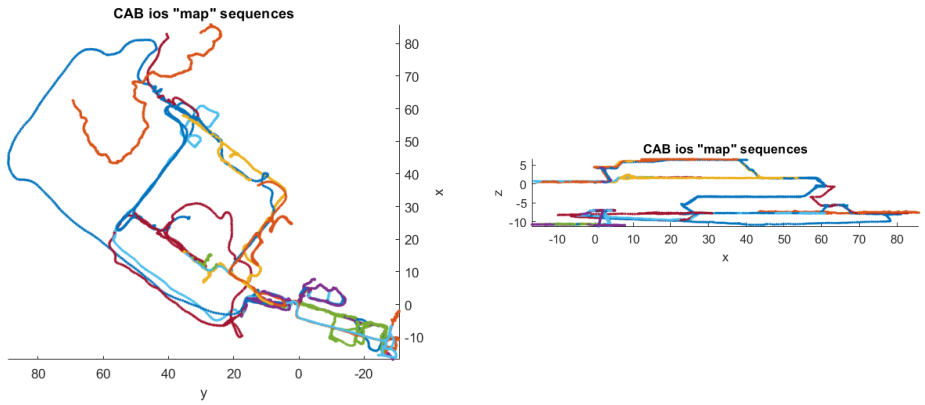


Figure 3.1 Sequences camera centers (different colouring to distinct between them) in the GT reference *map* for the CAB scene. Viewed from above (left) and the side (right). Out of the 17 sequences in this scene, 12 were deemed to provide enough overlap. In addition, the top sequence (orange) with barely any overlap was included as some means of comparison.

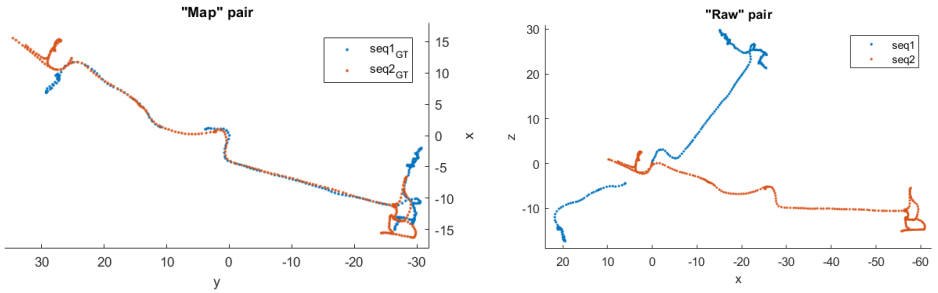


Figure 3.2 An example of overlapping *map* sequences (found at the bottom right in the left plot of figure 3.1, in cyan and orange) and its subsampled *raw* equivalences. These *raw* sequences originally consisted of $27,293$ contra $2,519$ poses, that were sampled down to match the images in *map*, with 258 and 319 respectively. Each *map* sequence's images is a subset in *raw*.

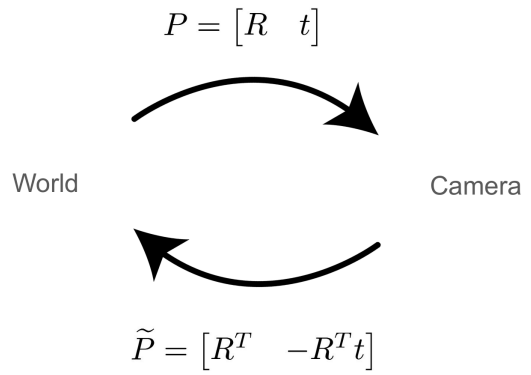


Figure 3.3 The relation between the Capture (left arrow) and Kapture (right arrow) formats.

3.2 Evaluation Metric

For the quantitative evaluation, there was needed some way of comparing the relative measures between the *raw* sequence pair contra the ground truth and avoid relying on absolute poses. This is due to the unaligned and GT sequences existing in their different coordinate systems. The error metric used in the paper *Image Matching Across Wide Baselines: From Paper to Practice* [32] was chosen since providing the relevant information needed. It is a metric is based on *angular* instead of positional errors.

For a camera pair from *raw/raw* - that is, one from each sequence - their relative rotation $\hat{R}_{ij}$ is computed. The corresponding ground truth camera pair is picked from *map/map*, computing R_{ij}. The angular difference θ_{diff} between the relative rotations is then found following the steps under 3.2.1. This is repeated for every camera pair[5] while counting the number of occurrences within a certain threshold, $\theta_{\text{diff}} < \theta$ and denote the fraction over all instances as Accuracy(θ). The *mean Average Accuracy* (mAA) is found by repeating the above procedure for different thresholds θ up to some maximum threshold θ_{max}, integrating the curve it yields. That is,

$$\text{mAA@}\,\theta_{\text{max}} = \frac{1}{\theta_{\text{max}}} \int_0^{\theta_{\text{max}}} \text{Accuracy}(\theta) d\theta.$$

Integrating, instead of choosing one fixed threshold, will favor methods with superior accuracy for lower thresholds, which might otherwise go unnoticed, deeming it on par with other, inferior methods. In line with [32], we set $\theta_{\text{max}} = 10^\circ$, while also exploring other values, yet noting that large angular thresholds should clearly be avoided.

3.2.1 Computing θ_{diff}

The corresponding estimated and GT relative rotations were initially transformed to relative quaternions $\hat{q}$ and q. The loss, or difference, between them was computed as

$$q_{\text{diff}} = \max(\varepsilon, (1 - \hat{q} \cdot q)^2)$$

with some small $\varepsilon > 0$ for numerical reasons in the implementation, and finally

$$\theta_{\text{diff}} = \arccos(1 - 2q_{\text{diff}}).$$

Using rotations matrices instead, the angle is given as

$$\theta_{\text{diff}} = \arccos\left(\frac{\text{tr}\left(\hat{R}^T R\right) - 1}{2}\right).$$

3.3 Global Rotation

As mentioned, for the optimization to work properly and having a chance at finding a correct minimum, it needs a reasonable initialization. Other than Procrustes alignment, which uses 3D scene points, the following method relies on camera poses from LaMAR and relative rotations estimated using PoseLib.

For this, the first sequence is aligned to the second, with cameras $P_j^2 = \begin{bmatrix} R_j^2 & t_j^2 \end{bmatrix}$, using a single so called global rotation R_g, by applying the homography

$$H = \begin{bmatrix} R_g & t_g \\ 0 & 1 \end{bmatrix}$$

to the cameras $P_i^1 = \begin{bmatrix} R_i^1 & t_i^1 \end{bmatrix}$, superscript denoting coordinate system. We have that,

$$R_i^2 = R_i^1 R_g$$

and with the relative rotation $\hat{R}_{ij}$ (estimated from PoseLib, see section 3.5.1) we get

$$R_j^2 = \hat{R}_{ij} \left(R_i^1 R_g\right).$$

[5] in contrast to [32] that only consider co-visible image pairs

The optimal R_g^* is found by doing the above for all camera pairs and solving

$$\arg\min_{R_g} \sum_{(i,j)\in\mathscr{E}} ||R_j^2 - \hat{R}_{ij}\left(R_i^1 R_g\right)||^2. \tag{3.1}$$

For simplification, it was only approximated by computing

$$R_{g,ij} = \left(\hat{R}_{ij} R_i^1\right)^T R_j^2, \quad \forall (i,j)$$

and finding an R_g as their average. Alternatively one could solve (3.1) using e.g. SVD, similarly as for Procrustes.

3.4 Implementation Pipeline

The following steps provide an overview of the project pipeline. A simplified visual is shown below in figure 3.4.

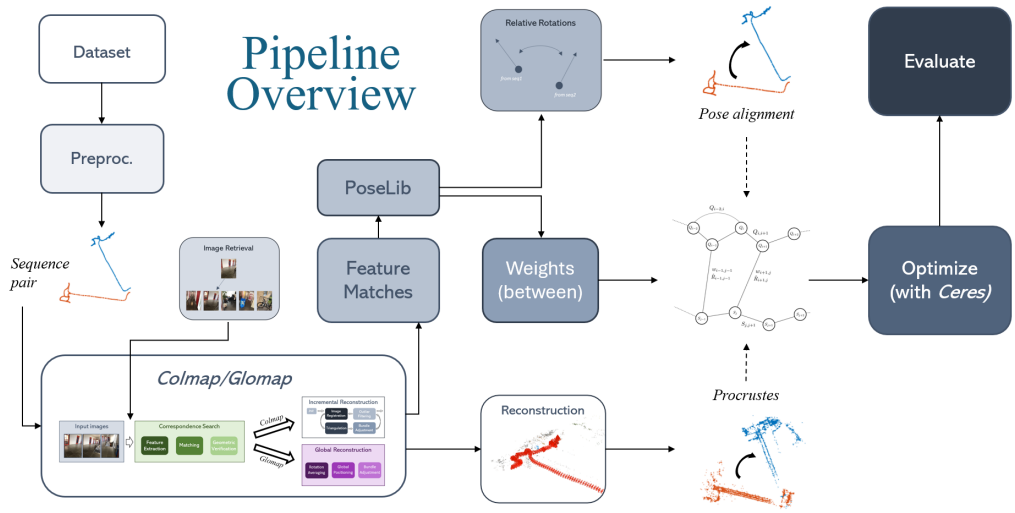


Figure 3.4 Illustrative description of the project pipeline. The process starts by extracting sequence pairs deemed to have enough overlap, that are then input to the following main steps: 1. Perform image retrieval with NetVLAD to obtain camera pairs between sequences. The aim of this is to accelerate the matching in COLMAP/GLOMAP by using *custom* instead of *exhaustive* matching. 2. Given the feature matches' keypoints of these camera pairs, PoseLib is used to compute $\hat{R}_{ij}$ and w_{ij}. 3. The edges and nodes (initially aligned using either the global alignment or rotation from Procrustes) are input to Ceres, which solves using Levenberg-Marquardt. 4. The resulting absolute rotations are input to the mAA evaluation - against GT orientations.

Pick out sequence pairs Find two somewhat overlapping sequences (full or subsampled) from the LaMAR *map* trajectories.[6] This only had to be done once. Retrieve their corresponding *raw* sequences. Note that subsampling is due to the relation in number of images, there may be over hundred times more for *raw*, causing unnecessary memory allocation since in the end aiming to compare only those with existing ground truth.

`database_seq1.db` and `database_seq2.db` For each *raw* sequence, create a database file using the following two commands[7] in COLMAP or GLOMAP:

1. `feature_extractor`
   `--database_path path/to/db`
   `--image_path path/to/images`

2. `sequential_matcher`
   `--database_path path/to/db`

This will perform the feature extraction and matching as described under section 2.5.1, storing keypoints, descriptors and image matches.

Pointcloud [8] Create databases also for *map* as in the step above.[9] For each of the four sequences (following the specified format[10]): create files `cameras.txt` (containing intrinsics) and

[6] For each sequence do the following: extract a list of all other sequences' camera centers, then for each camera center in the sequence, find the closest among the other. The sequence with most number of closest cameras (within a certain distance) becomes the pair.

[7] See the documentation for using COLMAP in command-line interface for more information: `https://colmap.github.io/cli.html`.

[8] In this, a point cloud map is not strictly needed other than for Procrustes since working with pose rotations to find the transformation T. However, the eventual goal would be to finally merge the maps using T.

[9] Note that feature extraction might have to be performed multiple times for *map*, depending on how many image folders that corresponds to *raw*.

[10] `https://colmap.github.io/format.html`.

`images.txt` (containing poses, empty lines for keypoints) and an empty `points3D.txt` file, all stored at `path/to/txt_input`. This is for keeping the original cameras when triangulating, to not get a new distorted sequence. With the database and image folder as in previous step, use the following additional command to obtain a point cloud:

3. `point_triangulator`
   `--database_path path/to/db`
   `--image_path path/to/images`
   `--input_path path/to/txt_input`
   `--output_path path/to/out`

Image Retrieval Pass the two sequences' images through NetVLAD to get the text file for image pairs between sequences (stored at `path/to/pairs_netvlad.txt` and resembling the example in figure 2.3).

Copy features to `database_between.db` To avoid the risk of feature inconsistency (and to circumvent having to specify parameter settings) the file `database_between.db` is created from `database_seq1.db` and `database_seq2.db`, containing their stacked features, using basic database manipulation in SQLite3. This was also to avoid incorrect indexing in the next step, finding 3D correspondences from 2D matches.

3D scene corr. from 2D matches (only for Procrustes) In order to perform the Procrustes alignment, one needs at least a subset of scene point correspondences between the two point clouds (*raw* to *raw* or *map* to *map*). Using feature matches between image pairs in `database_between.db` the following was done for each keypoint pair x_i, y_i in each image pair `(im1,im2)`: if both x_i and y_i triangulates to any 3D scene points X_i and Y_i respectively, these will constitute a correspondence X_i, Y_i.

Custom matching Using the NetVLAD image pairs, the feature matches were found in COLMAP/GLOMAP by:

```
matches_importer
--database_path path/to/database_between.db
--match_list_path path/to/pairs_netvlad.txt
```

This will make so that only the listed image pairs will be considered for the feature matching. That is, here is where the benefit of NetVLAD is used.

Relative poses With the feature matches between image pairs stored in `database_between.db` from previous step, PoseLib, see section 3.5.1, was used to find $\hat{R}_{ij}$ between these cameras, as well as weights (as inlier counts/ratios).

Initialization Find global and Procrustes rotations R_g and R_P, see sections 3.3 and 2.2.1 respectively. Note that RANSAC is used when implementing both methods, to account for outliers. The aligned sequences using these rotations were used as either baseline or initialization to the optimization.

Optimize Specify optimization settings and solve (2.16) in Ceres, see sections 3.5.2 and 3.6 for details.

Compute mAA Get a quantitative evaluation metric to be compared with the values for sequences aligned using only R_g or R_P, without optimization (baselines).

3.5 Tools

Below is very briefly described three of the libraries' relevant functions used in the project.

3.5.1 PoseLib

The function `estimate_relative_pose()` (with Python bindings) from the minimal solver library PoseLib [33] was used. As input, it takes the set of *un*calibrated keypoint coordinates for the image pair's feature matches as well as camera intrinsics used to normalize them. Among other stats, the function outputs relative rotations estimated from two-view geometry, computed in a RANSAC-based framework that uses the Sampson error[11] as inlier threshold criterion, with a Cheirality (in front of

[11] $e_{\text{Sampson}} = \frac{(\mathbf{x}_2^T E \mathbf{x}_1)^2}{||E_{12}\mathbf{x}_1||^2 + ||(E^T)_{12}\mathbf{x}_2||^2}$, with $E = [t]_\times R$ being the essential matrix and projections $\mathbf{x}_1$, $\mathbf{x}_2$ as described under Appendix A.1.

camera) check.[12] Finally, if the number of inliers exceeded five, they perform a bundle adjustment refinement, with an iterative Levenberg-Marquardt solver.
The function also outputs the number of inlier matches and inlier rate, both of which were used as edge weights w_{inl} between sequences (but mostly the rate).

3.5.2 Ceres

The optimization toolbox Ceres[13] was used for solving the minimization in (2.16). A few comments on the implementation are provided below.
The option `ceres::EigenQuaternionManifold()` was used as Ceres will then produce manifold-aware updates, thus internally perform the mapping between Lie group and algebra, avoiding having to explicitly apply exponential and logarithmic functions, as described in Appendix A.5.1. It is then important to note that Ceres uses the standard Hamiltonian quaternion convention (q_x first as in (2.4)) whereas the Eigen library defines it as in (2.3), with q_w first[14], like for the LaMAR dataset. Another important thing to note is that Ceres internally squares the residuals $r(x)$.
An iteration callback was used to log the individual node costs, and it was tested to set one node as constant, as to fix the gauge freedom and avoid drift (which however seemed not to make any difference in this case).

3.5.3 NetVLAD

For all potential image pair, the descriptors outputted by the cropped CNN are inputted to a batched matrix product.[15] This produces a matrix containing similarity scores for each candidate pair, for which each element is tested if being larger than a certain threshold, denoted `min_score` in hloc. This variable was adjusted to be used as a function parameter. Those pairs with a smaller similarity are masked as invalid, and the valid pairs with top k (here 5) highest scores were chosen. Figure 3.5 illustrate these intersequential image pairs for two different trajectories, three values of `min_score`.

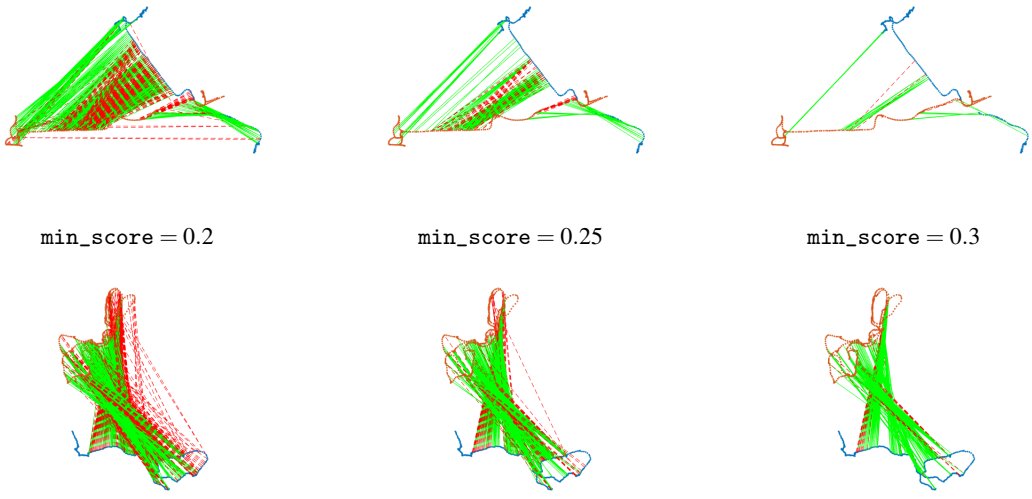


Figure 3.5 Correct (green) and incorrect (red, dashed) image pairs from NetVLAD for two sequence pairs and three different values of `min_score`. An image pair was deemed incorrect if the GT poses were more than three meters apart.

3.6 Experiments

As mentioned, the Procrustes and global rotation alignment were performed as baseline models. The aim was to extend these by allowing non-rigidity and introducing some flexibility to the model with the

[12] That is, a version of the original 5-point solver [34].
[13] Ceres is written in C++, here used with Python bindings.
[14] (although the Eigen function `q.coeff()` again gives back with q_x first)
[15] Here they use `torch.einsum()`, which can be highly useful when working with for instance batched data since being readable and generalizing easily for different kinds of batches.

purpose of being better able to match the ground truth trajectories. The experiments will be presented together with the results in next chapter, with baselines and methods as in table 3.1.

Table 3.1 Summary of evaluated methods with section references.

Name	Description	Section ref.
Baselines		
R_g	Global rotation (with RANSAC)	3.3
R_P	Procrustes (with RANSAC)	2.2.1
Experimental Methods		
w_{th}	Thresholding PoseLib inlier ratios w_{inl}	3.5.1, 4.1.2
GT edges	Using "GT" edges from *map* pair camera centres	4.1.2.1
c_{th}	Restricting weight sizes + pruning	4.1.3
k_{ext}	"Extreme" reweighing	4.1.3
Frob.	Frobenius norm, no robustifier	4.1.4
δ_H	Huber robustifier	2.4.2, 4.1.4
δ_C	Cauchy robustifier	2.4.2, 4.1.4
α/β	Common weights within/between sequences	2.7, 4.1.5
Batch	"Batch optimization"	4.5.1

4

Results

In table 4.1 are shown the sequence pairs used for evaluation. Important to note is that the mAA values vary significantly in magnitude for sequences of different length and structure. Please see Appendix B for complete tables, as only averages will be presented under the results chapter and individual metrics may offer more insight in this case. Sequence *pair4* was excluded from any averaged results, since barely covering any overlap and thus not being relevant for the task, but is included in the full tables as a comparison. See Discussion 5 for further comments about the results.

Table 4.1 The sequence pairs, with their LaMAR folder name, that were used in the following experiments. Figures are shown under Appendix C.

Sequence pair	Folder name	Figure ref.
pair0	`14.21.49` vs `17.56.28`	C.1
pair1	`14.48.13` vs `13.33.34`	3.2
pair2	`15.00.23` vs `15.01.57`	C.7
pair3	`12.16.58` vs `15.01.57`	-
*pair4**	`15.50.47` vs `17.44.49`	C.11
pair5	`14.30.40` vs `14.21.49`	C.9
pair6	`14.51.09` vs `14.07.18`	C.3
pair7	`15.01.57` vs `15.00.23`	-
pair8	`16.13.52` vs `17.56.28`	C.4

* With little to no overlap.

4.1 mAA@10

For the following mean average accuracy tests the parameters were set as presented for the *Basic* case in table 4.2 (unless the parameter itself was concerned). Results in tables 4.4-4.8 (or those in Appendix B) better than the corresponding mAA@10 without optimization, see table 4.3, are ***highlighted***.

Table 4.2 Three different parameter settings. The settings for *Ext.* (extreme weighing) and *cc* (convex combination) were chosen as they gave some of the higher mAA@10 values (although see Discussion 5 about suboptimality).

Name	w_{th}	k_{ext}	c_{th}	δ_H	δ_C	α	β
Basic	0.5	-	-	1	-	1	1
Ext.	0.8	0.01	-	-	10^{-4}	1	10^{-3}
cc	0.8	-	0.8	-	10^{-4}	1	10^{-3}

4.1.1 Baseline values

In table 4.3 are shown averages over the used sequence pairs' baseline mAA@10. That is, the rigid alignments without any further fine-tuning, provided as a means of comparison for if the optimization was able to perform any improvement beyond the initial alignment.

Table 4.3 mAA@10 for rigidly aligned sequences using rotation $R = R_g$ or R_P, no optimization. These are provided as baseline values to compare with. Although the values differ significantly for the sequences, averages (excluding *pair4*) were computed.

Rigid alignment (only)	mAA@10
R_g	0.5830
R_P	0.5784

4.1.2 Edge Assignment

Other than adding the relative rotations *between* sequences, obtained from PoseLib, to the view graph $\mathscr{G}$, it had to be determined which edges to use *within* each, since a well-connected graph is crucial. These relative rotations were computed using the sequence's absolute rotations. The sequential structure of the data was considered by adding a window in time of edges around each camera. Or alternatively, a region in space rather than time, but one would then have to be careful to avoid creating false loop closures.

As to avoid too unreliable relative rotations, the ones obtained from PoseLib with an inlier rate $w_{\text{inl}} < w_{\text{th}}$, for different threshold values, were omitted in the sense that they were replaced with the relative rotation computed using absolute rotations from the sequences, with a significantly small weight assigned to it. Average mAA@10 for the inlier rate thresholding are shown in table 4.4, with increasing values for larger w_{th}, as expected.

Table 4.4 Avg. mAA@10 (w_{th}). Sequences aligned with global or Procrustes rotation.

Inl. rate th.	mAA@10	
(w_{th})	R_g init.	R_P init.
0.2	0.1919	0.2035
0.4	0.2198	0.2368
0.6	0.4292	0.4447
0.8	0.5147	0.4964

4.1.2.1 Ground Truth Edges As a means of excluding the influence from NetVLAD and PoseLib, it was tested to use "handpicked" relative rotations from the GT *map* sequence pair. For each pose i in one of the sequences, R_{ij} was computed if a pose j from the other was within a certain distance, here 5 meters.

Note that the averages in table 4.5 are not strictly part of the results, but can act as some assessment on what an upper limit might be for the optimization itself (other parts such as NetVLAD and PoseLib excluded). The value for using R_P was in the expected range, however mAA@10 for using R_g seems rather low. Compare this with the plots in figure 4.2 which illustrate results for using the actual edges, where instead Procrustes is the worse of the two rigid alignment methods. This possibly indicate a coding bug or other error.

Table 4.5 Ground truth edges (using corresponding *map* sequence pair).

Initialization	mAA@10
R_g	0.7502
R_P	0.9622

4.1.3 Reweighing

The PoseLib inlier rates w_{inl} were manipulated in various ways such as convex combining them as

$$\sum_i w_i = 1.$$

To avoid underflow, weights under a certain threshold c_{th} were set to zero before the convex combination (which in practice essentially corresponded to pruning $\mathscr{G}$). Another test included a weight assignment as

$$w_{\text{new}} = e^{k_{\text{ext}} w_{\text{inl}}} / (4k_{\text{ext}})$$

for some k_{ext}, to obtain more of an extreme weight distribution.

Average mAA@10 for the two reweighing schemes are presented in table 4.6. Note that $c_{\text{th}} > w_{\max}$, where $w_{\max}$ is the largest (intersequential) edge weight, would mean that all edges between the sequences are removed, and no fine-tuning is done (therefore being a quite misleading case to include).

Table 4.6 Avg. mAA@10 (c_{th} & k_{ext}).

Conv. comb.	mAA@10		*Extreme*	mAA@10	
(c_{th})	R_g init.	R_P init.	(k_{ext})	R_g init.	R_P init.
0.2	0.5382	0.4593	10^{-2}	***0.5995***	0.5715
0.4	0.5382	0.4593	10^{-1}	0.5580	0.5249
0.6	0.5740	0.4806	10^{0}	0.5030	0.4726
0.8	***0.5858***	0.4945	10^{1}	3668	0.3777

4.1.4 Loss Function

The optimization was performed either with or without a robustifier, the latter case corresponding to simply using the Frobenius norm. When using either the Huber or Cauchy robustifier, the parameters δ_H and δ_C were varied, with a smaller value generally finding more outliers.
Averaged mAA@10 for using the standard loss function with Frobenius norm are shown in table 4.7 and robustifiers in table 4.8. Using a small δ_C in the Cauchy loss function gives best results in terms of mAA@10, however, this could be due to suppressing the influence from any constraints, except those where the residual is negligible. See equation (2.15).

Table 4.7 Avg. mAA@10 (Frob.).

Initialization	mAA@10
R_g	0.3692
R_P	0.3485

Table 4.8 Avg. mAA@10 (Huber and Cauchy).

Huber loss	mAA@10		*Cauchy loss*	mAA@10	
(δ_H)	R_g init.	R_P init.	(δ_C)	R_g init.	R_P init.
10^{-4}	0.4002	0.4239	10^{-4}	0.5822	0.5887
10^{-3}	0.3858	0.4011	10^{-3}	0.4529	0.5253
10^{-2}	0.3740	0.3788	10^{-2}	0.3939	0.4266
10^{-1}	0.3544	0.3530	10^{-1}	0.3771	0.3747
10^{0}	0.3482	0.3490	10^{0}	0.3467	0.3480

4.1.5 Collective Weights

The weights α and β were adjusted (separately). This was to better understand the influence of accounting most for the edges within or between sequences. Altering these constant weights yielded average mAA@10 as shown in figure 4.1.
The unexpected behaviour of initially decreasing and later increasing mAA@10 for different α (and β with Procrustes) could be investigated further, but might just be coincidental. The curves for α and β respectively were expected to somewhat mirror each other. Note that by setting both weights to their

separate best values could potentially harm the optimization, gathered from the behaviour of the R_P curve in the right plot (dashed, orange) for small β, which is again becoming worse for values of β smaller than approximately 10^{-4}.

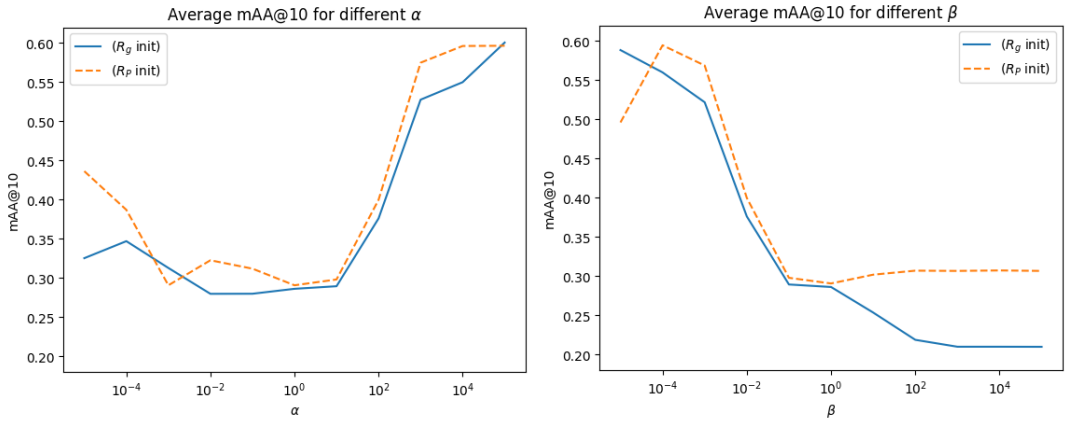


Figure 4.1 Mean average accuracy at $\theta_{\max} = 10$, averaged over the different sequence pairs, for when adjusting α (left) or β (right). Blue lines are for doing the initial rotation using the "global" method and orange, dashed lines for using the rotation from Procrustes. Note the logarithmic x-scale.

4.2 mAA@$\theta_{\max}$

See plots in figure 4.2 for average mAA over different thresholds $\theta_{\max}$. Here we see that the optimization has failed. There is a miniscule increase in mAA for using the global initialization, which holds no evidence for improvement. For Procrustes, there is instead a significant worsening of the results after optimization.

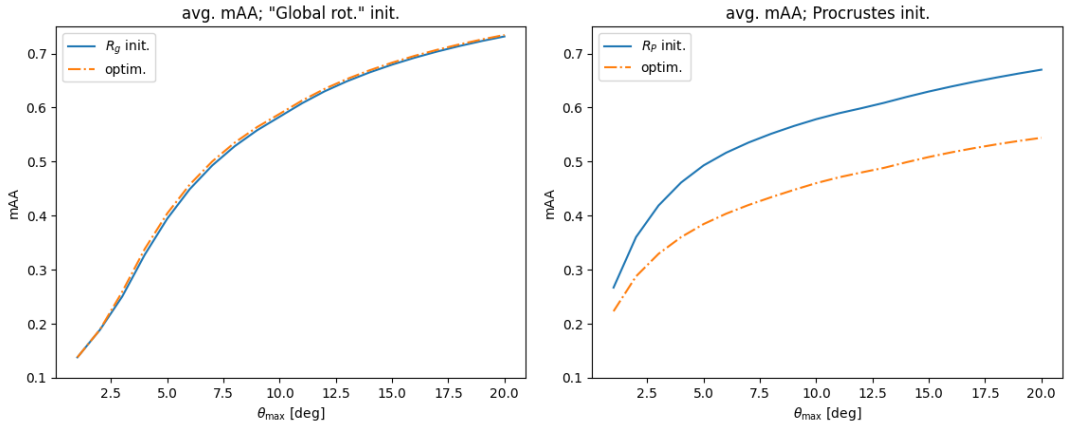


Figure 4.2 Plots of average mAA@$\theta_{\max}$ over different $\theta_{\max}$, indicating that the optimization with R_g as initialization has barely improved at all, whereas the R_P initialization has worsened significantly.

4.3 NetVLAD

Since obtaining suboptimal results for the (average) mean average accuracy, it was tested whether the issue lies within the optimization or elsewhere. To gain some understanding of how NetVLAD may influence the overall results, it was experimented with how the similarity score affects the number of image pairs (and their correctness). Due to NetVLAD providing rather varying number of image pairs depending on the choice of `min_score` (and depending on which sequence pair), see figure 3.5, a range of values were tested for each sequence pair.

Figure 4.3 shows precision[1] for the NetVLAD pairs plotted against `min_score`, averaged over the nine sequences (including *pair4*). True positives were assessed by using GT *map* poses and deeming a pair within five meters as correct.
The curve is somewhat expected; a trend between increasing precision and `min_score`. It may be a good idea to also consider the recall (and F1 score), since high precision can be achieved with very few retrieved samples, whereas recall accounts for the instances that were not retrieved.

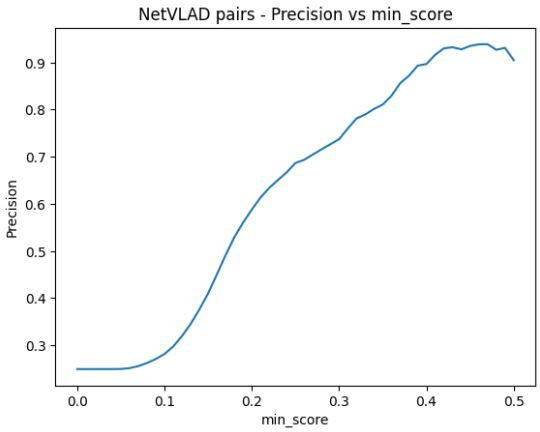


Figure 4.3 Image retrieval precision for `min_score` $\in [0, 0.5]$ (increments of 0.01), averaged over the nine sequence pairs in table 4.1. See figure 3.5 for an illustration of NetVLAD pairs for `min_score`$= 0.2$, 0.25 and 0.3, for two different sequence pairs.

4.4 PoseLib

As for NetVLAD, isolated experiments were conducted on the PoseLib outcome, separating the evaluation from the optimization. The obtained relative rotations were compared with the ones using rotations from the corresponding *map* sequences. A histogram of their angular errors and their relation to corresponding inlier ratios are shown in figure 4.4.

Most angular differences are within approximately 20°. However, there is a "clustering" of edge weights (approx. $0.3 < w < 0.6$) for this range of θ. But since any large weights ($w > 0.7$) only occur for the instances with relatively small angular difference (up to 50° but mostly within approx. 15°), the results should not be affected negatively as edges with the smaller weights can be safely thresholded away.

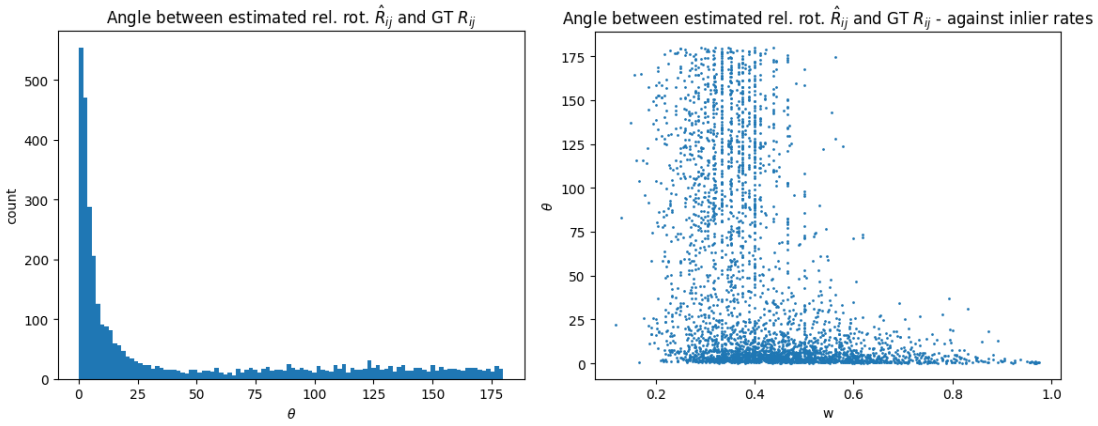


Figure 4.4 Histogram over angular differences between GT relative rotations and those estimated from PoseLib (left) and scatter plot of the angular differences against their corresponding inlier rate weights (right).

[1] Precision $= \frac{\text{True Positives}}{\text{True Positives+False Positives}}$

4.5 Qualitative Evaluation

Colormaps with angular change and accuracies of the same sequence are shown in plots in figure 4.11. Although the results are suboptimal, these plots could provide some relevant visual guide for how the optimization affected the trajectories. For instance, in figures 4.5 and 4.6 we see how the cameras have been mostly affected around the "gap" (where probably lacking edge constraints), which is the area with the least accurate result.

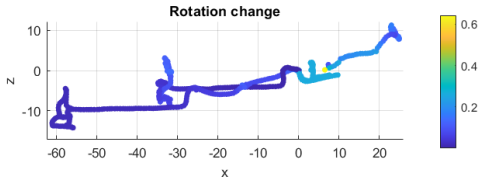


Figure 4.5 Basic param.; θ_{change}.

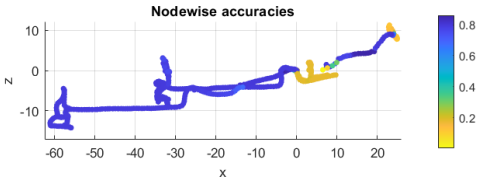


Figure 4.6 Basic param.; accuracies.

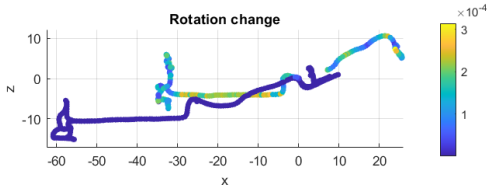


Figure 4.7 "Extreme" weights; θ_{change}.

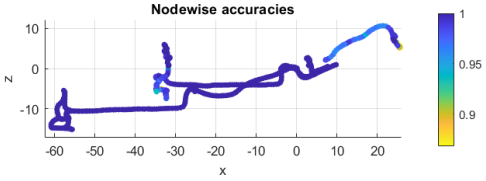


Figure 4.8 "Extreme" weights; accuracies.

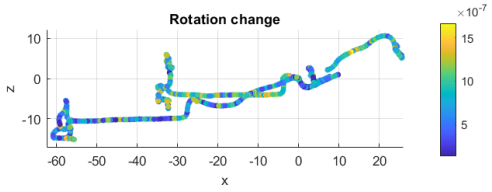


Figure 4.9 Conv. comb.; θ_{change}.

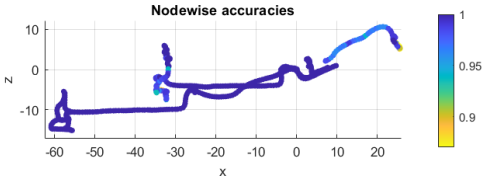


Figure 4.10 Conv. comb.; accuracies.

Figure 4.11 Example illustration of how the optimization affects the absolute rotations, for *pair1*. The rotation changes (left column) indicate the magnitude of the angle between the initial and optimized nodes. Since not providing any information about direction of the angle, whether it was good/bad, these plots should be viewed together with the nodewise accuracies (right column), computed as for mAA, and with $\theta_{\text{max}} = 10$. Three sets of parameter settings; see table 4.2. Note that there is different scaling on the colorbars for the three cases.

4.5.1 Batch Optimization

Inspired by machine learning concepts, the optimization was performed over subsets (of size B, possibly smaller for the last) of the data. It was also tested to be followed by a combined optimization of the batch solutions, which is however not included since bringing the solution back to suboptimal minima.
This method performed orders of magnitude faster than the corresponding test without dividing the graph $\mathscr{G}$ into smaller groups, but since the overall results above are suboptimal, this may or may not be relevant in any way. See figure 4.16 for colormap plots.

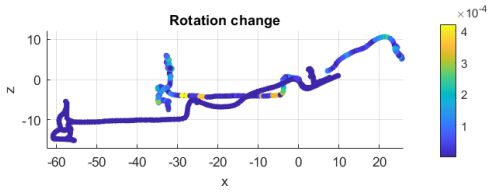


Figure 4.12 Basic param., batch; θ_{change}.

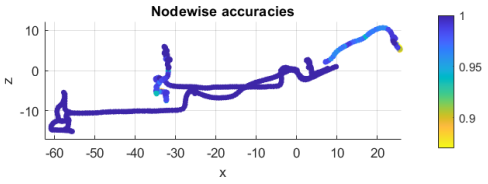


Figure 4.13 Basic param., batch; accuracies.

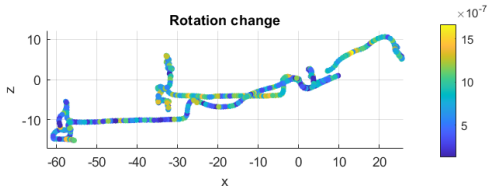


Figure 4.14 Conv. comb., batch; θ_{change}.

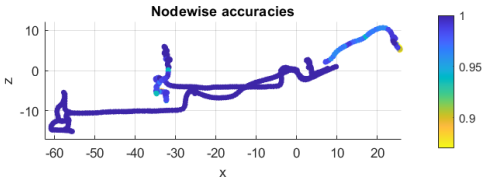


Figure 4.15 Conv. comb., batch; accuracies.

Figure 4.16 Similar plots as in figure 4.11. Note the places where the rotations have changed disproportionate to the surrounding ones, indicating failure in these places, which is most likely where the graph was split up in batches.

5

Discussion

5.1 Reflections about Results

There remains uncertainty about why the optimization was unable to perform aspirationally, and whether it stemmed down to logical issues or errors inflicted through bugs. The code is currently being scrutinized. Although some highlighted mAA@10 in the tables in Appendix B appeared as though indicating improvements, most of these however represented the absence of any intersequential edges (since some of the methods simply correspond to edge pruning, and thus giving the same mAA values). This caused the individual sequences to be "optimized", without accounting for any relative constraints between them. It could be appropriate to study the relation between results and number of included edges. A similar thing might happen when using a too small δ_C.
As expected, both NetVLAD and PoseLib seemed to work properly and looking at the suboptimal GT edges in table 4.5, which should become close to 1, this provided more evidence that the error seemed to lie within the optimization.

However, one thing to note about the PoseLib inlier ratios is that although most estimated relative rotations only differed around 0 to 20 degrees, most of their weights were rather low (between 0.3 and 0.5 on a zero to one scale), seen as the clustering in the scatter plot in figure 4.4. However, there can also be seen a trend of smaller angular differences for increasing weights, indicating that there should be no issue since there still remains good edges when thresholding off those with low weights.
Number of inliers were also used, but had to be normalized as to not weigh too much influence to the intersequential constraints, alternatively increasing α or decreasing β. If doing so, this gave similar results as for the rates.

As for the influence of α on the mAA@10, see figure 4.1 and tables B.14 and B.15, it was a bit interesting how the value evolved for the different α. For some sequences there was an improvement in mAA@10 for larger α and for some it was the opposite, while for certain sequence pairs the behaviour was as in the plot, initially a decline in performance and later an incline. But due to a similar trend as this occurring for multiple sequence pairs, the influence of α should most likely not be ruled out and it would be interesting to inspect the cause of these responses.

Figure 4.11 aims to provide a feel for how sequences are affected from optimization. Here we saw that majorly unconnected parts, such as gaps - see blue trajectory in figure 3.2 for clarification, were highly affected by the *Basic* parameter set, but near unaffected using the *Ext.* parameters and changes more spread out for the *cc* (the latter two giving the same mAA@10, but visually appearing quite differently). Although not performing any actual improving optimization, it could still be a visual guide for what regions are the most adjusted and sensitive, as a means of finding zones that need more proper tending to and how different parameter settings will affect. It could be an interesting task to further investigate, how, generally speaking, structural parts of the trajectories provide different outcome.

5.1.1 Other Comments

A subset of optimization experiments that were tested but left out of the Results chapter included perturbing the initialization by adding noise to each rotation (using axis-angle) and briefly exploring other methods of initialization such as ICP. A semi-definite programming (SDP) implementation in MATLAB

for quaternion averaging, based on [19], was looked into but was not able to yield any reasonable results.

The results in previous chapter were for the full sequences. They were also tested to be subsampled to only poses that had some overlap (a point from the other sequence within 5 meters) but this yielded slightly worse results (when one might have expected it to be the opposite). The sequences were also sampled to significantly shorter parts, of a few poses.

Although only using the *rotation* from Procrustes, it might have been a better choice to strictly use Euclidean instead of similarity transformation, as to not risk the scaling to influence incorrectly (which might have been the case sometimes, where depth of scene points were slightly inaccurate due to the near co-linear motion). However, it is not believed that this was the reason for the much lower mAA, see figure 4.2, which will be investigated more closely.

5.2 Troubleshooting

When using the rotation averaging implemented in Ceres on graphs with 3-5 nodes, both using edges with agreement, slightly perturbed and edges directly from the LaMAR sequence, the solutions converged to the correct minima. This will want to be tested more systematically as to get an understanding of what size and complexity the optimization can handle and if there is a turning-point where the problem starts to become more difficult. This should be coupled with more quantitative tests about the optimization's upper limit, isolated from NetVLAD and PoseLib, since the values for using the GT edges seemed lower than expected.

Note that convergence plots were made but not included as they were not of much assistance since minima were quickly found, usually after a few iterations, but clearly to incorrect solutions. Instead, one could potentially look at nodewise losses, obtained in Ceres using a callback function, especially for the more problematic cameras, discerned from figures 4.5-4.16. Another property one might want to study is the stability of Jacobian, both locally and after the final iteration. The condition number $\kappa(J) = ||J||_2 \cdot ||J^{-1}||_2 = \frac{\sigma_{\max}}{\sigma_{\min}}$ tells us about the sensitivity to small errors.

It would also be desired to make further tests about the dependency on the image retrieval. The sequence pairs had somewhat quite different response to different `min_score` values, as seen in figure 3.5. Note that the value was set rather low in this project as the images would later anyway go through a second "sieving" in PoseLib, but one would possibly not want too many pairs, for efficiency reasons. However, for comparison an exhaustive search was run using COLMAP for two sequence pairs, and the impact of NetVLAD (even with `min_score` $= 0$) was apparent.
Now, the precision (true positives over retrieved instances) was plotted, see figure 4.3, but it could be more interesting to also study the true/false positive *rates*, using the negatives; the left-out cases. One would then need to extract the unmasked descriptor similarity metric matrices, and could plot the ROC curve and obtain AUC, two common evaluation metrics. Another experiment could be to evaluate different `min_scores` and plot some distribution of true positives vs how far apart cameras are.

5.2.1 Other Methods to try

Other than the SDP for quaternion averaging that was tested, it was hoped to try another method known as Shonan rotation averaging [20] with official implementation available through gtsam (Georgia Tech Smoothing and Mapping). It was however not able to be built correctly on Windows and the source code would then have to be looked at more, alternatively using a Linux environment.

Other than quaternions, a Ceres implementation[1] using axis-angle minimal rotation representation in $\mathfrak{so}(3)$ was shortly tested. Additionally, GLOMAP uses as mentioned their own implementation of the **L1-IRLS** algorithm [35], which is probably the method that would be the most interesting to be tested since then potentially being able to smoothly integrate the map merging pipeline to this 3D reconstruction framework.

[1] By Chatterjee in a discussion forum, slightly altered to suit our data.

5.3 Future Work

Apart from the troubleshooting, following are some things that might want to be tried if achieving a pipeline with usable optimization.

As seen, the different parameter settings affected the outcome significantly, which may be a remaining issue even after successful rotation averaging and it could be interesting to look into if they can be interpreted or derived in some logical was, as to find some reasonable way to set their values.

As a sidenote, one might want to explore concurrency using the so called batch optimization. However, note how we may lose important edges, see yellow highlighted poses in figure 4.12, so such parallelization might critically impair the results. This could cause the resulting sequences to be "jagged", but possibly there are ways around this such as optimizing in batches multiple time (which may still be faster) or imposing constraints around the edges where the batch dividing occurs.

Finally, if successfully implementing the pipeline, one would want to extend it for including camera positions as well as the orientations. This could be done in some way inspired by e.g. the global positioning in GLOMAP or other approach.

6

Conclusion

In this project we investigated how rotation averaging could be adopted for optimizing camera orientations in a map merging framework. Sequences were initially aligned using either the rotation from Procrustes (on point clouds) or with a method using relative rotations estimated from PoseLib. These two-view geometries where then also included in the graph to be optimized, weighted using PoseLib inlier ratios between sequences. The SfM pipelines COLMAP or GLOMAP were used for feature matching and reconstruction, together with the neural network layer NetVLAD for image retrieval to evade the time-consuming exhaustive search.

Eliminating other parts of the pipeline as faulting sources, it is still unclear as to why the optimization itself caused suboptimal results, whether due to a bug or the nature of the problem. Under the Discussion in chapter 5 are presented potential later course of action, both for trying to sort out the remaining issues as well as future work after obtaining a well-working pipeline with rotation averaging for fine-tuning trajectory alignment.

Bibliography

[1] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. "maplab: An Open Framework for Research in Visual-inertial Mapping and Localization". *IEEE Robotics and Automation Letters* **3**:3 (2018), pp. 1418–1425. DOI: 10.1109/LRA.2018.2800113.

[2] A. Cramariuc, L. Bernreiter, F. Tschopp, M. Fehr, V. Reijgwart, J. Nieto, R. Siegwart, and C. Cadena. "maplab 2.0 – A Modular and Multi-Modal Mapping Framework". *IEEE Robotics and Automation Letters* **8**:2 (2023), pp. 520–527. DOI: 10.1109/LRA.2022.3227865.

[3] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. "Robust visual inertial odometry using a direct ekf-based approach". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 298–304. DOI: 10.1109/IROS.2015.7353389.

[4] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. "A tutorial on graph-based slam". *IEEE Intelligent Transportation Systems Magazine* **2**:4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[5] Z. Jiang, J. Zhu, Y. Li, J. Wang, Z. Li, and H. Lu. "Simultaneous Merging Multiple Grid Maps Using the Robust Motion Averaging". en. *Journal of Intelligent & Robotic Systems* **94**:3-4 (2019), pp. 655–668. ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-018-0895-4. URL: http://link.springer.com/10.1007/s10846-018-0895-4 (visited on 2025-03-11).

[6] A. Birk and S. Carpin. "Merging occupancy grid maps from multiple robots". *Proceedings of the IEEE* **94**:7 (2006), pp. 1384–1397. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2006.876965. URL: http://ieeexplore.ieee.org/document/1677951/ (visited on 2025-04-15).

[7] E. Ozkucur and H. L. Akin. "Cooperative multi-robot map merging using fast-slam." In: 2009, pp. 449–460.

[8] S.-H. Kim, H. Lee, and S.-H. Lee. "Improved TSDF-based map merging with Kalman filter and covariance intersection". en. *Intelligent Service Robotics* **18**:2 (2025), pp. 293–306. ISSN: 1861-2776, 1861-2784. DOI: 10.1007/s11370-025-00586-1. URL: https://link.springer.com/10.1007/s11370-025-00586-1 (visited on 2025-04-20).

[9] G. Flood, D. Gillsjö, A. Heyden, and K. Åström. "Efficient Merging of Maps and Detection of Changes". en. In: M. Felsberg et al. (Eds.). *Image Analysis*. Vol. 11482. Springer International Publishing, Cham, 2019, pp. 348–360. ISBN: 9783030202040 9783030202057. DOI: 10.1007/978-3-030-20205-7_29. URL: https://link.springer.com/10.1007/978-3-030-20205-7_29 (visited on 2025-01-30).

[10] S. Carpin. "Merging maps via hough transform". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 1878–1883. DOI: 10.1109/IROS.2008.4650616.

[11] S. Carpin. "Fast and accurate map merging for multi-robot systems". en. *Autonomous Robots* **25**:3 (2008), pp. 305–316. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-008-9097-4. URL: http://link.springer.com/10.1007/s10514-008-9097-4 (visited on 2025-04-14).

[12] P. Yin, H. Lai, S. Zhao, R. Ge, J. Zhang, H. Choset, and S. Scherer. *AutoMerge: A Framework for Map Assembling and Smoothing in City-scale Environments*. 2022. DOI: 10.48550/ARXIV.2207.06965. URL: https://arxiv.org/abs/2207.06965 (visited on 2025-03-16).

[13] G. Flood, D. Gillsjö, P. Persson, A. Heyden, and K. Åström. *Generic Merging of Structure from Motion Maps with a Low Memory Footprint*. 2021. DOI: 10.48550/ARXIV.2103.13246. URL: https://arxiv.org/abs/2103.13246 (visited on 2025-01-30).

[14] I. Arvidsson, N. C. Overgaard, K. V. Leemput, and R. Larsen. *Lecture in Medical Image Analysis; Landmark-Based Registration*. LTH, 2023.

[15] R. Hartley, J. Trumpf, Y. Dai, and H. Li. "Rotation averaging". *International Journal of Computer Vision* **103**:3 (2013), pp. 267–305. DOI: 10.1007/s11263-012-0601-0.

[16] K. Levenberg. "A method for the solution of certain nonlinear problems in least squares". *Quarterly of Applied Mathematics* **2**:2 (1944), pp. 164–168.

[17] D. W. Marquardt. "An algorithm for least-squares estimation of nonlinear parameters". *Journal of the Society for Industrial and Applied Mathematics* **11**:2 (1963), pp. 431–441.

[18] C. Olsson. *Lecture Notes in Computer Vision*. LTH, 2022.

[19] J. Fredriksson and C. Olsson. "Simultaneous Multiple Rotation Averaging Using Lagrangian Duality". In: D. Hutchison et al. (Eds.). *Computer Vision – ACCV 2012*. Vol. 7726. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 245–258. ISBN: 9783642374302 9783642374319. DOI: 10.1007/978-3-642-37431-9_19. URL: http://link.springer.com/10.1007/978-3-642-37431-9_19 (visited on 2025-02-11).

[20] F. Dellaert, D. M. Rosen, J. Wu, R. E. Mahony, and L. Carlone. "Shonan rotation averaging: global optimality by surfing $so(p)^n$". *CoRR* **abs/2008.02737** (2020). arXiv: 2008.02737. URL: https://arxiv.org/abs/2008.02737.

[21] J. L. Schonberger and J.-M. Frahm. "Structure-from-Motion Revisited". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Las Vegas, NV, USA, 2016, pp. 4104–4113. ISBN: 9781467388511. DOI: 10.1109/CVPR.2016.445. URL: http://ieeexplore.ieee.org/document/7780814/ (visited on 2024-12-04).

[22] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". en. *International Journal of Computer Vision* **60**:2 (2004), pp. 91–110. ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94. URL: http://link.springer.com/10.1023/B:VISI.0000029664.99615.94 (visited on 2025-01-30).

[23] L. Pan, D. Baráth, M. Pollefeys, and J. L. Schönberger. *Global Structure-from-Motion Revisited*. arXiv:2407.20219. 2024. DOI: 10.48550/arXiv.2407.20219. URL: http://arxiv.org/abs/2407.20219 (visited on 2025-01-09).

[24] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. *From Coarse to Fine: Robust Hierarchical Localization at Large Scale*. 2018. DOI: 10.48550/ARXIV.1812.03506. URL: https://arxiv.org/abs/1812.03506 (visited on 2025-01-22).

[25] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. *SuperGlue: Learning Feature Matching with Graph Neural Networks*. arXiv:1911.11763. 2020. DOI: 10.48550/arXiv.1911.11763. URL: http://arxiv.org/abs/1911.11763 (visited on 2025-01-22).

[26] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. *NetVLAD: CNN architecture for weakly supervised place recognition*. 2015. DOI: 10.48550/ARXIV.1511.07247. URL: https://arxiv.org/abs/1511.07247 (visited on 2025-01-09).

[27] H. Jégou, M. Douze, C. Schmid, and P. Pérez. "Aggregating local descriptors into a compact image representation". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 3304–3311. DOI: 10.1109/CVPR.2010.5540039.

[28] P.-E. Sarlin, M. Dusmanu, J. L. Schönberger, P. Speciale, L. Gruber, V. Larsson, O. Miksik, and M. Pollefeys. *LaMAR: Benchmarking Localization and Mapping for Augmented Reality*. 2022. DOI: 10.48550/ARXIV.2210.10770. URL: https://arxiv.org/abs/2210.10770 (visited on 2025-01-09).

[29] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. "From coarse to fine: robust hierarchical localization at large scale". In: *CVPR*. 2019.

[30] D. DeTone, T. Malisiewicz, and A. Rabinovich. "Superpoint: self-supervised interest point detection and description". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018, pp. 337–33712. DOI: 10.1109/CVPRW.2018.00060.

[31] O. Chum, J. Matas, and J. Kittler. "Locally Optimized RANSAC". In: G. Goos et al. (Eds.). *Pattern Recognition*. Vol. 2781. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 236–243. ISBN: 9783540408611 9783540452430. DOI: 10.1007/978-3-540-45243-0_31. URL: http://link.springer.com/10.1007/978-3-540-45243-0_31 (visited on 2025-05-01).

[32] Y. Jin, D. Mishkin, A. Mishchuk, J. Matas, P. Fua, K. M. Yi, and E. Trulls. "Image matching across wide baselines: from paper to practice" (2020). DOI: 10.48550/ARXIV.2003.01587. URL: https://arxiv.org/abs/2003.01587 (visited on 2025-04-04).

[33] V. Larsson and contributors. *PoseLib - Minimal Solvers for Camera Pose Estimation*. 2020. URL: https://github.com/vlarsson/PoseLib.

[34] D. Nistér. "An efficient solution to the five-point relative pose problem". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. IEEE, 2004, pp. 195–202. DOI: 10.1109/CVPR.2004.1315200.

[35] A. Chatterjee and V. M. Govindu. "Efficient and robust large-scale rotation averaging". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 521–528. DOI: 10.1109/ICCV.2013.70.

[36] R. Hartley, K. Aftab, and J. Trumpf. "L1 rotation averaging using the weiszfeld algorithm". In: *CVPR 2011*. 2011, pp. 3041–3048. DOI: 10.1109/CVPR.2011.5995745.

A

Appendix

A.1 Basics in Computer Vision

For readers potentially unfamiliar with Structure from Motion (SfM), or the similar robotics counterpart visual[1] SLAM (simultaneous localization and mapping), here follows a short description, as well as a quick recap of some other related computer vision concepts, based on structure and definitions of the lecture material provided for the course in *Computer Vision (FMAN95)* on LTH [18].
Being one of the main classical problems of Computer Vision, SfM revolves around the task of retrieving a 3D model (*structure*) and estimating the pose (position and orientation) of cameras (*motion*) given a set of images of the scene, captured by the cameras. The result may look like in figure 2.1.

Camera Model

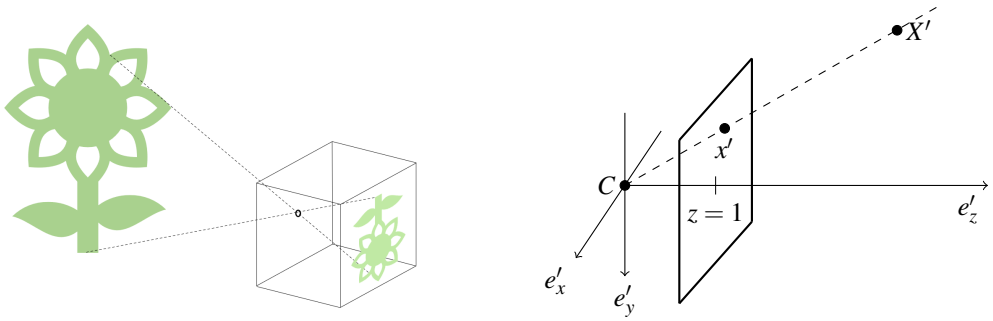


Figure A.1 Illustration of a pinhole camera (*left*) and a mathematical model (*right*) in camera coordinate system e'_x, e'_y, e'_z with camera center C, scene point X' and its projection x'. The viewing direction (or principal axis) is e'_z. The dashed line is the viewing (or image) ray.

In this project, we are exclusively working with (simple) pinhole cameras, see figure A.1 (left). In its related mathematical model, figure A.1 (right), the *image plane* is placed in front of the *camera center* $C = (0,0,0)$, in $z = 1$, to avoid the effect of images appearing upside down. Intersecting the *viewing ray* from the camera center to the 3D scene point $X' = (X'_1, X'_2, X'_3)$ in the image plane generates a projection

$$\mathbf{x} = \begin{pmatrix} X'_1/X'_3 \\ X'_2/X'_3 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix},$$

assuming $X'_3 \neq 0$.
This is depicted from the "camera's point of view", the *camera coordinate system*, however it is often useful to observe from a more general setting, the *global* (or *world*) *coordinate system*. Intuitively, there is a rotation matrix R and a translation vector t transforming the camera between the spaces. Assuming the scene point X' above has the correspondence $X = (X_1, X_2, X_3)$ in the global coordinate system, their relation is given as

[1] *Visual* since main sensors being cameras.

$$\begin{pmatrix} X_1' \\ X_2' \\ X_3' \end{pmatrix} = R \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} + t \tag{A.1}$$

where the LHS can be rewritten as

$$X_3' \begin{pmatrix} X_1'/X_3' \\ X_2'/X_3' \\ 1 \end{pmatrix}$$

to easier see that it can be interpreted as the homogeneous[2] coordinates of the scene point's projection in the image plane (embedded in $\mathbb{R}^3$). The RHS of (A.1) can be written on matrix form as

$$\begin{bmatrix} R & t \end{bmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \end{bmatrix} \mathbf{X}.$$

The K-matrix Mappings can be used to transform between world and coordinate space, as seen in equation (A.1). If dealing with an invertible map $H : \mathbb{P}^n \mapsto \mathbb{P}^n$, one talks of a *projective transformation* or a *homography*. One example of such a map is the *K-matrix* or the *intrinsic matrix*, which for a pinhole camera is on the form

$$K = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

where f_x and f_y are the focal lengths (rescaling into pixels), with $f_x = f_y$ for a *simple* pinhole camera, and (x_0, y_0) the principal point (the point which the center $(0,0,1)$ in the image in $\mathbb{R}^3$ is transformed to). The K matrix transforms the image plane embedded in $\mathbb{R}^3$ to the real image coordinate system (in pixels), see figure A.2.

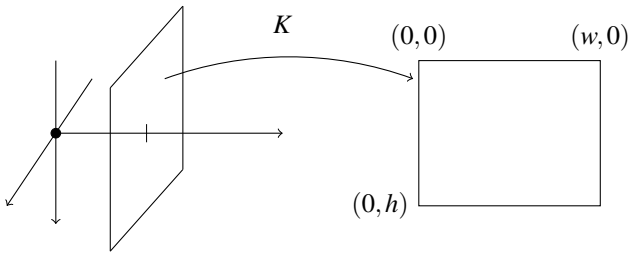


Figure A.2 The matrix K maps from the image plane to the real image, with width w and height h (pixels).

Including K in (A.1) leads to the so called camera equations

$$\lambda \mathbf{x} = P\mathbf{X}.$$

where $P = K \begin{bmatrix} R & t \end{bmatrix}$. The camera matrix $\tilde{P} = K^{-1}P = \begin{bmatrix} R & t \end{bmatrix}$ is known as the *calibrated*, or *normalized* camera. The new $\tilde{\mathbf{X}} = K^{-1}\mathbf{x}$ are the calibrated image points.
Perspective-n-Point (PnP; a resectioning method) is the process of finding cameras from a set of known scene points and their projections. The "opposite", finding scene points from (at least two) projections measured into known cameras is known as triangulation. Both can be solved using direct linear transformation (DLT)[3], useful since in practice the measurements will most likely not align precisely, such as viewing rays not triangulating to an exact scene point but instead be skew, due to noise or other uncertainties. Another common concept is bundle adjustment, which is the simultaneous optimization of

[2] A type of coordinates used in projective geometry. An element in the projective space $\mathbb{P}^n$ can be represented by its homogeneous coordinates, vectors in $\mathbb{R}^{n+1}$. For instance, the vectors $(3,2,1)$ and $(6,4,2)$ are both homogenous coordinates of an element in $\mathbb{P}^2$, which in turn can be interpreted as the point $(1,2) \in \mathbb{R}^2$, by dividing by the third coordinate.

[3] DLT is an algorithm for solving similarity ($\sim$) relations. A homogeneous linear system of equations is set up and solved by finding an approximate null space of a system matrix M.

cameras and scene points by locally adjusting the "bundle" of viewing rays to reduce the reprojection errors, and is often used as a refinement step.

A.2 Quaternion vs Matrix Representation

A.2.1 Quaternion to Rotation Matrix

Given a unit quaternion as in (2.3), the corresponding rotation matrix is

$$R = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2q_xq_y - 2q_wq_z & 2q_xq_z + 2q_wq_y \\ 2q_xq_y + 2q_wq_z & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2q_yq_z - 2q_wq_x \\ 2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

or equivalently

$$R = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_wq_z & 2q_xq_z + 2q_wq_y \\ 2q_xq_y + 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_wq_x \\ 2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix}.$$

A.2.2 Rotation Matrix to Quaternion

Given a rotation R as in equation (2.1) the quaternion can be found in two steps:

1. Find the magnitude of each quaternion component using:

$$|q_w| = \frac{\sqrt{1 + r_{11} + r_{22} + r_{33}}}{2}$$
$$|q_x| = \frac{\sqrt{1 + r_{11} - r_{22} - r_{33}}}{2}$$
$$|q_y| = \frac{\sqrt{1 - r_{11} + r_{22} - r_{33}}}{2}$$
$$|q_z| = \frac{\sqrt{1 - r_{11} - r_{22} + r_{33}}}{2}$$

2. Any given rotation matrix will have two possible quaternion representations, q and $-q$. To resolve the sign ambiguity, the q_i with largest absolute value is set as positive and used as denominator when computing the remaining components, see table A.1. (Taking the largest magnitude helps as best as possible in avoiding numerical instability and overflow).

Table A.1 Compute remaining elements in q.

$\lvert q_w \rvert$ largest	$\lvert q_x \rvert$ largest	$\lvert q_y \rvert$ largest	$\lvert q_z \rvert$ largest
$q_x = \frac{r_{32} - r_{23}}{4q_w}$	$q_w = \frac{r_{32} - r_{23}}{4q_x}$	$q_w = \frac{r_{13} - r_{31}}{4q_y}$	$q_w = \frac{r_{21} - r_{12}}{4q_z}$
$q_y = \frac{r_{13} - r_{31}}{4q_w}$	$q_y = \frac{r_{12} + r_{21}}{4q_x}$	$q_x = \frac{r_{12} + r_{21}}{4q_y}$	$q_x = \frac{r_{13} + r_{31}}{4q_z}$
$q_z = \frac{r_{21} - r_{12}}{4q_w}$	$q_z = \frac{r_{13} + r_{31}}{4q_x}$	$q_z = \frac{r_{23} + r_{32}}{4q_y}$	$q_y = \frac{r_{23} + r_{32}}{4q_z}$

A.3 Procrustes Transformation - Derivation

First, focus only on solving (2.6) for the rotation. We expand a simplified least squares sum

$$\begin{aligned}\mathscr{D}(R) &= \sum_{i=1}^{N} ||\tilde{Y}_i - R\tilde{X}_i||^2 \\ &= \sum_{i=1}^{N} \left(||\tilde{Y}_i||^2 - 2\tilde{X}_i^T R^T \tilde{Y}_i + ||\tilde{X}_i||^2\right) \\ &= \sum_{i=1}^{N} \left(||\tilde{Y}_i||^2 + ||\tilde{X}_i||^2\right) - 2\sum_{i=1}^{N} \mathrm{tr}\left(R^T \tilde{Y}_i \tilde{X}_i^T\right) \\ &= \{\mathrm{tr}(A) + \mathrm{tr}(B) = \mathrm{tr}(A+B)\} \\ &= \sum_{i=1}^{N} \left(||\tilde{Y}_i||^2 + ||\tilde{X}_i||^2\right) - 2\mathrm{tr}\left(R^T \sum_{i=1}^{N} \tilde{Y}_i \tilde{X}_i^T\right),\end{aligned}$$

where $H := \sum_{i=1}^{N} \tilde{Y}_i \tilde{X}_i^T$. Only the second term of $\mathscr{D}(R)$ is dependent of R and thus

$$\min_R \mathscr{D}(R) = \max_R \mathrm{tr}\left(R^T H\right). \tag{A.2}$$

Using that U and V are orthogonal matrices, and Σ and X^T diagonal, rewrite the objective as

$$\mathrm{tr}\left(R^T H\right) = \mathrm{tr}\left(R^T U \Sigma V^T\right) = \mathrm{tr}\left(R^T H\right) = \left\{\text{let } R = UXV^T\right\} = \mathrm{tr}\left(VX^T \Sigma V^T\right) = \{\mathrm{tr}(AB) = \mathrm{tr}(BA)\} = \mathrm{tr}(X^T \Sigma).$$

Thus, the optimization becomes

$$\max_X \mathrm{tr}\left(X^T \Sigma\right).$$

The squared singular values of a matrix are the eigenvalues of the matrix multiplied by its Hermitian conjugate. Since R is orthogonal, this means that

$$\sigma_i^2(R) = \lambda_i(R^* R) = \lambda_i(R^T R) = \lambda_i(I) = 1 \; \forall i.$$

That is, the solution to (A.2) becomes $R^* = UV^T$ or rather the solution as in (2.8) to account for the determinant by modifying the sign of the last column accordingly.
Now include the scale (but still omit the translation explicitly). At optimum, we assume that

$$\frac{d}{ds}\sum_{i=1}^{N} ||\tilde{Y}_i - sR^*\tilde{X}_i||^2 = \frac{d}{ds}\left[\sum_{i=1}^{N} \left(||\tilde{Y}_i||^2 + ||\tilde{X}||^2 - 2\tilde{Y}_i^T sR^*\tilde{X}_i\right)\right] = \sum_{i=1}^{N} \left(-2\tilde{Y}_i^T R^*\tilde{X}_i + 2s||\tilde{X}_i||^2\right) = 0$$

which gives the answer as in (2.9).
For the translation, use the optimal R^* and s^* in the objective of (2.6), take the derivative, expand and set to zero

$$\frac{d}{dt}\sum_{i=1}^{N} ||Y_i - t - s^*R^*X_i||^2 = \sum_{i=1}^{N} 2\left(Y_i - t - s^*R^*X_i\right) = 2\sum_{i=1}^{N} Y_i - 2\underbrace{\sum_{i=1}^{N} t}_{Nt} - 2\sum_{i=1}^{N} s^*R^*X_i = 0$$

which, with the centroid definition in (2.7), gives the solution as in (2.10).

A.4 Levenberg-Marquardt

With default settings, Ceres will adopt an exact step QR decomposition approach for small problems with dense Jacobian and Cholesky for larger, sparse cases. The inexact (iterative) step is followed for very large problems.

A.4.1 Exact Step

For both of the following decomposition methods, with $r = r(x_0)$ and $J = J(x_0)$, the objective in (2.14) is reformulated as

$$||r + J\Delta x||^2 + \lambda ||\Delta x||^2 = || \underbrace{\begin{bmatrix} J \\ \sqrt{\lambda} I \end{bmatrix}}_{\tilde{J}} \Delta x + \underbrace{\begin{bmatrix} r \\ 0 \end{bmatrix}}_{\tilde{r}} ||^2$$

and thus rewriting the problem as a standard least squares problem

$$\arg\min_{\Delta x} ||\tilde{J}\Delta x + \tilde{r}||^2. \tag{A.3}$$

A.4.1.1 Cholesky Decomposition From (A.3), we get the normal equations

$$\underbrace{\tilde{J}^T \tilde{J}}_{J^T J + \lambda I} \Delta x = - \underbrace{\tilde{J}^T \tilde{r}}_{J^T r}.$$

Defining the symmetric (or rather Hermitian) positive-definite matrix $A = J^T J + \lambda I$, and setting $b = -J^T r$, gives the system

$$A\Delta x = b$$

to be solved. Factorize $A = LL^T$ using Cholesky decomposition, where L is lower triangular. That is

$$L \underbrace{L^T \Delta x}_{y} = b.$$

Simply solve for y using forward substitution and finally find the step Δx from

$$L^T \Delta x = y$$

using backward substitution.

A.4.1.2 QR Decomposition Another solution to (A.3) comes by decomposing $\tilde{J} = QR$ where Q is orthogonal and R upper triangular. The objective is then given as

$$||QR\Delta x + \tilde{r}||^2 = \left\{ ||Q^T v|| = ||v|| \right\} = ||R\Delta x + Q^T \tilde{r}||^2.$$

Thus, the step is given by solving the triangular system $R\Delta x = -Q^T \tilde{r}$.

A.4.2 Iterative Solver

The following can also be combined with a Schur complement trick[4] in Ceres to reduce the system size. Expanding the objective in (2.14) gives

$$\begin{aligned}
&||r(x_0) + J(x_0)\Delta x||^2 + \lambda ||\Delta x||^2 \\
&\quad = (r(x_0) + J(x_0)\Delta x)^T (r(x_0) + J(x_0)\Delta x) + \lambda \Delta x^T \Delta x \\
&\quad = \Delta x^T J(x_0)^T J(x_0)\Delta x + r(x_0)^T J(x_0)\Delta x + \Delta x^T J(x_0)^T r(x_0) + \lambda \Delta x^T \Delta x + r(x_0)^T r(x_0) \\
&\quad = \Delta x^T \left(J(x_0)^T J(x_0) + \lambda I \right) \Delta x + 2 r(x_0)^T J(x_0)\Delta x + r(x_0)^T r(x_0).
\end{aligned}$$

This is differentiated with respect to Δx and set to zero

$$2 \left(J(x_0)^T J(x_0) + \lambda I \right) \Delta x + 2 J(x_0)^T r(x_0) = 0$$

finally giving that

$$\Delta x = - \left(J(x_0)^T J(x_0) + \lambda I \right)^{-1} J(x_0)^T r(x_0).$$

Updating of x occurs until fulfilling a termination criterion.

[4] Dividing the Jacobian matrix into blocks, which gives reduced computational complexity and improved numerical stability, especially for large-scale vision problems.

A.5 Common Concepts for Solving Rotation Averaging

There exists many various methods for rotation averaging, depending on the structure of the view graph and what rotation convention is used. Two notions that are mentioned or indirectly applied in this project are described in short below.

A.5.1 Lie Theory

Many utilize the property of $SO(n)$ forming a Lie[5] group. It is a common concept that will most likely be encountered when reading about rotation averaging.
A *group* is a non-empty set with a binary operation (combining two elements to a third one also in the set) that satisfy the four group axioms: closure, associativity, identity and invertibility. A *Lie* group is a group that is also a differentiable manifold[6], with group operations that are smooth (infinitely differentiable; in C^∞). In the case of $SO(3)$, it can be viewed as a differentiable *Riemannian* manifold, which is a smooth manifold equipped with a so called Riemannian metric[7], which allows for a number of geometric operations such as computation of distances, angles and curvatures. (Note that a Riemannian *metric* is a different concept from the distance function of a metric space).
The Lie group $SO(3)$ has the associated Lie algebra $\mathfrak{so}(3)$, which corresponds to the tangent space at a point (3×3 matrix) of the manifold. An advantage of $\mathfrak{so}(3)$ is that it has none of the constraints in $SO(3)$ (although this comes at the cost of emerging singularities). Another very beneficial property is that it is a vector space, with all usual operations of a such, and using some tricks, this may facilitate the optimization significantly. One maps between the spaces using the exponential and logarithmic maps[8], such that

$$R = e^{[\omega]_\times} \in SO(3)$$

and

$$[\omega]_\times = \log(R) \in \mathfrak{so}(3).$$

with the skew-symmetric matrix

$$[\omega]_\times = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

for a vector $\omega \in \mathbb{R}^3$, which is the axis-angle representation.

A.5.2 L_1 norm

The method **L1-IRLS**, presented in [35], uses the fact that the L_1 (Manhattan) norm is more robust than the L_2 (Euclidean) norm, as it penalizes large residuals linearly rather than quadratically, reducing the impact of outliers. Rotation averaging is performed in the Lie algebra $\mathfrak{so}(3)$, and the solution serves as initialization for an iteratively reweighted least squares (IRLS) scheme, which updates weights based on residuals.
Similarly, the Weiszfeld algorithm [36] also leverages L_1 robustness and operates in the Lie algebra.

A.6 COLMAP - Incremental Reconstruction

See [21] for details on the following steps.

Initialization The edge in $\mathcal{G}_{\text{smf}}$ from where incremental SfM reconstruction begins is important to choose well, since the process may otherwise not be able to recover from a bad initialization. There is also a tradeoff between accuracy and runtime, based on if starting from sparse or dense areas in the view graph. The following four processes occur in a loop.

[5] Pronounced "lee" (/liː).
[6] A manifold is a space that can be complex and high-dimensional, yet locally resembling a Euclidean space.
[7] The Riemannian metric g on the manifold M is a smooth and positive-definite choice of inner product on the tangent space T_pM at each point $p \in M$.
[8] $e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}$ and $\log(R) = \frac{||\omega||}{2\sin||\omega||}(R - R^T)$ where $\omega = \arccos\left(\frac{\text{tr}(R)-1}{2}\right)$ obtained by using Rodrigues' formula.

Image Registration New images in a metric reconstruction can be registered (aligned and integrated) by solving PnP, estimating the pose of the new image using 2D-3D feature correspondences with points from already registered images.

Triangulation For triangulation COLMAP adopts a DLT approach in combination with RANSAC. Once at least two new images (from different viewpoints) has been registered and if they cover some of the same, new scene part, these new scene points can be triangulated and added to the set of reconstruction points.

Bundle Adjustment Using only image registration and triangulation, the SfM might quickly drift to a non-recoverable state. They use bundle adjustment to address this, solved using Levenberg-Marquardt with the Schur complement trick, and also dividing the viewgraph $\mathscr{G}_{\text{smf}}$ into subgroups of images with major overlap.

Outlier Filtering During the above processes, outliers are filtered out by for instance: omitting points with large reprojection errors using RANSAC, using a Cauchy loss function in the bundle adjustment optimization to weigh down these points, imposing a minimum triangulation angle, degeneracy checks, etc.

B

Sequence-Individual Results

In subsections below are presented full tables, containing mAA@10 for each sequence pair for the different test cases. Values better than the corresponding mAA@10 without optimization, see table B.1, are ***highlighted***.

Baseline values

Table B.1 mAA@10 for rigidly aligned sequences using rotation $R = R_g$ or R_P, no optimization.

Seq. / -	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
R_g	0.4759	0.6957	0.3346	0.7610	0.6184	0.7845	0.5123	0.3371	0.7629	**0.5830**
R_P	0.5012	0.7676	0.3807	0.8251	0.7583	0.9616	0.3405	0.3780	0.4727	**0.5784**

Edge Assignment

Table B.2 mAA@10 (w_{th}). Sequences aligned with global rotation R_g.

Seq. / w_{th}	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
0.2	0.2597	0.0621	0.1987	0.3669	(0.0634)	0.2191	0.0318	0.2072	0.3186	0.1919
0.4	0.2629	0.1130	0.2355	0.4204	(0.2110)	0.1118	0.0610	0.2441	0.3187	0.2198
0.6	***0.5044***	0.3881	0.2140	***0.7639***	(0.4526)	0.3930	***0.5253***	0.1566	0.4647	0.4292
0.8	***0.4775***	***0.7009***	***0.3424***	***0.7639***	(0.2503)	0.5943	***0.5144***	***0.3444***	0.6440	0.5147
1.0	***0.4775***	***0.7009***	***0.3424***	***0.7639***	(***0.6277***)	***0.7916***	***0.5144***	***0.3444***	***0.7718***	***0.5884***

Table B.3 mAA@10 (w_{th}). Sequences aligned with Procrustes rotation R_P.

Seq. / w_{th}	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
0.2	0.2579	0.0642	0.2442	0.3456	(0.0749)	0.2498	0.0252	0.2516	0.3179	0.2035
0.4	0.2698	0.1247	0.2930	0.4275	(0.2533)	0.1268	0.0596	0.2584	0.3181	0.2368
0.6	***0.5956***	0.4107	0.2596	0.6451	(0.5619)	0.4044	***0.5430***	0.2315	0.3506	0.4447
0.8	***0.5012***	0.4121	***0.4045***	0.6451	(0.6678)	0.6834	0.3379	***0.4047***	0.4106	0.4964
1.0	***0.5012***	0.4121	***0.4045***	0.6451	(***0.7855***)	0.5020	0.3379	***0.4047***	***0.4727***	0.4962

"GT" Edges

Table B.4 mAA@10 (GT edges). Sequences aligned with global rotation R_g or R_P from Procrustes.

Seq. / -	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
R_g	***0.5939***	***0.9104***	***0.7261***	***0.9623***	(0.2811)	0.3173	***0.8766***	***0.7720***	***0.8430***	***0.7502***
R_P	***0.9396***	***1.0000***	***0.9229***	***0.9950***	(***0.8222***)	***0.9857***	***0.9383***	***0.9233***	***0.9926***	***0.9622***

Reweighing

Table B.5 mAA@10 (c_{th}). Sequences aligned with global rotation R_g.

Seq. / c_{th}	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
0.2	0.4655	0.6656	0.3322	***0.7637***	(0.5606)	0.7459	0.5020	0.3355	0.4723	0.5382
0.4	0.4655	0.6656	0.3322	***0.7637***	(0.5606)	0.7459	0.5020	0.3355	0.4723	0.5382
0.6	0.4702	0.6944	***0.3347***	***0.7637***	(0.6058)	0.7680	0.5123	***0.3378***	0.6790	0.5740
0.8	***0.4775***	***0.7010***	***0.3425***	***0.7637***	(0.6150)	0.7784	***0.5144***	***0.3443***	***0.7646***	***0.5858***
1.0	***0.4775***	***0.7010***	***0.3425***	***0.7637***	(***0.6277***)	***0.7916***	***0.5144***	***0.3443***	***0.7717***	***0.5883***

Table B.6 mAA@10 (c_{th}). Sequences aligned with Procrustes rotation R_P.

Seq. / c_{th}	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
0.2	0.4883	0.3912	***0.3915***	0.6451	(0.7013)	0.4535	0.3177	***0.3932***	0.3519	0.4593
0.4	0.4883	0.3912	***0.3915***	0.6451	(0.7013)	0.4535	0.3177	***0.3932***	0.3519	0.4593
0.6	0.4932	0.4084	***0.3987***	0.6451	(0.7559)	0.4706	0.3304	***0.3990***	0.4237	0.4806
0.8	***0.5012***	0.4121	***0.4045***	0.6451	(***0.7844***)	0.4938	0.3379	***0.4047***	0.4666	0.4945
1.0	***0.5012***	0.4121	***0.4045***	0.6451	(***0.7855***)	0.5020	0.3379	***0.4047***	***0.4727***	0.4962

Table B.7 mAA@10 (Ext.). Sequences aligned with global rotation R_g.

Seq. / k	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-2}	***0.5874***	0.6878	***0.3373***	***0.7639***	(0.6176)	***0.7917***	0.5111	***0.3414***	***0.7751***	***0.5995***
10^{-1}	***0.5882***	0.5057	0.3135	***0.7639***	(0.5858)	***0.7878***	0.4226	0.3119	0.7426	0.5580
10^{0}	***0.5873***	0.4569	0.2932	***0.7639***	(0.4174)	0.7677	0.3085	0.2952	0.6366	0.5030
10^{1}	0.2847	0.3019	0.2706	***0.7639***	(0.2838)	0.6306	0.1088	0.2720	0.3848	0.3668
10^{2}	0.2558	0.0471	0.2719	***0.7641***	(0.1368)	0.0633	0.0372	0.1785	0.3202	0.2305

Table B.8 mAA@10 (Ext.). Sequences aligned with Procrustes rotation R_P.

Seq. / k	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-2}	0.3381	0.7340	0.3737	0.8251	(***0.7892***)	***0.9623***	0.2839	0.3688	0.4684	0.5715
10^{-1}	0.3236	0.5505	0.3444	0.8251	(0.7193)	0.9494	0.2411	0.3224	0.4481	0.5249
10^{0}	0.3028	0.4907	0.3195	0.8251	(0.5067)	0.8963	0.2039	0.3044	0.4037	0.4726
10^{0}	0.2656	0.4255	0.2922	0.8251	(0.3509)	0.4842	0.1528	0.2687	0.3342	0.3777
10^{2}	0.1263	0.3905	0.3057	***0.8318***	(0.3293)	0.7705	0.1998	0.2468	0.3601	0.3956

Loss Function

Table B.9 mAA@10 (Frob.). Sequences aligned with global rotation R_g or R_P from Procrustes.

Seq. / -	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
R_g	0.2614	0.4237	0.2175	***0.7639***	(0.1728)	0.5155	0.2233	0.2063	0.3417	0.3692
R_P	0.2900	0.4472	0.2617	0.6451	(0.2176)	0.4771	0.2262	0.2502	0.3210	0.3485

Table B.10 mAA@10 (Huber). Sequences aligned with global rotation R_g.

δ_H \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-4}	0.4370	0.5119	0.2142	***0.7639***	(0.2820)	0.6161	0.2330	0.2185	0.3252	0.4002
10^{-3}	0.4135	0.4470	0.2144	***0.7639***	(0.2414)	0.6558	0.2474	0.1641	0.3251	0.3858
10^{-2}	0.2874	0.4237	0.2262	***0.7639***	(0.2405)	0.6408	0.2428	0.2119	0.3291	0.3740
10^{-1}	0.2648	0.4237	0.2176	***0.7639***	(0.2307)	0.4985	0.2237	0.2169	0.3494	0.3544
10^{0}	0.2664	0.4237	0.2175	***0.7639***	(0.1749)	0.5155	0.2233	0.2064	0.3422	0.3482

Table B.11 mAA@10 (Huber). Sequences aligned with Procrustes rotation R_P.

δ_H \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-4}	0.4525	0.5474	0.2588	0.6451	(0.3785)	0.7144	0.2401	0.2558	0.3222	0.4239
10^{-3}	0.3604	0.4742	0.2591	0.6451	(0.3115)	0.7468	0.2509	0.2403	0.3217	0.4011
10^{-2}	0.2985	0.4472	0.2742	0.6451	(0.3005)	0.6366	0.2451	0.2408	0.3210	0.3788
10^{-1}	0.2688	0.4473	0.2636	0.6451	(0.2887)	0.4635	0.2262	0.2525	0.3216	0.3530
10^{0}	0.2902	0.4473	0.2617	0.6451	(0.2217)	0.4771	0.2263	0.2502	0.3212	0.3490

Table B.12 mAA@10 (Cauchy). Sequences aligned with global rotation R_g.

δ_C \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-4}	***0.4815***	***0.7450***	***0.3413***	***0.7639***	(***0.6531***)	0.7686	***0.6302***	***0.3427***	0.5140	0.5822
10^{-3}	***0.5658***	0.6487	0.3000	***0.7639***	(0.1535)	0.7194	0.3991	0.1651	0.3609	0.4529
10^{-2}	0.2888	0.4291	0.2252	***0.7639***	(0.4416)	0.6568	0.2543	0.1599	0.3252	0.3939
10^{-1}	0.2869	0.4238	0.2266	***0.7639***	(0.2623)	0.6399	0.2248	0.2223	0.3437	0.3771
10^{0}	0.2658	0.4237	0.2172	***0.7639***	(0.1977)	0.4865	0.2234	0.1992	0.3428	0.3467

Table B.13 mAA@10 (Cauchy). Sequences aligned with Procrustes rotation R_P.

δ_C \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-4}	0.5012	0.6013	0.3769	0.6451	(***0.7814***)	0.9058	***0.6547***	***0.4064***	0.4256	0.5887
10^{-3}	***0.6318***	0.6839	0.3328	0.6451	(0.5229)	0.8388	***0.4202***	0.3155	0.3364	0.5253
10^{-2}	0.3155	0.4538	0.2756	0.6451	(0.5278)	0.7391	0.2593	0.3008	0.3220	0.4266
10^{-1}	0.2968	0.4472	0.2737	0.6451	(0.3206)	0.5851	0.2273	0.2557	0.3204	0.3747
10^{0}	0.2895	0.4472	0.2582	0.6451	(0.2505)	0.4521	0.2263	0.2419	0.3214	0.3480

Collective weights α and β

Table B.14 mAA@10 (α). Sequences aligned with global rotation R_g.

α \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-5}	0.0517	0.0524	0.3323	***0.7637***	(0.5612)	0.7458	0.0451	0.3355	0.0402	0.3253
10^{-4}	0.2659	0.1384	0.3283	***0.7637***	(0.5029)	0.7280	0.0161	0.3311	0.0483	0.3470
10^{-3}	0.2661	0.1425	0.2953	***0.7637***	(0.3212)	0.6668	0.0155	0.2965	0.0474	0.3128
10^{-2}	0.2676	0.1488	0.2140	***0.7637***	(0.2548)	0.5963	0.0136	0.2127	0.0462	0.2798
10^{-1}	0.2646	0.1760	0.2244	***0.7639***	(0.2484)	0.5654	0.0140	0.2063	0.0563	0.2799
10^{0}	0.2664	0.3065	0.2175	***0.7639***	(0.1749)	0.5155	0.0189	0.2064	0.1055	0.2862
10^{1}	0.2886	0.3181	0.2136	***0.7639***	(0.1821)	0.5130	0.0180	0.1995	0.1084	0.2895
10^{2}	0.3221	0.4514	0.2620	***0.7639***	(0.4352)	0.7230	0.0375	0.2713	0.1175	0.3760
10^{3}	***0.6337***	0.4507	0.3231	***0.7639***	(***0.6947***)	0.7785	0.4498	0.3100	0.3443	0.5276
10^{4}	***0.6345***	0.4481	0.3247	***0.7639***	(***0.6476***)	0.7760	***0.6078***	0.3124	0.4340	0.5499
10^{5}	***0.4775***	***0.7008***	***0.3425***	***0.7639***	(***0.6278***)	***0.7916***	***0.6113***	***0.3444***	***0.7718***	***0.6005***

Table B.15 mAA@10 (α). Sequences aligned with Procrustes rotation R_P.

α \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-5}	0.4575	0.3722	***0.3915***	0.6451	(0.7024)	0.4525	0.0462	***0.3932***	0.4657	0.4363
10^{-4}	0.3042	0.2713	***0.3892***	0.6451	(0.6256)	0.3972	0.0345	***0.3908***	0.4248	0.3870
10^{-3}	0.2699	0.1054	0.3689	0.6451	(0.4048)	0.2573	0.0250	0.3664	0.1732	0.2907
10^{-2}	0.2697	0.2914	0.3703	0.6451	(0.3171)	0.5110	0.0174	0.3681	0.1128	0.3226
10^{-1}	0.2858	0.3403	0.2474	0.6451	(0.3119)	0.5515	0.0159	0.2937	0.1149	0.3119
10^{0}	0.2902	0.3403	0.2617	0.6451	(0.2217)	0.4771	0.0151	0.2502	0.1151	0.2907
10^{1}	0.3019	0.3391	0.2579	0.6451	(0.2303)	0.5318	0.0185	0.2446	0.1138	0.2981
10^{2}	0.3436	0.4800	0.3145	0.6451	(0.5303)	0.8354	0.0354	0.2834	0.1300	0.3998
10^{3}	***0.6693***	0.4750	***0.3827***	0.6451	(***0.8756***)	0.9241	***0.4612***	0.3632	0.3772	0.5748
10^{4}	***0.6741***	0.4711	***0.3847***	0.6451	(***0.8067***)	0.9167	***0.6267***	0.3667	***0.4733***	0.5961
10^{5}	***0.6743***	0.4711	***0.3847***	0.6451	(***0.8056***)	0.9165	***0.6304***	0.3667	***0.4728***	0.5964

Table B.16 mAA@10 (β). Sequences aligned with global rotation R_g.

β \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-5}	***0.4775***	***0.7009***	***0.3425***	***0.7639***	(***0.6278***)	***0.7916***	***0.5145***	***0.3444***	***0.7718***	***0.5884***
10^{-4}	***0.4917***	0.6836	0.3247	***0.7639***	(***0.6477***)	0.7760	***0.6050***	0.3124	0.4342	0.5599
10^{-3}	***0.6205***	0.4507	0.3231	***0.7639***	(***0.6957***)	0.7785	0.4226	0.3100	0.3311	0.5218
10^{-2}	0.3223	0.4514	0.2620	***0.7639***	(0.4352)	0.7230	0.0392	0.2713	0.1175	0.3762
10^{-1}	0.2900	0.3181	0.2136	***0.7639***	(0.1802)	0.5130	0.0180	0.1994	0.1085	0.2894
10^{0}	0.2664	0.3065	0.2175	***0.7639***	(0.1749)	0.5155	0.0189	0.2064	0.1055	0.2862
10^{1}	0.2688	0.1760	0.2293	***0.7639***	(0.2033)	0.3512	0.0184	0.2155	0.0563	0.2536
10^{2}	0.2676	0.1488	0.2244	***0.7639***	(0.0656)	0.2272	0.0144	0.2108	0.0462	0.2188
10^{3}	0.2660	0.1425	0.2252	***0.7639***	(0.0633)	0.1913	0.0112	0.1792	0.0474	0.2100
10^{4}	0.2660	0.1419	0.2252	***0.7639***	(0.0629)	0.1910	0.0112	0.1792	0.0488	0.2100
10^{5}	0.2661	0.1415	0.2252	***0.7639***	(0.0629)	0.1909	0.0110	0.1792	0.0485	0.2099

Table B.17 mAA@10 (β). Sequences aligned with Procrustes rotation R_P.

β \ Seq.	*pair0*	*pair1*	*pair2*	*pair3*	*pair4*	*pair5*	*pair6*	*pair7*	*pair8*	***avg.***
10^{-5}	***0.5012***	0.4121	***0.4045***	0.6451	(***0.7856***)	0.5020	0.3370	***0.4047***	0.4727	0.4961
10^{-4}	***0.6611***	0.4710	***0.3843***	0.6451	(***0.8088***)	0.9170	***0.6240***	0.3667	***0.4729***	0.5945
10^{-3}	***0.6542***	0.4750	***0.3827***	0.6451	(***0.8774***)	0.9241	***0.4333***	0.3632	0.3633	0.5687
10^{-2}	0.3436	0.4800	0.3145	0.6451	(0.5304)	0.8354	0.0361	0.2834	0.1300	0.3998
10^{-1}	0.3042	0.3391	0.2579	0.6451	(0.2267)	0.5318	0.0176	0.2446	0.1139	0.2979
10^{0}	0.2902	0.3403	0.2617	0.6451	(0.2217)	0.4771	0.0151	0.2502	0.1151	0.2907
10^{1}	0.2858	0.3403	0.2735	0.6451	(0.2933)	0.4964	0.0153	0.2523	0.1154	0.3019
10^{2}	0.2865	0.3403	0.2894	0.6451	(0.2927)	0.5387	0.0171	0.2395	0.1138	0.3070
10^{3}	0.2858	0.3403	0.2892	0.6451	(0.2912)	0.5412	0.0167	0.2370	0.1135	0.3067
10^{4}	0.2852	0.3403	0.2894	0.6451	(0.2957)	0.5431	0.0167	0.2367	0.1133	0.3073
10^{5}	0.2834	0.2222	0.2886	0.6451	(0.2966)	0.5402	0.0167	0.2364	0.2306	0.3067

C

CAB Sequence Pairs

Some overlapping *map* sequence pairs from the LaMAR dataset's CAB scene are shown in figures C.1-C.9, with caption as the last six digits of the corresponding *raw* folder names, omitting "`ios`" and date. Naturally these are not all possibilities, but are some of the pairs considered having the most overlap for each sequence by counting the number of cameras that were closest to (and within a reasonable distance from) remaining sequences.

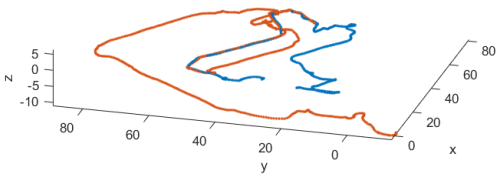


Figure C.1 `14.21.49` vs `17.56.28`

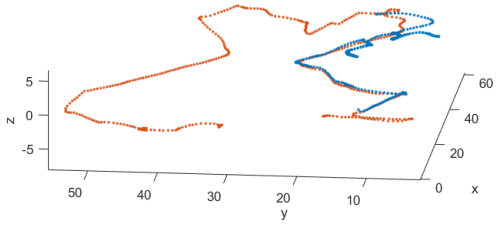


Figure C.2 `16.02.21` vs `14.21.49`

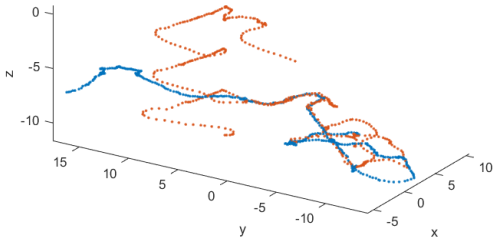


Figure C.3 `14.51.09` vs `14.07.18`

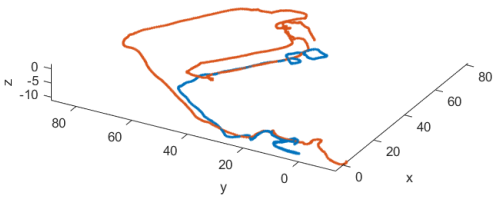


Figure C.4 `16.13.52` vs `17.56.28`

Figure C.5 *map* overlap.

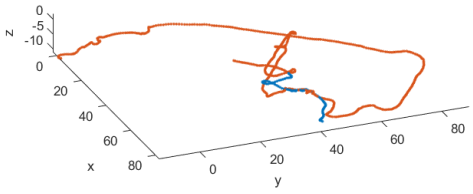


Figure C.6 `17.44.49` vs `17.56.28`

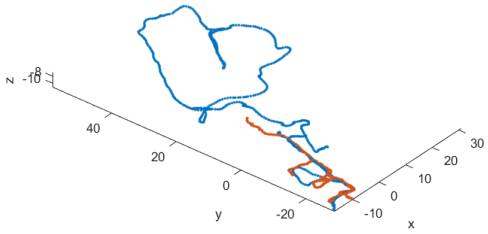


Figure C.7 `15.00.23` vs `15.01.57`

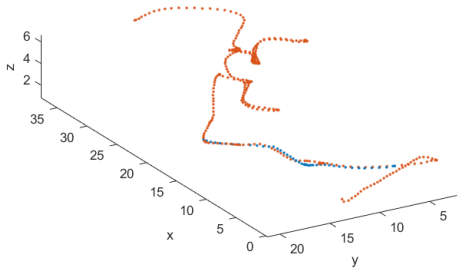


Figure C.8 `16.07.07` vs `16.02.21`

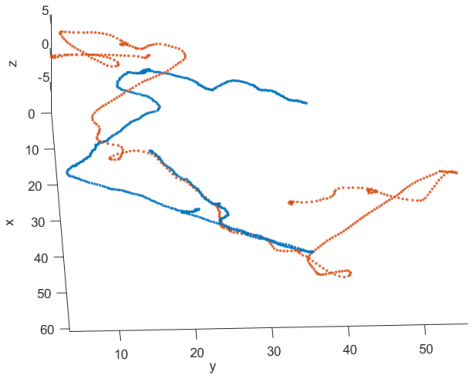


Figure C.9 `14.30.40` vs `14.21.49`

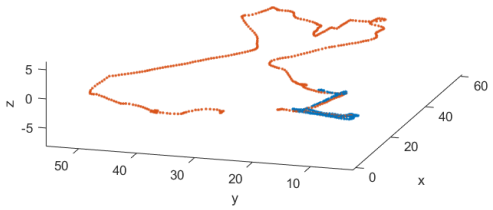


Figure C.10 `18.27.53` vs `14.21.49`

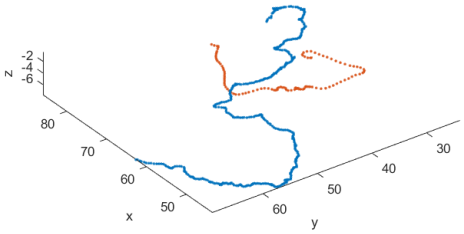


Figure C.11 `15.50.47` vs `17.44.49`

Figure C.12 *map* overlap. (The last subfigure illustrate that one of the sequences had no overlap longer than a few cameras).

Master's Theses in Mathematical Sciences 2025:E51
ISSN 1404-6342

LUTFMA-3593-2025

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/