Assessing the Efficiency of COLMAP, DROID-SLAM, and NeRF-SLAM in 3D Road Scene Reconstruction

Marcus Ascard, Farjam Movahedi

Master's thesis
2023:E35

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

3D reconstruction is a field in computer vision which has evolved rapidly as a result of the recent advancements in deep learning. As 3D reconstruction pipelines now can run in real-time, this has opened up new possibilities for teams developing Advanced Driver Assistance Systems (ADAS), which rely on the camera system of the vehicle to enhance the safety and driving experience.

This thesis presents a comparative analysis of two state-of-the-art visual SLAM pipelines, DROID-SLAM and NeRF-SLAM, and the classical Structure-from-Motion system, COLMAP. The objective was to utilize the multi-camera system on a Volvo vehicle, and public datasets, to accurately estimate trajectories and generate annotatable 3D road scenes. To assess the performance of the three methods, an evaluation pipeline was developed.

The results showed that COLMAP and DROID-SLAM can generate estimated trajectories with high accuracy when utilizing the Volvo vehicle's multi-camera system. Additionally, these systems were found to be capable of creating annotatable 3D road scenes, with some differences in quality and runtime efficiency. Generally, COLMAP demonstrated high-quality results, but its extensive runtimes makes it impractical to use at scale. The method found to be the least promising for Volvo Cars' use case was NeRF-SLAM, which failed to produce acceptable reconstructions using the multi-camera system.

Conclusively, DROID-SLAM showed the most potential for Volvo Cars' use case out of the three methods evaluated in this thesis. Despite being predominantly used off-the-shelf, it demonstrated the ability to generate impressive results with low runtimes. Nevertheless, additional research and fine-tuning is needed to optimize its performance for Volvo Cars' setup.

Acknowledgements

We wish to extend our gratitude to our supervisor Amer Mustajbasic at Volvo Cars. His industry insights, exceptional guidance, and steadfast support have been pivotal to our research. Amer's mentorship not only contributed significantly to our work, but also enriched our understanding of the field, for which we are profoundly thankful.

We also want to express our gratitude to Viktor Larsson, our supervisor at Lunds Tekniska Högskola. His academic expertise, helpful feedback, and guidance has been highly beneficial to our study.

Contents

1 Introduction **8**

1.1 Objective . . . 8

1.2 Scope . . . 9

2 Computer Vision: Theory **11**

2.1 Camera Models and Feature Processing . . . 11

2.1.1 Camera Pinhole Model . . . 11

2.1.2 Camera Intrinsics . . . 12

2.1.3 Transformations . . . 12

2.1.4 Fisheye Camera Model . . . 14

2.1.5 Feature Extraction and Matching . . . 15

2.2 The Structure-from-Motion Algorithm . . . 16

2.2.1 The Classical SfM Problem . . . 16

2.2.2 The Resection Problem . . . 17

2.2.3 Triangulation . . . 18

2.2.4 COLMAP: An Incremental SfM problem . . . 19

2.2.5 Visual SLAM . . . 20

2.2.6 DROID-SLAM . . . 21

2.3 NeRF: Neural Radiance Fields . . . 22

2.3.1 Instant-NGP: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding . . . 22

2.3.2 NeRF-SLAM . . . 23

3 Method **25**

3.1 Preparation of Datasets . . . 25

3.1.1 TartanAir . . . 25

3.1.2 The KITTI Vision Benchmark Suite . . . 26

3.1.3 Volvo Cars Data . . . 30

3.2 Evaluation Pipeline . . . 32

3.2.1 Camera Pose Evaluation . . . 32

3.2.1.1 Pose Evaluation Metrics . . . 34

3.2.2 3D Reconstruction Evaluation . . . 35

3.2.2.1 General Case of Reconstruction Evaluation . . . 35

3.2.2.2 Point Cloud Evaluation Metrics . . . 39

3.3 Visual SLAM Method Selection . . . 40

3.4 Experimental Setup . . . 41

3.4.1 COLMAP 41
3.4.2 DROID-SLAM 42
3.4.3 NeRF-SLAM 43
3.5 Mesh Generation from Multi-Camera Point Clouds 44

4 Results and Discussion **46**
4.1 TartanAir 46
4.2 KITTI Visual Odometry / SLAM Evaluation 2012 52
4.3 Volvo Cars Data 1: Single Camera 58
4.4 Volvo Cars Data 2: Multi-Camera System 63
4.5 NeRF-SLAM Limitations 69

5 Conclusions 70

Appendices 74

A SfM Ambiguity with Camera Intrinsics 74

B COLMAP Automatic Reconstruction and Output Files 75

1 Introduction

Volvo Cars is a global automotive manufacturer based in Sweden, renowned for its commitment to safety, innovation, and high-quality craftsmanship. As part of its commitment to safety, Volvo Cars has been at the forefront of developing Advanced Driver Assistance Systems (ADAS) that aim to enhance driver awareness, improve vehicle control, and mitigate potential risks on the road. Some examples of ADAS are the emergency auto-brake system and the lane-keeping aid. ADAS functions are conditioned on proper vehicle localization and perception of the surrounding environment. For that purpose, a vehicle is equipped with multiple sensors like cameras, radars, ultrasonic sensors, or LiDARs.

Volvo Cars' camera system comprises multiple cameras positioned at the front and around the vehicle. Among these cameras, some are equipped with fisheye lenses that offer a horizontal field of view exceeding 180 degrees and a vertical field of view of over 150 degrees. These cameras provide image data which, in conjunction with other sensor information like LiDAR, is utilized to construct ADAS. ADAS function developers focusing on low-speed maneuvering scenarios leverage this type of wide-angle image data in their development, thereby enhancing the safety and driving experience of Volvo cars.

Significant progress in the field of visual SLAM has paved the way for the development of systems that can accurately extract additional information from camera frames, beyond just 2D, in real-time. Historically, complementary information has been acquired through the use of other sensors, such as LiDARs for distance measurements and point cloud generation. These are then annotated by adding labels and bounding boxes to identify the objects within, based on their size and type, and used to enhance ADAS functions. But with these recent advancements in the field of visual SLAM, it is now more feasible to utilize camera data to generate point clouds, thus yielding more detailed and comprehensive insights about the surrounding environment and enhancing the overall perception capabilities.

1.1 Objective

The Low-Speed Perception team at Volvo Cars combines 2D image data obtained from fisheye cameras and accompanied LiDAR information to develop ADAS functions. Looking forward, this approach presents several drawbacks in real-world driving scenarios.

Firstly, the quality of the 3D point cloud generated by the LiDAR system is heavily limited by its placement. LiDAR sensors are typically placed on the rooftop of the vehicle, which can result in the occlusion of areas close to the front and rear of the car.

Secondly, the point clouds generated by LiDAR sensors lack color information. Consequently, human-in-the-loop style annotation becomes increasingly cumbersome, due to the difficulty in distinguishing between objects. In addition, the sparsity of the LiDAR point clouds entails an additional complication to the annotation process.

However, the previously mentioned advances in visual SLAM have enabled the generation of accurate 3D information. Deep learning-based 3D reconstruction using state-of-the-art algorithms such as DROID-SLAM and NeRF-SLAM has made it possible to create dense and precise reconstructions of complex scenes in near real-time. These algorithms show great potential for Volvo Cars' use case.

In this thesis, the aim is to conduct a comparative analysis of state-of-the-art and classical 3D reconstruction algorithms in a scale-aware manner, using one or more cameras mounted on vehicles, to reconstruct 3D road scenes. The objective of the thesis is to explore the

potential of generating 3D data to enhance the ADAS functions developed by the Low-Speed Perception team at Volvo Cars.

1.2 Scope

1. **Acquire and Prepare Data:** Collect and preprocess data to ensure the data is in a suitable format for 3D reconstruction.
2. **Develop Evaluation Protocol:** Create a standardized method for measuring the performance of different algorithms.
3. **Generate 3D Reconstructions:** Apply predominantly off-the-shelf implementations of COLMAP, DROID-SLAM, and NeRF-SLAM, to produce 3D reconstructions.
4. **Assess Reconstruction Quality:** Evaluate the quality of the 3D reconstruction results using the established evaluation protocol.
5. **Experiment with Camera Systems:** Explore the effectiveness of using single or multi-camera systems.
6. **Identify Challenges and Solutions:** Identify any remaining challenges and propose possible solutions to overcome them.

2 Computer Vision: Theory

Computer Vision is a relatively newly emerged field as a branch of computer science. The main goal of this branch is to get a sense of understanding of the world from pure visual data, most commonly images, and as such to design algorithms that can interpret these images or video, recognize patterns or features, and finally extract all necessary information to simulate the world scene it is representing.

In our thesis, we focus on 3D reconstruction, a sub-domain of computer vision. The aim is to generate three-dimensional models of scenes from 2D images (and some other added preliminary information), where each image represents a viewpoint in the scene from a certain location and angle, without any known given relation between these captured viewpoints. This implies that the position and orientation of the captured viewpoints also need to be determined.

In this thesis, we will primarily evaluate three 3D reconstruction algorithms, namely COLMAP, DROID-SLAM, and NeRF-SLAM. We will first take an in-depth look at something known as the *Structure-from-Motion* (SfM) problem, which builds the foundation of COLMAP. We will then describe another 3D reconstruction approach known as *Simultenous Localization and Mapping* (SLAM), which builds the foundation of DROID-SLAM. Lastly, we will delve deep into a new area of 3D representation known as *Neural Radiance Fields* (NeRF), which is used for NeRF-SLAM.

We will first and foremost familiarize the reader with some basic elements of 3D reconstruction and also some key terminologies used during this thesis project.

2.1 Camera Models and Feature Processing

2.1.1 Camera Pinhole Model

One of the most commonly used camera models for capturing images or frames is what is known as a pinhole camera model. It deploys a relatively easy-to-understand mathematical concept of how cameras and images are utilized in computer vision. The pinhole camera model is based on the principle of projecting a 3D point onto a 2D image plane through a small hole, known as the pinhole. The projection creates an inverted image of the 3D point on the image plane. The pinhole camera assumes that the pinhole is the only point through which light can enter the camera. This is illustrated in Figure 1 (Left). [1]

The camera coordinate system has a pinhole or center in the origin, with the z-axis pointing to the camera's forward, the x-axis pointing to the camera's right, and the y-axis pointing downwards. There is also also a *North-East-Down* (NED) frame variant of the camera coordinate systems, where the x-axis is pointing to the camera's forward direction, the y-axis is pointing to the camera's right, and the z-axis is pointing downwards. Some 3D reconstruction pipelines such as DROID-SLAM use the NED variant.

Assume we have a 3D scene point $\mathbf{X} : (X, Y, Z)$ in camera coordinate system. The projected 2D point x : (x, y) is located at the image plane $z = 1$, on the same viewing ray from center C towards $\mathbf{X}$. Since the z-axis is the norm of the image plane (i.e. they are perpendicular), and the plane is parallel to the x, y-axis, we will end up with two similar triangles, one between Cx-line and z-axis, and the other between $C\mathbf{X}$-line and z-axis. By the properties of triangle similarity, We can obtain the 2D point x by dividing the coordinates of $\mathbf{X}$ by its third coordinate, more specifically x : $(x, y) = (zX/Z, zY/Z) = (X/Z, Y/Z)$,

where $z = 1$ in our case. See Figure 1 (Right). A similar procedure applies for the NED variant, but with x-axis as the forward viewing direction instead.

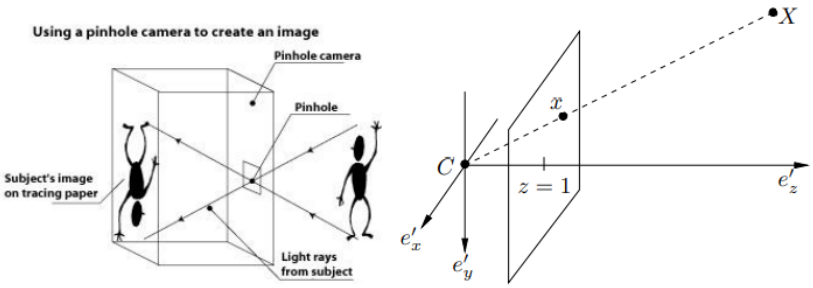


Figure 1: The Pinhole camera (left), and a mathematical model with z-forward axis (right). The image plane is located at $z = 1$, where the scene points are projected.

Usually, when a camera is mentioned, especially an RGB camera, with no other add-on, it is implied that it is a Pinhole camera model. Different RGB cameras do however introduce varying levels of distortion due to camera lenses, which need to get corrected before 3D reconstruction.

2.1.2 Camera Intrinsics

The intrinsics, or inner parameters of a camera refer to the unique internal parameters of the camera that affect how the camera captures the images and how image points are presented. These parameters include the focal length, the principal point, and the aspect ratio. The intrinsics of a camera such as the Pinhole camera model can be utilized in the SfM technique to more accurately estimate the 3D coordinates of a point from its 2D projections, where calibration puts additional constraint such that it results in a less ambiguous reconstruction. The intrinsics are also supplied to DROID-SLAM and NeRF-SLAM, although they are used differently in the 3D reconstruction pipeline.

The focal length (f_x, f_y) is the distance between the pinhole and the image plane and determines the scale of the captured images, usually expressed in units of pixels as mentioned above. The principal point (x_0, y_0) is the intersection point between the optical axis of the camera and the image plane, and it determines the location of the center of the image (normally in pixel format). See Figure 2 for illustration. The aspect ratio, γ, is the ratio of the height of pixels (if used) to their width, which in turn affects the field of view and shape of the captured images. This ratio in the pinhole camera models is typically 1 due to squared pixels. The camera intrinsics are usually represented with the following matrix:

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

2.1.3 Transformations

A mathematical problem that arises in 3D reconstruction is the fact that we have multiple viewpoints, each representing its own perspective of the 3D world. This means that

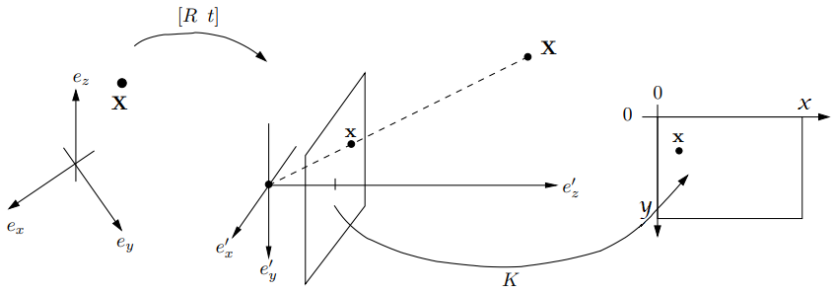


Figure 2: From 2D projection to pixel format.

each camera/image has its own coordinate system. So in order to combine and construct a unified point cloud, we need a way for the camera frames at different angles (or orientations) and positions to be located and defined in a common global coordinate system. This projection is illustrated in Figure 3.

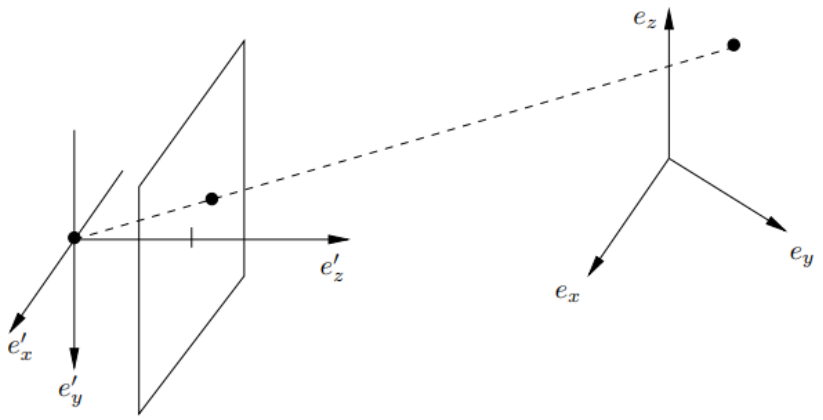


Figure 3: Projecting a scene point onto an image plane. We first need to find out the point coordinates in the camera coordinate system, given its global position.

A common way to define the orientation of a camera in the 3D world is to use a rotation matrix. More precisely, it provides information on how much rotation the axes of the camera coordinate system have with respect to the global coordinate system, around all three axes. It is an orthogonal 3×3 matrix, with the assumption that the axes in both coordinate systems consist of evenly spaced unit vectors on their axes and that the axes are perpendicular to each other. The multiplication of a rotation matrix with a vector results in the same vector being rotated around the origin of the coordinate system.

We also need to describe the position of a camera in the 3D world, that is the location with respect to the global coordinate system, or to transform the coordinates of a point from one camera coordinate system to another. This adds three new translational variables, known as the translation vector, which is a 3×1 column-vector.

The rotation and translation of a camera system in 3D space are usually fused into one 4×4 matrix, known as the transformation matrix. This matrix is used to represent the pose (position and orientation) of a camera with respect to some fixed coordinate system. In the case of 3D reconstruction, transformation matrices are used to transform the coordinates

of a point from one camera view to another, including from or to the global coordinate system. The transformation matrix is defined as:

$$T = \begin{bmatrix} sR & t \\ 0 & 1 \end{bmatrix} \tag{2}$$

Note that this is a general similarity transformation matrix, where $R \in \mathbb{R}^{3\times3}$ represents the rotation matrix, and $t \in \mathbb{R}^3$ represents the translation vector. The scaling factor s is 1 for Euclidean transformations, thus preserving the distances between points. If used as a pose transformation from/to some global coordinate system, we can also use the camera matrix notation, $P = \begin{bmatrix} R & t \end{bmatrix}$.

Quaternions are another mathematical representation of rotations in 3D space as well. They are an alternative to rotation matrices and offer several advantages, such as faster computations and avoiding singularities. A quaternion (ω, x, y, z) is a 4D vector that represents a rotation around an axis in 3D space. In 3D reconstruction, quaternions are used to represent the orientation of an object or a camera, and they can be converted to rotation matrices and vice versa. This syntax of representation is used by many off-the-shelf 3D reconstruction algorithms when they output poses, and ready-to-use datasets in their ground truth poses.

2.1.4 Fisheye Camera Model

We will introduce another camera model that was used for the Volvo Cars dataset, collectively known as radial distortion model. It is not to be confused with projective distortion during SfM reconstruction, which will be covered later. Rather it completely drops the basic property of a pinhole camera model, which is preserving 3D straight lines when projected onto 2D image planes. This characteristic, which is seen in fisheye cameras, can help capture a broader range of visual information with a wider field of view, at the expense of introducing distortion, which adds an extra layer of complexity (for undistortion preprocessing).

This distortion is, however, not so random. Rather, the projected lines are usually arched centered around some fixed point, hence the keyword "radial". It can be modeled by having projected (undistorted) points move along radial lines, that is moving points towards or away from the principal point, on all the lines that go through this principal point, at varying distances depending on how close the points are to the principal point. For illustration, see Figure 4.

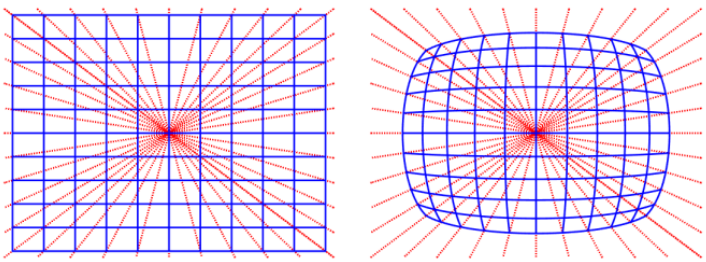


Figure 4: Radial distortion can by simulated by moving 2D points along lines intercepting the principal point. Here it is a positive radial distortion, meaning the 2D points were moved towards the principal point.

The distance that the 2D points move towards or away from the principal point depending on their location is actually an additional parameter in and of itself, which is used alongside the camera intrinsics, K, during the undistortion procedure. Given a distorted point, $\mathbf{x}_d \sim (x_d, y_d, 1)$, and an assumption of a principal point at $(0, 0, 1)$, the corresponding undistorted point $\mathbf{x}_u \sim (x_u, y_u, 1)$ is defined as:

$$\mathbf{x}_u \sim \begin{bmatrix} d(r_d)\, x_d \\ d(r_d)\, y_d \\ 1 \end{bmatrix} = \begin{bmatrix} d(r_d) & 0 & 0 \\ 0 & d(r_d) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}. \tag{3}$$

In this equation, $r_d = \sqrt{{x_d}^2 + {y_d}^2}$ is the distance between the distorted point and the principal point, and $d(r_d)$ is a function that returns a multiplier based on this distance to set the final position of $\mathbf{x}_u$ relative to the principal point. Equation (3) works since the principal point, $\mathbf{x}_d$ (for camera i and scene point j) and the corresponding $\mathbf{x}_u$ are located on the same radial line.

The function $d(r_d)$ is central to fisheye camera models since it needs to be able to effectively remove even the most severe distortion and as such there are many choices as to how it is implemented. Sometimes, an unknown radial distortion in a 2D image is complicated enough that the aforementioned distance function may not suffice, and it instead needs a distance function $d(x_d, y_d)$ that takes the x, y-coordinates as separate arguments (in contrast to the Euclidean distance r_d alone). The coefficients in the mathematical expression of $d(r_d)$ are commonly known as distortion coefficients, $k_a,\ \forall a$. One common fisheye distortion model is known as the Kannala-Brandt model, with the following polynomial expression:

$$d(\theta) = \theta \cdot (1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8), \quad \text{where} \quad \theta = \operatorname{arctan2}(r_w, z_{wc}). \tag{4}$$

In this equation, $r_w = \sqrt{{x_w}^2 + {y_w}^2}$ corresponds to the radial distance in the world coordinate system, and z_{wc} is the z-coordinate of the 3D point in camera coordinate system. We can see the parallel between r_w and r_d. Furthermore, We can also observe that in the case of homogeneous coordinate system, where the image plane is located at $z = 1$, we could reformulate $\theta = \operatorname{arctan2}(r, z)$ to the simplified expression $\theta = \arctan(r)$. This is the case for some publications and computer vision libraries such as `OpenCV`.

The undistortion process serves as an intermediary step during 3D reconstruction, more specifically after calibrating the image points, that is $\mathbf{x}_d = K^{-1}\mathbf{x}_{\text{pixel}}$, and before the 3D reconstruction algorithms used for pinhole cameras. During our comparative analysis, we seek to evaluate the performance of different 3D construction algorithms on publicly available datasets captured by pinhole cameras, but also to challenge the algorithms when supplying undistorted fisheye images from Volvo Cars. In this thesis, the provided data from Volvo Cars have already underwent an undistortion process.

2.1.5 Feature Extraction and Matching

We conclude this section by familiarizing the reader with a group of feature processing algorithms, known as feature detection and feature matching. It is a step that comes after image undistortion and before the actual 3D reconstruction algorithm.

In its simplest form. feature detection involves identifying distinctive patterns or clumps of pixels. To be specific, the extracted features may be edges, corners, key points, *blobs* (bi-

nary large objects), or a combination of them. These key points and the geometry around them are then described by a compact representation technique, known as a feature descriptor. Given an image pair with feature maps, a feature matching algorithm then finds the common feature points in the images.

Feature detection is an active area of research in computer vision, and many techniques have been presented in this regard, most notably the feature detection algorithm *FAST* (Features from Accelerated Segment Test) [2], the feature description algorithm *BRIEF* (Binary Robust Independent Elementary Features) [3], the feature detection and description algorithm *ORB* (Oriented FAST and rotated BRIEF) [4] and *SIFT* (Scale-Invariant Feature Transform) [5]. We refer the reader to these papers for further details on the intricacies of these techniques.

2.2 The Structure-from-Motion Algorithm

In this section, we first familiarize the reader with the SfM problem. For the sake of simplicity and to make the concept more comprehensible to the reader from a linear algebraic aspect, we assume that feature extraction and matching already has been conducted. We will then briefly describe the incremental SfM procedure used in COLMAP for sparse reconstruction. It is worth noting that multiple modifications, improvements and variants of the SfM approach exist today.

2.2.1 The Classical SfM Problem

The classical Sfm problem is a generic formulation of SfM, where $P = K \begin{bmatrix} R & t \end{bmatrix} = \begin{bmatrix} A & \tau \end{bmatrix}$ is a 3×4 uncalibrated camera matrix. A is not necessarily a rotation matrix. This means that the SfM approaches, including COLMAP, work even when the camera intrinsics are unknown, albeit at the expense of additional computational complexity.

Assume we are given feature points $x_{ij}, \forall i, j$, where each 2D point x_{ij} is the projected point of the 3D point j in an image i. Then, the main goal is to to determine all 3D point coordinates X_j, as well as poses such that:

$$\lambda_{ij} \mathrm{x}_{ij} = P_i \cdot \mathbf{X}_j, \quad \forall i, j, \tag{5}$$

where λ_{ij} is an unknown scalar. The RHS of Equation (5) is interpreted as rotating the current coordinate system depicting a 3D point, and then translating it to the camera coordinate system, if we know the intrinsics:

$$P_i \cdot \mathbf{X}_j = K \begin{bmatrix} R_i & t_i \end{bmatrix} \cdot \begin{bmatrix} X_j \\ 1 \end{bmatrix} = K \left(R_i X_j + t_i \right) \tag{6}$$

The result of the multiplication in Equation (6) will be another 3D point, which corresponds to X_j but in the new coordinate system. For our use case, the new 3D point is in the i:th camera coordinate system. So in order to obtain the 2D point x_{ij}, we divide it with its z-coordinate. This is what the LHS of Equation (5) represents. More specifically, the scalars λ_{ij} define the depth (z-coordinate) of the projected points x_{ij}. In SfM we are primarily interested in estimating the 3D scene points and camera extrinsics (poses), similar to other 3D reconstruction methods. However, the scalars are bi-products of the formulation in (5) and they need to be determined alongside other unknowns.

A side effect of 3D reconstruction from mere 2D data, that needs to be contained as much as possible, is projective ambiguity. The less prior information that is provided, the more ambiguity we will end up with. Camera intrinsics, which are parameters that we are provided with, is an example of a priori to combat projective ambiguity such as distorted angles in a 3D reconstructed object. To see the process of how it will remove such ambiguity in SfM, and how much reduction in overall ambiguity we can benefit from, we refer the reader to Appendix at the end of this project.

We divide the main SfM problem of 3D points and pose estimation into two subsections, which are subsequently presented to the reader. We begin with the resection problem for pose estimation,

2.2.2 The Resection Problem

We present the general formulation of the resection problem to obtain an (uncalibrated) pose $P = \begin{bmatrix} A & \tau \end{bmatrix}$. We assume that we have (some) known or well-initiated scene points with their corresponding 2D points. The problem is formulated as:

$$\lambda_j \mathrm{x}_j = P \cdot \mathbf{X}_j, \quad j = 1, \ldots, n \tag{7}$$

Similar to the camera equation (Equation (5)), λ_j is a bi-product of the resection problem that needs to be determined as well. We rewrite Equation (7) as a homogeneous linear system of equations, with zeros at one end. After gathering the variables in one side, we have the following equation system:

$$\begin{aligned} \mathbf{X}_j^\top p_j - \lambda_j x_j &= 0 \\ \mathbf{X}_j^\top p_j - \lambda_j y_j &= 0 \\ \mathbf{X}_j^\top p_j - \lambda_j &= 0, \end{aligned} \tag{8}$$

where p_1, p_2, p_3 are 4×1 vectors of rows of P. The goal of Equation (8) is to construct a system matrix and then find an approximate null space of it. Before writing the matrix form of Equation (8), we may remark the lack of solution for the resection problem for one $\mathbf{X}_j, \mathrm{x}_j$ pair, due to the fact that there are far more unknowns than equations. Note that each individual element of P is an unknown.

The minimum number of known scene points with corresponding 2D points needed for the problem to be well-defined is $n = 6$ (for the general case). This will provide a total of $6 \cdot 3 = 18$ equations, while only introducing $6 - 1 = 5$ new unknowns, which are the new scaling factors $\lambda_{j'}$. The matrix system of Equation (8) for $n \geq 6$ is:

$$\begin{bmatrix} \mathbf{X}_1 & 0 & 0 & -x_1 & 0 & 0 & \cdots \\ 0 & \mathbf{X}_1 & 0 & -y_1 & 0 & 0 & \cdots \\ 0 & 0 & \mathbf{X}_1 & -1 & 0 & 0 & \cdots \\ \mathbf{X}_2 & 0 & 0 & 0 & -x_2 & 0 & \cdots \\ 0 & \mathbf{X}_2 & 0 & 0 & -y_2 & 0 & \cdots \\ 0 & 0 & \mathbf{X}_2 & 0 & -1 & 0 & \cdots \\ \mathbf{X}_3 & 0 & 0 & 0 & 0 & -x_3 & \cdots \\ 0 & \mathbf{X}_3 & 0 & 0 & 0 & -y_3 & \cdots \\ 0 & 0 & \mathbf{X}_3 & 0 & 0 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{9}$$

This approach of constructing a large matrix system of equations is called *Direct Linear Transformation* (DLT). The matrix system in Equation (9), $M \cdot v = 0$ will not yield an exact solution (a non-zero vector v) in many cases due to noise. Instead we use a solver with scaling constraint:

$$\min_{\|v\|^2=1} \| Mv \|^2, \tag{10}$$

This solver is of course a homogeneous least squares minimizer. Since we have $v^\top (M^\top M) v$ in Equation (10), we can make use of Singular Value Decomposition (SVD). We refrain from going into any further details on how exactly it is approached.

After yielding P, we can use RQ-factorization to extract the camera intrinsics, $P = K \begin{bmatrix} R & t \end{bmatrix}$, which of course is unnecessary in our case since we do have the intrinsics. For instance, we can reduce the required number of known 2D-3D point correspondences to $n = 3$ if calibrated cameras are used. In this case, an applied formulation and matrix system of (7) is used, where the camera poses now have 6 degrees of freedom – since the left 3×3-block must be rotational. The pose estimation problem where intrinsics are provided is more commonly known as the Perspective-n-Point (PnP) problem.

2.2.3 Triangulation

In Structure-from-Motion, we seek to triangulate a scene point X from given feature-matched points (x_i, y_i) in known camera poses P_i. This problem can be formulated as:

$$\lambda_i \mathrm{x}_i = P_i \cdot \mathbf{X}, \quad i = 1, \ldots, m \tag{11}$$

We have somewhat an identical general problem formulation as in the resection problem. The three coordinates of X and the scaling factors λ_i are unknown variables. Two projections will suffice for the triangulation problem in Equation (11) to be well-defined, that is $m = 2$. Geometrically, it means to find the intersection points between the two viewing lines from 2D points to their corresponding scene point (going through the camera centers) in two-view SfM. Although in real-life application, the interference of noise will cause the viewing lines to not quite intersect each other, and in the case of $m > 2$, it is unlikely for all viewing lines to have exactly one single intersection point. This situation is illustrated by Figure 5.

This means that we need to determine a 3D point that minimizes the distance to the viewing rays. More 2D points/poses provide a more accurate estimation of the 3D point, at the cost of computational complexity. Once again we use an approximated DLT approach. The difference from the resection DLT is that the unknowns in P are swapped for X in the unknown vector v. The linear system of equations, after moving the known parameters to the M matrix, is:

$$\begin{bmatrix} P_1 & -\mathrm{x}_1 & 0 & \cdots & 0 \\ P_2 & 0 & -\mathrm{x}_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_m & 0 & 0 & \cdots & -\mathrm{x}_m \end{bmatrix} \cdot \begin{bmatrix} X \\ \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{12}$$

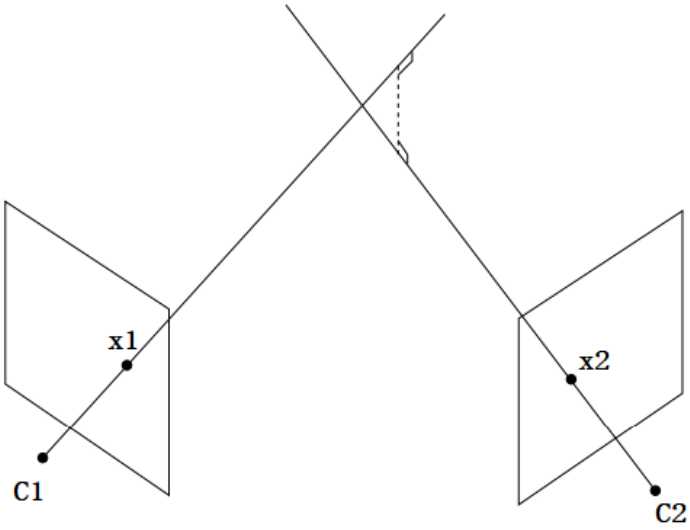


Figure 5: Triangulating a 3D point in Two-view SfM. Note that in a noisy environment, two viewing rays may not necessarily intersect each other.

We can use the similar least square minimizer as before, more specifically:

$$\min_{\|v\|^2=1} \| Mv \|^2, \tag{13}$$

which once again can be solved through the singular value decomposition of M.

2.2.4 COLMAP: An Incremental SfM problem

One of the algorithms we use in our project, is a program/algorithm called COLMAP. [6] The sparse reconstruction pipeline of COLMAP is, as stated previously, an incremental SfM-based algorithm. Alongside COLMAP, various other state-of-the-art SfM-based algorithms are also always some variation of incremental SfM techniques. In this subsection, we briefly present the general pipeline and outline only the essential aspects.

Incremental SfM shares some common characteristics of the classical SfM. It starts with feature extraction on an unordered collection of images, then matches features in image pairs by finding the most similar features in the scene overlaps. By default, SIFT is used in COLMAP. There is also an intermediary step before the SfM-based reconstruction stage, which is known as the geometric verification. At its core, the goal is to first find a valid homography (A 2D-to-2D transformation matrix) that matches as many feature points as possible from one image unto the corresponding points on another image. If the number of overlapped feature points are sufficient, that image pair is considered geometrically verified. Otherwise, with too many mismatched 2D points, the outliers needs get discarded. COLMAP used a variant of RANSAC for this. The output is a scene graph, where nodes are images and the edges are between nodes that are pairwise verified. The scene graph of COLMAP is also augmented with the geometric relations between image pairs.

At the next stage, the SfM reconstruction is an incremental process. This means that the model starts with a sufficiently good initial image pair in a two-view SfM reconstruction, perform image registration and pose estimation as explained in section 2.2.2 (Resection,

PnP or a variation of them), and then initialize a 3D structure via triangulation. After that, we perform *Bundle Adjustment* (BA). BA is a local optimization problem, with the goal of adjusting a bundle of viewing rays to minimize the total reprojection error (after projecting the initial triangulated points unto the image plane). It is a well-known least-square problem and, as we will see in this project, widely used in 3D Reconstruction. COLMAP performs local BA on an array of the most-connected images after each image registration, and a global BA periodically only after the 3D reconstructed object grows by a certain percentage. COLMAP also performs a re-triangulation before global BA to account for drifting errors. After some outlier filtering and refinement, the algorithms proceeds to receive new verified images to be registered, processes their feature maps, and thus, incrementally builds upon the previous 3D structure. These steps are performed iteratively until all the images have been processed.

COLMAP also supplies an optional method for generating dense reconstructions [7], which is utilized in this thesis. This method will produce a denser representation of the 3D environment from its initial sparse estimated point cloud. The primary purpose of dense modeling is to provide a detailed and more complete picture of the 3D environment from an aesthetic standpoint, typically used for VR/AR, rendering, localization and in our case, better annotation and classification capacity, all without sacrificing too much of the accuracy and robustness of sparse modeling. Dense reconstruction belongs to a separate category of computer vision problems known as *Multi-View Stereo* (MVS) problems.

2.2.5 Visual SLAM

SLAM can be categorized into several different approaches, whilst all sharing the same fundamental characteristics explained above. The one most relevant to our case of 3D reconstruction to study, is *visual SLAM* (vSLAM). In vSLAM, the goal is to use cameras, in monocular or stereo setups, as the primary sensors for capturing the environment and estimating the camera poses.

vSLAM methods in turn usually fall under either *direct* methods or *indirect* methods. The latter, also known as *feature-based* vSLAM, utilizes a set of feature points per image for 3D reconstruction. This stage of vSLAM is processed after image undistortion as explained in section 2.1.5.

These features are then matched across multiple images to determine the pose motion and the structure of the environment. The aforementioned popular algorithm SIFT is also widely used for feature matching. As the camera moves in the trajectory, depth can be inferred through parallax by comparing the features in subsequent images, assuming there is an adequate overlapping of visual information, i.e. shared features between a subset of images.

As the trajectory grows with more captured images, new visual observations are added into the equation, and the newly extracted features need to be associated to the already established 3D map by finding key elements within the map. Stitching the 3D map with new visual information introduces uncertainty when the 2D image does not quite match the expected/predicted location and motion within the trajectory. This inevitably accumulates noise in the 3D map with more unseen data. Therefore it is, as mentioned before, important to continually update the 3D representation and pose estimation for each batch of captured frames, to prevent future accumulation of noise.

A commonly used update method used in vSLAM is the previously mentioned bundle adjustment, that performs a non-linear least square operation on the current iteration of 3D

map and the estimated trajectory, for every batch of captured frames. A common practice is to use keyframes in the reconstruction update step instead of a fix-sized batch. That is to say, we selectively pick a small set of frames that are satisfactory representatives of the entire 3D environment. Since non-key frames in-between will be disregarded during map/pose update, the overlapping visual observation between keyframes is also smaller, and instead each keyframe contains key visual information unique to that frame.

2.2.6 DROID-SLAM

Ever since the concept of SLAM came to surface, many different implementations and approaches have been introduced stages in 3D point cloud and pose reconstruction, from image undistortion (correction) to feature matching. It started with enhancements to the probabilistic and filtering-based models, and has now advanced to various optimization-based formulations of the SLAM problem. Previously mentioned BA is an example of one such formulation, which enables joint optimization of 3D maps and trajectories in a single unified least square problem (a step forward from alternating pose and mapping recalculation). The most recent development in this field is utilizing and refining deep learning for different SLAM sub-problems, such as feature detection and matching, outlier removal, and (re-)localization, for which the research still remains at full pace.

One such deep learning-based SLAM system, developed as recently as 2021, is DROID-SLAM. The authors of the paper *DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras* [8] claim that their motivation of developing this system was to recover the lack of robustness of previous SLAM methods and to tackle newly emerging failures since the recent modernization. Some of which are lost feature tracks across frames, divergence in the aforementioned optimization problem, accumulation of drift, and more.

Widely regarded as a state-of-the-art visual SLAM system, DROID-SLAM significantly surpasses previous methods, classical and learning-based alike, in terms of estimation accuracy, robustness and even input generalization. This means that while it was trained solely with monocular synthetic data, it is able to directly use stereo or RGB-D input to produce improved accuracy with no retraining. DROID-SLAM combines different elements of its predecessors. For instance, it does neither label itself as an indirect nor direct visual SLAM. It uses the full image, performing (and updating) per-pixel depth estimation instead of processing feature detection, description and matching that lead to a sparse layer of corners and edges. In this sense, it borrows concepts from direct vSLAM. For updating poses and 3D reconstruction (depth maps), it uses elements from optical flow, namely RAFT [9], to perform recurrent iterative updates using a differentiable layer known as *Dense Bundle Adjustment* (DBA). In contrary to indirect vSLAM, DROID-SLAM optimizes per-pixel depth map directly rather than the sparse feature map. However, similar to BA from indirect vSLAM and unlike direct vSLAM, it utilizes the more accurate geometric reprojection error instead of the photometric error for minimization. The differentiality of DBA allows the DROID-SLAM network to use a gradient-based optimization method such as backpropagation to train the parameters. This includes training per-pixel depth values independently instead being bound to a fixed depth basis function. Consequently, the system of DROID-SLAM inhibits strong generality in terms of inputs.

2.3 NeRF: Neural Radiance Fields

The fundamental concept of NeRF methods, as introduced by [10], is to synthetically generate novel views given a sparsely sampled input image sequence. Specifically, this method operates by taking a collection of images captured at various angles and positions to produce additional views of the scene from previously unobserved locations. The resultant output is a 3D scene in which each point is represented differently based on the angle of observation, therefore enabling various lighting effects such as reflections and shininess. Mathematically, a NeRF-scene is represented as a 5D vector-valued function that takes a 3D location $\mathbf{x} = (x, y, z)$ and corresponding viewing direction (θ, ϕ) as input, and outputs the emitted color $\mathbf{c} = (r, g, b)$ and volume density σ. Thus, in contrast to the outputs of previously discussed reconstruction methods, a NeRF output is not a 3D point cloud. The 5D representation is approximated using a multilayer perceptron (MLP) network F_Θ : $(\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$, where $\mathbf{d}$ is the viewing direction expressed as a 3D Cartesian unit vector. This network optimizes its weights to provide mappings from each 5D coordinate to a volume density and directional emitted color. Then, by using a set of techniques called volume rendering, 2D images can be retrieved from the reconstructed scene. Finally, the scene representation can be optimized by using the $\mathcal{L}^2$ loss function between synthesized and ground truth images.

2.3.1 Instant-NGP: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

The major drawback that presents itself with NeRFs is the training time, which makes it impractical to use it at scale where the goal is to reconstruct several scenes. However, with the release of the paper *Instant Neural Graphics Primitives with a Multiresolution Hash Encoding* by [11], the authors at Nvidia presented a novel method for generating NeRF scenes which reduced previous training times from hours to a couple of seconds. This paper's main contribution which led to the drastically reduced runtime was the introduction of a novel way of encoding the input to the MLP. The input $\mathbf{y} = \text{enc}(\mathbf{x}; \theta)$ is encoded using a multiresolution hash structure, which then serves as input to the MLP $m(\mathbf{y}; \Phi)$. In addition to having trainable weight parameters Φ in the neural network, as in the previously discussed NeRF case, the parameters θ that are used in the encoding are also trainable.

In this approach, the first step is to create a grid of L levels with increasing resolution. The first and second levels, $l = 0$ and $l = 1$, are shown in Figure 6 as blue and red. Next, the resolution growth factor b is calculated between the coarsest and finest resolutions N_{min} and N_{max} as:

$$b := \exp\left(\frac{\ln N_{\text{max}} - \ln N_{\text{min}}}{L - 1}\right), \tag{14}$$

and the intermediate resolutions N_l as:

$$N_l := \left\lfloor N_{\text{min}} \cdot b^l \right\rfloor.$$

For a single level l, the input coordinate $\mathbf{x} \in \mathbb{R}^d$ is scaled to the level's grid resolution, and then rounded up and down:

$$\lceil \mathbf{x_l} \rceil := \lceil \mathbf{x} \cdot N_l \rceil, \;\; \lfloor \mathbf{x_l} \rfloor := \lfloor \mathbf{x} \cdot N_l \rfloor.$$

Hence, each rounded input coordinate is assigned a *voxel*, a cell of d dimensions, at all levels, and the voxels' corner coordinates are calculated. This yields a hash-like initial feature

vector of length T with F feature dimensions per entry, for every level, in which each entry contains mappings between input coordinates and voxel vertices. In Figure 6 this mapping is exemplified. The input coordinates $\mathbf{x}$ are assigned the center and bottom right voxels for levels $l = 0$ and $l = 1$ respectively, and the information regarding the voxels' corner locations are mapped to entries in the corresponding feature vector array for that level. The entries are mapped 1:1 if $(N_l + 1)^d \leq T$, which happens for coarse levels. Otherwise, for finer grid levels, a spatial hash function [12] is used to find the correct indices in the array:

$$h(\mathbf{x}) = (\bigoplus_{i=1}^{d} x_i \pi_i) \mod T, \tag{15}$$

where $\oplus$ represents the bit-wise XOR operation, and π_i denotes predefined, large prime numbers. As a final step in this encoding process, the values for each feature vector are linearly interpolated based on the relative position of $\mathbf{x}$, using the interpolation weight $\mathbf{w}_l := \mathbf{x}_l - \lfloor \mathbf{x}_l \rfloor$. Finally, the interpolated feature vector from each level is concatenated to represent the final vector that will serve as the input to the MLP, with an option to add auxiliary features. The steps explained in this section are visually demonstrated in Figure 6.

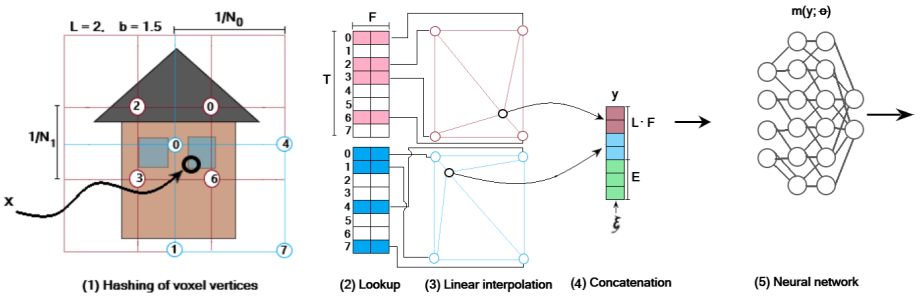


Figure 6: Given input coordinates x, the corresponding voxels are found for each grid level after scaling the input. The indices of these are then calculated and mapped to the feature vector for each level. Finally, the values in the feature vectors are interpolated, before adding auxiliary features and passing them to the MLP.

2.3.2 NeRF-SLAM

NeRF-SLAM is a state-of-the-art pipeline published in late fall 2022 that utilizes recent advancements in dense monocular SLAM and hierarchical volumetric neural radiance fields to achieve real-time scene reconstructions. The pipeline employs DROID-SLAM to estimate dense depth maps and camera poses, and proceeds by calculating their respective uncertainties. These uncertainties are then used for supervising a neural radiance field, which is implemented using the previously discussed real-time system Instant-NGP. By providing pose and depth estimates, and the marginal covariances, the authors claim that they are able to optimize the radiance field's parameters and improve the accuracy of the camera poses concurrently. The resulting pipeline generates accurate scene reconstructions and runs in real-time. Having previously examined DROID-SLAM and Instant-NGP in sections 2.2.6 and 2.3.1, our focus now turns to the contributions that [13] brought with NeRF-SLAM.

In short, the primary element introduced by the authors in NeRF-SLAM is the computation of uncertainties, which are used for weighting the depth loss function that supervises

the neural volume.

In the computation of uncertainties, the structure of the Hessian matrix, a matrix containing second-order partial derivatives, is leveraged to calculate the marginal covariances of the depth maps and poses estimated by DROID-SLAM. The calculations of these covariances for the depths $\mathbf{\Sigma_D}$ and poses $\mathbf{\Sigma_T}$ are formulated as follows:

$$\mathbf{\Sigma_D} = P^{-1} + P^{-T} E^T \mathbf{\Sigma_T} E P^{-1}, \quad \mathbf{\Sigma_T} = (LL^T)^{-1}, \tag{16}$$

where P represents the inverse depth per pixel per keyframe, E is the off-diagonal blocks of the Hessian, and L is obtained from a Cholesky decomposition of the reduced camera matrix.

The covariances, together with the input Images $\mathbf{I}$, poses $\mathbf{T}$, and depths $\mathbf{D}$ (weighted by $\mathbf{\Sigma_D}$), then serve as input for training the neural radiance field. In contrast to previous NeRF methods that use the $\mathcal{L}^2$ loss during the optimization stage of a scene reconstruction, the authors introduce a loss function that takes these uncertainties into account:

$$\mathcal{L}_M(\mathbf{T}, \Theta) = \mathcal{L}_{rgb}(\mathbf{T}, \Theta) + \lambda_D \mathcal{L}_D(\mathbf{T}, \Theta). \tag{17}$$

This mapping loss is minimized with regard to the poses $\mathbf{T}$ and neural parameters Θ, and has the hyper-parameter set to $\lambda_D = 1.0$. The depth loss in Equation 17 is defined by

$$\mathcal{L}_D(\mathbf{T}, \Theta) = \|D - D^*(\mathbf{T}, \Theta)\|^2_{\Sigma_D}, \tag{18}$$

where D^* is the rendered depth, D is the estimated depth, and Σ_D is the uncertainty of the depth. The color loss function $\mathcal{L}_{rgb}$ is identical to that in the original NeRF method, as mentioned in Section 2.3. The outline of the flow of the pipeline is shown in Figure 7.

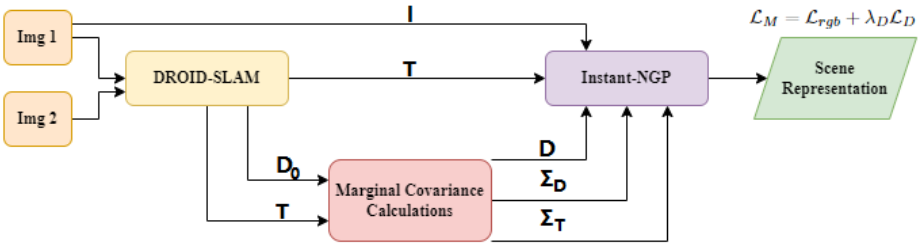


Figure 7: Flow diagram for the NeRF-SLAM pipeline, in which the input is two images. DROID-SLAM estimates camera poses $\mathbf{T}$ and depths $\mathbf{D_0}$. These are used to calculate the uncertainties $\mathbf{\Sigma_D}$ and $\mathbf{\Sigma_T}$, in which the former is used to weight the depths $\mathbf{D_0}$ to yield weighted depths $\mathbf{D}$. Finally, these three variables are passed to Instant-NGP in conjunction with the images $\mathbf{I}$ and poses $\mathbf{T}$ to generate the scene representation, which is optimized using the loss function $\mathcal{L}_M(\mathbf{T}, \Theta)$.

3 Method

3.1 Preparation of Datasets

For the purpose of this thesis, a diverse collection of datasets was acquired. The selection of appropriate public datasets, complementary to the data provided by Volvo Cars, was based on a few criteria:

(i) Access to camera intrinsics, ground truth poses and 3D points should exist, or the ability to extract them from the data.

(ii) The collection should contain both synthetic and real-world data.

The first criterion needs to be satisfied as this thesis aims to evaluate the performance of different reconstruction algorithms. As discussed in Section 2.2.1, the camera intrinsics need to be provided as a prior to the reconstruction algorithms. While it may appear self-evident, ground truths are needed in the task of evaluation since the estimations need something to be compared with. This thesis is limited to the evaluation of poses and 3D point cloud, which is why these specific ground truths are necessary. During the research of potential public datasets, we found that there was limited availability of pre-generated ground truth point clouds. This resulted in the second part of criterion (i).

In criterion (ii), the justification for using synthetic data in addition to real-world are based on a few reasons. One major advantage of synthetic data is that the ground truths are nearly error-free. Synthetic scene representations are made using 3D rendering software, such as the Unreal Engine (UE), which enables the possibility to sample ground truth points with known locations and distances. Whereas, in the real-world case, ground truth 3D points are usually gathered using a LiDAR in which the quality is affected by various factors such as the placement and angle of incidence, weather conditions, and more. In addition to generating accurate ground truths for our evaluation, with a more general aspiring goal, it is valuable for Volvo Cars to investigate the possibilities of using synthetic data in a broader sense. If the reconstructions generated using synthetic data hold a high enough standard, this may allow for the training of ADAS functions on synthetically reconstructed scenes. This could then eventually be used as a complement to Volvo Cars' real-world datasets, especially to recreate rare driving scenarios.

The resulting public datasets that were selected for this thesis are the synthetic TartanAir dataset [14] and datasets from the road scenic KITTI Vision Benchmark Suite [15].

3.1.1 TartanAir

The TartanAir dataset is entirely synthetically generated and contains photo-realistic scenes in various challenging environments. By utilizing the Unreal Engine, the authors in [14] were able to design photo-realistic sequences in different lighting and weather conditions, with complex geometry and various textures. To complete the dataset, the authors then used the AirSim plugin to collect multi-modal sensor data that contains diverse motion patterns and precise ground truth labels. This dataset includes a variety of sensor data, such as RGB- and Depth-images, camera poses, optical flows, segmentations etc. The intrinsics of the synthetic camera are also provided. An example of the synthetic RGB image from one of its sequences called *Abandoned factory* can be seen in Figure 8 (Right).

For our purpose in this project, we only use the per-sequence `.txt` file containing the camera poses, and RGB-Depth data in form of `.png` image files and `.npy` matrix files, respectively. Each line in the `.txt` file signifies a pose, and the number of poses/lines is equal to the number of RGB images and depth maps. The camera poses are primarily used as ground truth for pose evaluation, and RGB-D data is required for extracting a unified 3D point cloud, to be later used as ground truth for 3D points evaluation. For this, we utilize the transformation matrices.

The Depth sensor and the RGB camera are integrated as one camera model, capturing scenes simultaneously at the same location and rotation per timestamp. This means that the rotation matrix is an identity matrix, the translation vector is zero, and each pixel has corresponding RGB values and Depth scalars. We can create a (partial) point cloud by calibrating one RGB image to yield the 2D points on the image plane, and then simply use the depth maps to assign the depth values as z-coordinates to all the points in that image. The points with a large depth value are filtered out, since the distant objects (also the "sky") are not of interest, and they are usually captured in a few frames. The only objective that remains, is to transform each partial point cloud from the camera coordinate systems to one common global coordinate system. This is where we use camera poses once more, this time as transformation matrices from camera to global coordinate system. Now with all the partial point clouds created from the RGB-D dataset, in a common coordinate system, we simply gather all these 3D points in one unified point cloud. The unified ground truth point cloud of *Abandoned Factory* sequence, with z-coordinate (depth) cutoff of 45 meters, can be seen in Figure 8 (Right)

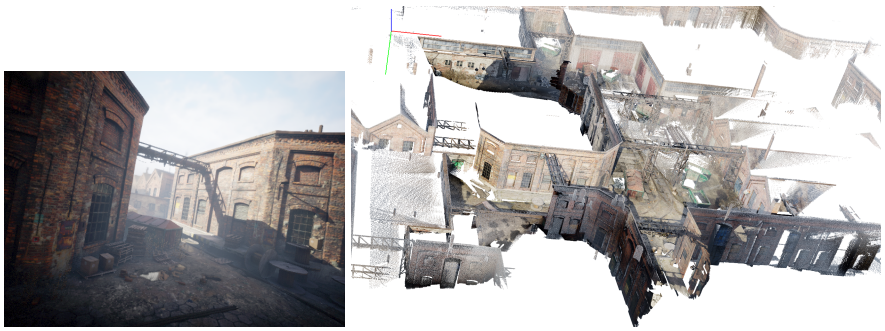


Figure 8: The *Abandoned factory* sequence from TartainAir. *Left*: One frame illustrating the synthetic RGB image. All pixel depth values of this frame can be extracted from its corresponding depth map. *Right*: *Abandoned factory* ground truth in it entirety, after converting depth maps to a unified point cloud. Sparse segments imply only a few frames have captured those parts, with little overlapping areas between images.

3.1.2 The KITTI Vision Benchmark Suite

The KITTI vision benchmark suite is a collection of benchmarks constructed for various tasks in computer vision, such as stereo, optical flow, SLAM, and 3D object detection. The data contained in these datasets was collected utilizing a car equipped with an inertial navigation system consisting of a Global Position System (GPS) receiver and Inertial Measurement Unit (IMU), a Velodyne laserscanner (LiDAR), two grayscale and color cameras, and varifocal imaging lenses, as shown in Image 9.

In the context of this thesis, our objective is to generate and assess accurate ground truth poses and 3D point clouds, leading us to choose the Visual Odometry / SLAM Evalua-

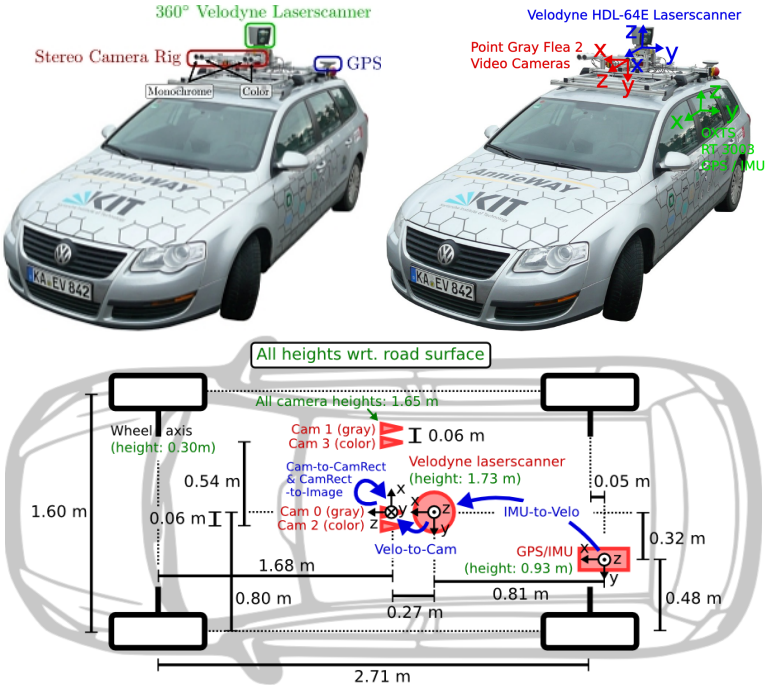


Figure 9: Sensor configuration, including the stereo camera rig in red, 360° . [Image source: `KITTI website`]

tion 2012 dataset. This dataset provides data for 11 road scene sequences, including images from dual RGB cameras, ground truth poses, and Velodyne laser data, or LiDAR point clouds. Despite the absence of color information, the precision of distance measurements and thus positional information is high, qualifying it as a reliable benchmark for evaluating the accuracy of our 3D reconstructions. The dataset includes pose data in a text file, represented as one-dimensional arrays that correspond to the initial three rows of a 4×4 homogeneous camera matrix, as defined in equation 19. As part of the preprocessing phase, these compact pose representations are reverted to their original matrix structure through a straightforward reshaping operation. This step is necessary for the evaluation of camera poses, a process that will be further detailed in a subsequent section.

$$[R_{11}, R_{12}, R_{13}, t_x, R_{21}, R_{22}, R_{23}, t_y, R_{31}, R_{32}, R_{33}, t_z] \Rightarrow \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{19}$$

What remains is to extract the LiDAR maps from the Velodyne laser scanner, and create a unified point cloud in a common coordinate system. The sequence we extract a segment from contains over 4500 frames, each with a resolution of 1382×512 pixels. Extracting, transforming and visualizing the entire sequence is computationally demanding, so we only select a portion of it, but the following unification procedure can be replicated irrespective of KITTI sequence and its size. The procedure shares the same steps as in TartanAir dataset for creating a single ground truth point cloud, with the difference being the partial point

cloud is already provided. It is of course the LiDAR 3D points in the Velodyne coordinate systems that needs to be transformed. Velodyne absolute poses are not supplied by KITTI, instead we first transform from Velodyne to the front gray camera with the provided (fixed) transformation matrix, and then use camera poses for unification. The forward axis of the Velodyne system is the x-axis, so we perform one last additional step, that is to transform the ground truth points from the NED frame to z-forward coordinate system.

The resulting ground truth point cloud can be seen in Figure 10 (bottom). We observe that the point cloud gets sparser and more spread out at the edges at distances far from the trajectory of the car. The sparse parts can be explained by the fact that objects on the edges are not captured as frequently by the LiDAR rays. This is due to their greater distance to the LiDAR system. Thus, the objects are only well-captured when the car is close enough. Subsequently, since the vehicle turns right, the remaining parts of the crossroad which the car does not traverse end up with a sparser representation. The sparsity can also be attributed to distant LiDAR rays, which inherently are more prone to misdirection.

The scene points on the edges are also more horizontally spread or stretched when compared to the crossroad. This drifting in the point cloud naturally occurs when the LiDAR and the front camera we extract poses from are not fully synchronized. This slight delay in shutter time makes it so that instead of yielding one 360° LiDAR sweep per frame, we may end up with a slightly larger sweep per frame. The redundant LiDAR points of the sweep ($x > 360^\circ$) will not overlay on the previous LiDAR points of the current sweep ($x - 360^\circ$) due to car movement, hence giving the illusion of the horizontal drifting. The drifting effect exists on the entire LiDAR-based point cloud, but they become more apparent at larger distances. Because of the trigonometrical characteristics of drifting, even an unnoticeable angular disparity leads to a relatively wide gap at long range.

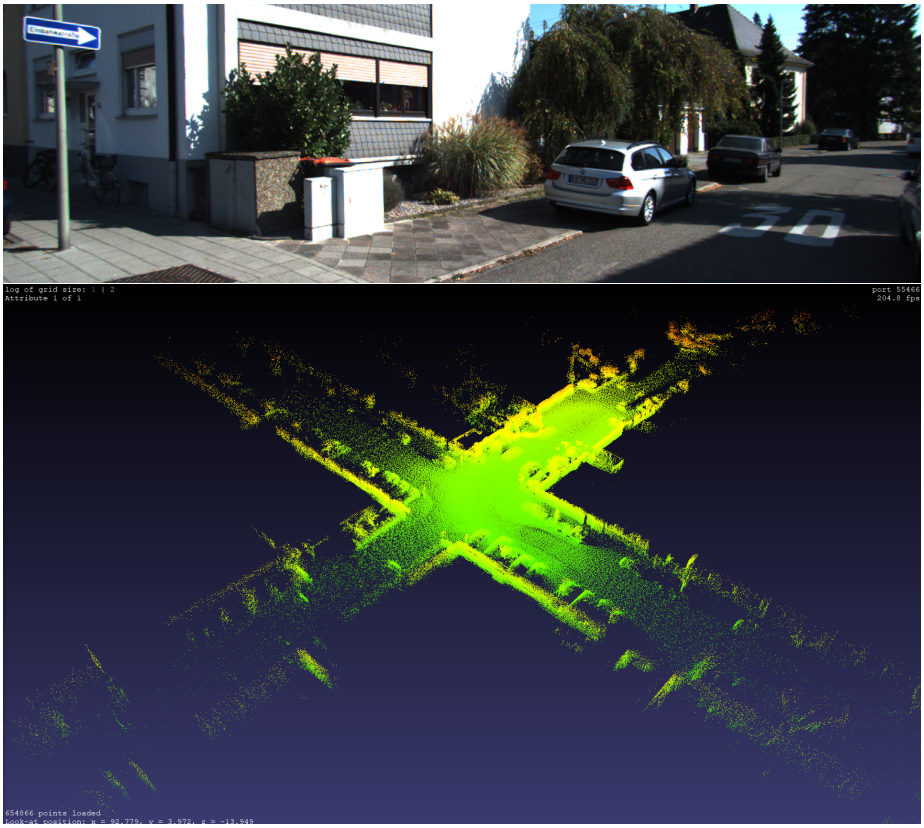


Figure 10: Top: Example image from the dataset. **Bottom:** LiDAR point cloud, used as ground truth for point cloud evaluation. Downsampled for visualization purposes.

3.1.3 Volvo Cars Data

Volvo cars are equipped with a perception system offering 360° peripheral vision facilitated by a combination of strategically placed cameras. This system includes four close-range (lower resolution) fisheye cameras: two located beneath the side-view mirrors, one integrated into the rear bumper, and the remaining one positioned within the grille. In addition to their ability to capture and detect objects in close range, these fisheye cameras also enable the detection of track lane markings and road details, capitalizing on their wide field of view. Enhancing the systems peripheral vision further, one long-range (higher resolution) fisheye camera is positioned on each side-view mirror, facing in the rearward direction. Furthermore, Volvo cars are equipped with a forward-facing long-range camera, located behind the upper segment of the windscreen. This camera provides a long-range perspective that augments depth perception and enables the detection of distant objects.

In addition to the camera-based perception, Volvo vehicles are further equipped with additional sensors, including LiDARs, IMUs and receivers for GPS or Global Navigation Satellite Systems (GNSS). These additional sensors offer a means to generate "ground truth" poses and point clouds used in this thesis. In short, the process of generating ground truth poses involves fusion of the position data derived from the GPS/GNSS receivers, which provides the translational component of the poses, with the orientation data derived from the IMUs, that provides the rotational component.

The sensor setup used in the Volvo vehicle which was used to gather the data used in this thesis is shown in Figure 11. Note that the other sensors visible in the image were not utilized in this thesis.

Figure 11: Image of the data-gathering car used to capture the images used in this thesis. The forward-facing fisheye and long-range cameras are highlighted with 1 and 2 respectively. $3, 4$ and $5, 6$ marks the positions of the short- and long-range fisheye cameras mounted on the side-view mirrors. $7, 8$ shows the location of the GPS/GNSS receiver and IMU. Finally, 9 shows the position of the LiDAR.

The images used in this thesis were captured by Volvo vehicles and provided by the Low Speed Perception team. Prior to this, the images underwent a series of processing steps, such as undistortion for the case of the fisheye images, and other image processing steps. This process followed the theoretical concepts outlined in Section 2.1.4. Two diverse sequences were selected for analysis, each of which presents unique environmental settings, contributing to the diversity of the evaluation. The first sequence solely contains images collected from the forward-facing long-range camera. The scene, as shown in Figure 12, was recorded within an urban setting where the vehicle drives along a straight path, and is

characterized by dynamic objects such as moving cars and the bright, sunny weather conditions. It is important to point out that no LiDAR data was provided with this dataset, and therefore there was no means of generating a ground truth point cloud to evaluate against. Consequently, only the estimated camera poses can be evaluated for this sequence.

In contrast to the first, the second sequence features images from all surrounding cameras on the vehicle, captured while the car navigates through a curve. This sequence was recorded in a static environment, specifically within a parking area, under overcast weather conditions. Images captured by the Volvo vehicle, in a single moment, are illustrated in Figure 13.

Figure 12: Image from the first sequence, captured by the forward-facing long-range camera.

Figure 13: Rectified images from the second sequence, captured by all of the cameras. *Top*: Forward-facing long-range camera. *Top row, left*: Short-range fisheye camera mounted on the left side-view mirror. *Top row, middle*: Forward-facing short-range fisheye camera. *Top row, right*: Short-range fisheye camera mounted on the right side-view mirror. *Bottom row, left*: Long-range fisheye camera mounted on the left side-view mirror. *Bottom row, middle*: Rear bumper fisheye camera. *Bottom row, right*: Long-range fisheye camera mounted on the right side-view mirror.

The generation of our ground truth point cloud for the second sequence was accomplished by manually adjusting four LiDAR sweeps such that they end up on a common coordinate system and consecutively, fusing them into one unified point cloud. These LiDAR sweeps correspond to four selected poses: Two poses at the start and end of the trajectory, and two other poses slightly before and after the curve in the trajectory. The ground truth point cloud is illustrated in Figure 14.

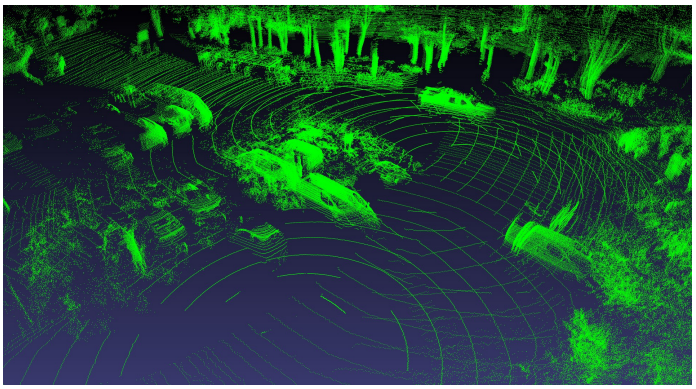


Figure 14: Ground truth point cloud created for the second Volvo sequence

3.2 Evaluation Pipeline

To assess the performance of the different reconstruction algorithms, an evaluation protocol was constructed. Selecting appropriate metrics for evaluating camera poses proved relatively straightforward, as most SLAM literature use the standardized metrics Absolute Pose Error (APE) and Relative Pose Error (RPE). Regarding the evaluation of reconstructed point clouds, the methods used in academia vary on a case-by-case basis and are not as standardized as in the pose evaluation case. It is worth noting that a considerable number of researchers refrain from publishing evaluations pertaining to their systems' ability to reconstruct 3D points, [16], [8], and instead provide images for the reader to visually assess. Inspired by the works of [17] and [18], we decided to use *Accuracy*, *Completion*, and *Completion Ratio* for our 3D evaluation.

To summarize, the metrics used for camera pose evaluation in this thesis are

- Absolute Pose Error
- Relative Pose Error,

and for 3D point clouds:

- Accuracy
- Completion
- Completion Ratio.

3.2.1 Camera Pose Evaluation

The evaluation of camera poses can be considered a relatively straightforward concept. Given a collection of estimated and ground truth camera poses, trajectories, which contains information about their positions, the goal is to measure the Euclidean distances between these positions. For this task, the `evo` Python package was used. The package has a range of functionalities for visualization, error calculations, and alignment, thus enabling accurate and easy comparison of the performance of different SLAM algorithms. Before using these functions, however, the data needs to be preprocessed.

The output format of COLMAP, DROID-SLAM, and NeRF-SLAM do not share a consistent representation. In the case of COLMAP, the camera positions need to be transformed from camera coordinates to global coordinates. To perform this transformation, the extraction of the rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ from the quaternion representation $\mathbf{Q} \in \mathbb{R}^4$ of the camera pose is required. A function from the `evo` package was utilized for performing this conversion. The result is a set of poses in which the positions are defined in relation to a common origin. Regarding the output format of trajectories generated by DROID-SLAM and NeRF-SLAM, no transformation is needed since they are naturally defined in global coordinates.

In addition to making sure that the poses are correctly orientated, they also need to have timestamps. With a timestamp for every pose, it is possible to find correspondences between estimations and the ground truth. Having timestamps as a prerequisite implies that we must make sure that both the ground truth poses and SLAM systems include timestamps. Instead, given that we can guarantee a one-to-one correspondence between the sets of estimated and ground truth poses, we simply index them with integers. To clarify, a one-to-one correspondence between the estimations and ground truth can be guaranteed due to the fact that we know which frame each pose corresponds to.

At this stage, the estimated trajectory can be synced with the ground truth using the timestamps, and then aligned. The alignment step is necessary since the poses can be defined in arbitrary coordinate systems, and was achieved using a function from the `evo` package which implements the Umeyama alignment method, as presented by [19]. This alignment method finds the similarity transformation $\mathbf{S} \in \mathbb{R}^{4\times4}$ that minimizes the squared errors between the estimated trajectory and ground truth poses. Assuming two trajectories of equal length, the algorithm first calculates the centroids (arithmetic means), variances, and covariance matrices of the trajectories. Next, singular value decomposition is computed, and then the optimal rotation matrix, scale factor and translation can be found, resulting in the final transformation $\mathbf{S}$. An illustration of a trajectory alignment is presented in Figure 15. Note that the transformation $\mathbf{S}$ is stored for later and used as a prior alignment in the point cloud evaluation stage. Further details regarding this will be provided in a subsequent section.

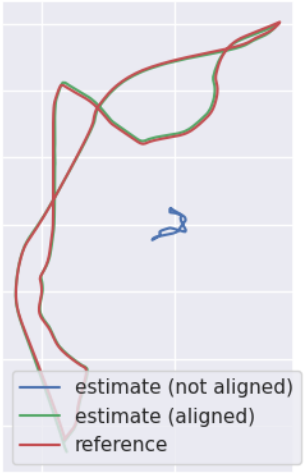


Figure 15: Illustration of a trajectory alignment process. The original trajectory is represented in blue, while the aligned trajectory is shown in green, following its alignment to the reference trajectory in red.

3.2.1.1 Pose Evaluation Metrics

As mentioned in the previous subsection 3.2, the metrics for evaluating estimated camera poses in this thesis are the absolute and relative pose errors.

The absolute pose error is a single number metric used to evaluate the accuracy of an estimated trajectory by measuring the Euclidean distance between the estimated and ground truth positions of the corresponding camera poses in 3D space.

Given an estimated trajectory $\mathbf{P}_{1:n} \in \mathbb{R}^{4\times4}$ consisting of n poses that has been aligned to the ground truth trajectory $\mathbf{Q}_{1:n} \in \mathbb{R}^{4\times4}$ using the similarity transformation $\mathbf{S}$, the absolute pose error at time step i can be computed as:

$$\mathbf{E}_i = \mathbf{Q}_i^{-1}\mathbf{P}_i. \tag{20}$$

The APE can then be used to compute various statistical values, such as the RMSE over all time indices:

$$\text{RMSE}(\mathbf{E}_{1:n}) := \left(\frac{1}{n}\sum_{i=1}^{n}||\text{trans}(\mathbf{E}_i)||^2\right)^{1/2}. \tag{21}$$

In addition to calculating RMSE($\mathbf{E}_{1:n}$), we also calculate the mean, median, standard deviation, minimum, and max values of the APE. The choice of only evaluating the translational error is due to the authors' conclusion in [20], in which they determined that the evaluation of the rotational component was not required due to the phenomenon of rotational errors showing up as translational errors upon camera movement. Thus, evaluating only the translational part can be considered sufficient since it indirectly captures the rotational errors.

The relative pose error measures the local deviation of the estimated camera trajectory from the true trajectory expressed over a fixed interval Δ, such as some time interval or on a frame-by-frame basis. The RPE at time step i is defined as:

$$\mathbf{F}_i = \left(\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta}\right)^{-1}\left(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}\right). \tag{22}$$

Furthermore, $m = n-\Delta$ relative pose errors are calculated and selected from the sequence of n poses, and the RMSE of the translational component is calculated as

$$\text{RMSE}(\mathbf{F}_{1:n}) := \left(\frac{1}{m}\sum_{i=1}^{m}||\text{trans}(\mathbf{F}_i)||^2\right)^{1/2}. \tag{23}$$

On the same basis as for the APE, the authors found the evaluation of the translational component alone to be sufficient when evaluating the RPE. As for the case of the APE, we supplement the evaluation of the RPE with the same statistical metrics mentioned previously.

In addition to syncing and alignment, the `evo` library provides all tools necessary for computing both the APE and RPE, and its corresponding statistical metrics. Notably, the library offers the ability to specify the units to be used in the metrics. The results can be presented in meters assuming that the ground truth poses are scaled accordingly. Furthermore, in this thesis, the RPE calculations are specified using a fixed interval Δ set to one frame. Thus, the RPE is measured between subtrajectories consisting of subsequent frames, on a frame-by-frame basis.

3.2.2 3D Reconstruction Evaluation

The secondary quantitative metric used in our findings is assessing the quality of the reconstructed 3D points. Similar to camera pose evaluation, we first need to transform the estimated point cloud such that it aligns with the ground truth in terms of position and rotation. In addition, we also need to align with a correct scale. We may also take re-sampling into measure, since it is likely that the two point clouds do not have identical density and number of 3D points.

This preprocessing step is a requisite for using many of the evaluation techniques, since commonly they rely on some form of relation in point pairs (such as using the distance between a ground truth point and a corresponding reconstructed point as an evaluation parameter). Thus it is vital to find a satisfactory array of mappings between the points in the ground truth and the estimated point cloud, as accurate as possible, before proceeding any further.

In a real life application, this task is of no easy feat, especially without a priori to infer. So the full process of the aforementioned steps may actually be more computationally demanding than the evaluation pipeline itself, depending on how formidable the dataset is.

3.2.2.1 General Case of Reconstruction Evaluation

As mentioned previously, we may not be able to use the evaluation metrics directly, as we have to first format the reconstructed point cloud $\mathbf{P}$ such that it is correctly scaled and aligned with ground truth $\mathbf{G}$. This then implies that we could use the classical *nearest neighbor* principle, i.e. to find point pairs in the two point clouds with the closest Euclidean distances, declaring them as point correspondences, and so continue the evaluation pipeline from here. However since we do not have a transformation matrix for alignment, this in turn requires a set of point correspondences – similar to how Umeyama algorithm takes as input pose correspondences in order to output alignment matrix. This problem is known as *point-set registration*.

There is a variety of Registration algorithms, all with the common goal of finding a spatial transformation that aligns two given point clouds. What distinguishes Registration methods is a multitude of factors such as the degree of correspondence, outliers, assumptions on (non-)Euclidean transformation, scaling, rotation, initial values, choice of objective/cost function etc. As we have discussed, the more prior information that is provided, the better accuracy (and generally with less complicated models) we end up with. A widely used Registration method when the two point clouds are of unequal size, where the corresponding points are unknown variables, is *Iterative Closest Point*, or *ICP*.

ICP is an algorithm with a good trade-off between performance and accuracy (with respect to the required prior information), first introduced by Yang Chen and Gerard Medioni (1991), and Paul J. Besl and N.D. McKay (1992), in which it performs Euclidean transformation in an iterative manner by alternating between the two steps: First find a set of corresponding points $\kappa \subset \mathbf{P} \times \mathbf{G}$ given the current transformation matrix T_i, and then find an optimal transformation matrix T by minimizing a predefined cost function $\mathcal{L}(T_{i+1})$ given the current correspondence set. The correspondence set κ can be anything from several key point pairs, to matching the number of points in $\mathbf{P}$, assuming the reconstructed point cloud is more sparse than ground truth $\mathbf{G}$. The objective function $\mathcal{L}(T)$ in point-to-point ICP is usually some variant of RMS-based optimization/error function we encountered earlier, for example $\mathcal{L}(T) = \sum_{(\mathbf{p},\mathbf{g})\in\kappa} \|\mathbf{p} - T\mathbf{g}\|^2$.

The supplied prior information for ICP Registration is of course some initial matrix $T_0 \in \mathbb{R}^{4\times4}$ in order to initiate the iterative two-step process. The initial alignment matrix T_0 needs to be somewhat acceptable. If it does not sufficiently align the point clouds, the Nearest Neighbor principle for assigning point correspondences may fail, and/or the minimization task may diverge – or at least it converges to suboptimal local minima. So a semi-correctly oriented point cloud $\mathbf{P}$ is still required for an overall effective alignment.

This is where we borrow another transformation matrix as prior that was previously used in a correspondence-based alignment task, namely pose correspondences in Umeyama algorithm. An example of ICP Registration can be seen in Figure 16. We can see that the ICP algorithm has diverged when an "incorrect" alignment, for example identity matrix, is used as prior. In this case both the scaling and the rotation are far from accurate even though the initial position is within reasonable range. As a matter of fact, we may get away with an unknown scaling if only the initial rotation and position are known. This way, we may reduce the prior requirement, but a correct initial translation on its own is not a sufficient prior information.

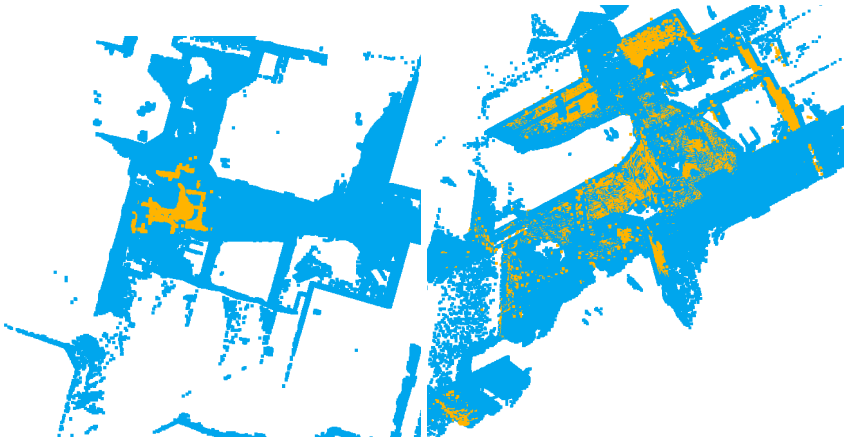


Figure 16: An excerpt of 3D reconstruction (yellow color) of the TartanAir dataset *Abandoned Factory* using DROID-SLAM, against the ground truth (blue color). *Left*: An attempt at alignment via ICP without a prior (4×4 identity matrix I is used as initial matrix). *Right*: ICP point cloud alignment with initial transformation matrix from pose alignment.

Generally in the case of unknown scaling, we can initiate a scaling by calculating the ratio of the axis-aligned bounding boxes (AABB) of the two point clouds, or some other average distance-based methods. Bounding boxes are helpful methods for a variety of applications, and they can be used to estimate a scaling factor even when the two point clouds are in different coordinate systems, i.e. without rotation nor translation. But this then will be an additional preprocessing step, since we need to contain the point clouds into suitable *oriented* bounding boxes (OBB), find the correct corresponding sides of the two OBBs and only then calculate the ratio. Bounding boxes are highly sensitive to outlier 3D points, so there is a further preprocessing step for efficient outlier removal (with a noisy reconstruction) before re-scaling. For our purpose, since we already have access to the scaling factor from the previous pose correspondences, we need not to perform this step, which may otherwise introduce a new source of error with noisy reconstruction.

For the ICP Registration procedure, we use the well-known `Open3D` library, available for many languages including Python. We use the method `o3d.pipelines.registration.registration_icp`, which is actually an optimized implementation of ICP, introduced

by [21].

So far, we focused on how point-set Registration ICP can be used to effectively align the two point clouds, which is, in its simplest form, an iterative series of assigning closest points, transforming the reconstructed point cloud such that the point pair distances are minimized, then re-assigning new closest points, and so on. Performing Nearest Neighbor search for each point in κ is however not as simple as it sounds. While it may be tempting to try to calculate (squared) Euclidean distance between a 3D point $\mathbf{p}$ and all potential points $\mathbf{g} \in \mathbf{G}$ and pick the closest $\mathbf{g}$, repeat that for all elements of κ per iteration, this will be an exhausting procedure in a long run. The keyword in the last sentence is the term "potential", meaning we have to define and selectively pick a handful of "good" points for distance measurement, and prune the remaining portions of the search space that is deemed unnecessary.

This is where we work with a space-partitioning data structure known as *trees*. This is of course a vast field of its own, so instead we turn our focus onto multi-dimensional binary search trees, or *k-d trees*. This multi-purpose algorithm is used for, inter alia, efficiently finding the point that is nearest to a given source point $\mathbf{p}$ using the tree properties, and also declare the point pair as correspondence. The latter is useful for 3D reconstruction evaluation metrics. As with many other algorithms presented so far, there have been multiple implementations, modifications and enhancements made to k-d trees, which are beyond the scope of this project.

The basic outline of vanilla Nearest Neighbor search using k-d tree in a 3D environment is as follows: The 3D space, which is the root node and can be thought as a cuboid bounding box, is partitioned into two equally sized half-spaces via a hyperplane, yielding two new rectangular cuboids which are the binary child nodes. We recursively perform this process for all sub-cuboids, until each cuboid contains a maximum of only one point of the point cloud. These cuboids, or voxels, containing a single point are then the leaf nodes and thus, a binary search tree is built. The tree stores positional information of one point cloud. A 3D points in this point cloud can then be efficiently addressed top-to-bottom from the root node to the corresponding leaf node, using binary addressing for each non-leaf node: Up/Down, Left/Right or Front/Rear. A simple illustration with 8 equally sized voxels can be seen in Figure 17.

With the Nearest Search search tree being constructed, we can *query* an arbitrary point, for example a 3D point from another point cloud, and use the tree to return the closest point. Due to the binary nature of this problem formulation, the average time complexity of the search algorithm is of course logarithmic, i.e. $\mathcal{O}(\log n)$, similar to search algorithms from other tree structures. So we calculate far fewer instances of Euclidean distance compared to the brute force approach mentioned earlier, which would otherwise have a time complexity of $\mathcal{O}(n^2)$ as we measure distance to all n point in the target point cloud. The latter time complexity occurs only in the worst case scenario of k-d search tree.

Assume the query point is within the root bounding box, that is an arbitrary 3D point among the target point cloud. The search algorithm first starts at the root node, and it moves down the tree recursively, where at each node it checks which half-space the query point ends up at, depending on its coordinates relative to the splitting hyperplane. This procedure is terminated as soon as we reach a cuboid with a single target point in it. This point can be interpreted as an "efficiently close" point to the query point, and so we calculate and save the Euclidean distance as the "current best" variable. There might be other points even closer to the query point from other directions, hence we backtrack in the k-d tree and search neighboring nodes/subtrees to the current best.

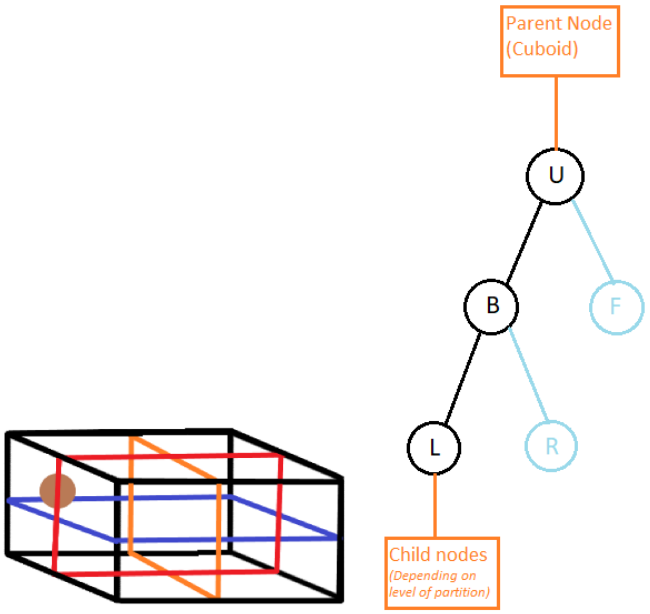


Figure 17: *Left*: A 3D point in some bounding box divided by 3 hyperplanes, resulting in 8 equally sized cuboids (or voxels if we have reached leaf nodes). *Right*: The 3D point can be locally addressed as *Up* $\rightarrow$ *Back/Rear* $\rightarrow$ *Left*. Note that this bounding box can be a child node of its parent cuboid and so on, when backtracking. Likewise, we can recursively partition the upper-back-left cuboid into 8 smaller equally-sized blocks and so on, depending on the density of the the point cloud. In the latter case, we go to a higher depth level (i.e. down in the tree) to address a 3D point.

The new subtree will be searched only if the shortest distance between the query point and the new cuboid is less than the "current best". In this case, it implies that there could be a "better" point in this new bounding box and thus, a similar procedure as above is applied. This means that once again, we move down this new subtree until a cuboid/voxel with a single point is found. This point will be assigned as the "current best" with its distance value saved only if the newly measured Euclidean distance is smaller less its previously saved value. We always avoid checking already-visited subtrees. We backtrack and repeat this process until we reach back to the root node, with no more potential node to visit. The final "current best" is the overall Nearest Neighbor to our query point. The k-d tree search algorithm is carried every time an arbitrary point is queried. The k-d tree of the target point cloud needs not to be rebuilt, though, as long as the point cloud remains the same.

3.2.2.2 Point Cloud Evaluation Metrics

Many scientific papers that present their state-of-the-art algorithms in the field of Computer Vision also perform thorough 3D reconstruction analysis, comparative to other algorithms. It could be, for example, a section dedicated to explaining in what scenarios which algorithms yield the best results, benefits and downside, viable parameter-tuning etc. It is, however, not as common for them to present side-by-side 3D point reconstruction evaluations in tables, in contrast to pose evaluation. As with the complications of defining true point correspondence and correct alignment, it is equally difficult to comparatively assess an entire reconstructed point cloud down to just numbers and score, as well as to agree upon a common evaluation pipeline for comparison.

Since we have selected a distance-based approach to declare point correspondences for 3D point cloud reconstruction, we also present our findings using distance-based quantitative metrics. The objective is not be interpret this as a definitive assessment, but rather to provide an overall image of the validity of the reconstructed point cloud and the algorithms used for it. To judge the "validity", we first include a modified definition of 'accuracy' that was used in the 3D Reconstruction tables of [18], as well as [17]. It is the average distance from the reconstructed point in $\mathbf{P}$ to their Nearest Neighbors in the ground truth point cloud $\mathbf{G}$. This means we construct a k-d tree of the ground-truth points, and query the reconstructed points one by one. We simply calculate the mean of the Euclidean distances we saved earlier from each query task. It is a length unit, so *lower* is better.

The aforementioned papers also present two other quantitative metrics: Completion and completion ratio. Completion works in a similar manner to accuracy, but the reverse: It is the average distance from the ground-truth points in $\mathbf{G}$ to their Nearest Neighbors in the reconstructed point cloud $\mathbf{P}$. So we build another k-d tree consisting of the estimated points as leaf nodes, and we query the ground truth points. Here as well *lower* is better, due to usage of length unit. We are also interested in the proportion of corresponding points that are sufficiently closely aligned. In other words, we output the percentage of completion queries with distances under a certain threshold. This ratio is known as completion ratio. Since it is a percentage-based metric, *higher* is better, with 1.0 or 100% being the maximum possible value.

To build k-d trees for the evaluation metrics, we take advantage of a package from Python library known as `SciPy`. We build a k-d tree by making an instance of the class `scipy.spatial.KDTree` and insert our point cloud as argument to its constructor. We can then use one of its method named `query()` to query arbitrary points and output the Nearest Neighbor dis-

tances. The tree data structure used in `SciPy` is a modified version of k-d trees implemented by [22].

3.3 Visual SLAM Method Selection

The process of selecting the appropriate visual SLAM methods was made based on a few criteria, as follows:

(i) Both classical and state-of-the-art deep learning methods should be included.

(ii) The selected methods should be monocular-based, relying solely on RGB-images and camera intrinsics as input data.

(iii) One method based on NeRF should be included.

In previous work, focus has been on using classical SfM-based approaches. But with the recent advancements in deep learning-based methods, it is of interest to compare performance between classical and deep learning-based approaches. The final criterion is motivated by the widespread attention that NeRFs have received within the research community, and reflects a desire to research their potential for generating high-quality point clouds and augmenting existing image sets through the synthesis of novel views.

- **COLMAP**: This classical SfM system has previously been researched and experimented with at Volvo Cars for generating reconstructions of road scenes. In general, the quality of the reconstructions has been sufficient, but the major drawback of this method is its runtime. In addition to the existing expertise with COLMAP in the low speed perception team, their interest in using the method in this thesis extends to a comparative evaluation of its performance in contrast to deep learning-based approaches.
- **DROID-SLAM**: DROID-SLAM has shown promising results that can be compared to more robust classical approaches. The biggest motivator for selecting this visual SLAM system in this thesis is related to its performance in terms of reconstructing scenes accurately at a very low runtime. Notably, DROID-SLAM excels in the ability to estimate precise camera poses, as demonstrated on various benchmarks. It is worth noting that the ability to reconstruct 3D point clouds has not been explicitly evaluated in the corresponding paper. But the visualizations that the authors provide look very promising. Since we perform evaluation of not only estimated camera poses, but also point clouds, we aim to contribute to the development of knowledge with these new insights.
- **NeRF-SLAM**: The number of scholarly publications on NeRF-style methods has drastically increased over the past few years. NeRF is an active field of research, and recently published NeRF-SLAM is an interesting approach to include in this comparative study due to its ability to generate data from novel views. The motivation behind selecting NeRF-SLAM specifically over other NeRF-methods was mainly due to its low runtime and accuracy compared to competing approaches, but also the fact that it satisfies criterion (ii). Numerous alternative NeRF-methods, such as [10] and [11], require the additional input of camera poses. Such methods require an additional preprocessing step, for example by estimating the camera poses utilizing

COLMAP, which significantly increases the overall reconstruction time. Furthermore, as brought up previously, it is also interesting to explore the viability of using NeRFs for augmenting existing datasets with synthetically produced images from novel views.

3.4 Experimental Setup

In this section, details regarding input and output formats, and conversions necessary to run the visual SLAM methods will be given. Furthermore, we will delve into the details of the extraction of the reconstructed camera poses and point clouds from each system, which in some cases proved to be a significant challenge.

3.4.1 COLMAP

There are multiple ways to get COLMAP started. We can either run its Graphical User Interface, or as command lines. Both ways offer almost the same range of functionalities and control. We select GUI. COLMAP offers two 3D reconstruction modes: We can either set it to perform the entire Incremental SfM pipeline automatically, or we can execute the SfM process step by step. We choose the manual processing. To see how to get started with COLMAP and its Automatic Reconstruction, we refer the reader to Appendix at the end of this project.

Manual reconstruction allows for a great flexibility of specification and parameter-tuning, of which we only use a few during our project. After selecting the working directory, we load the RGB image folder we like to work with. As we have seen in the theory section, the first step is to detect and extract features. We use the feature extraction method of COLMAP. We leave most of extraction parameters by their default values, as they already have an adequately good trade-off between reconstruction robustness/quality and performance/speed. We do however insert the known camera intrinsics, which are our prior information. See Figure 18. The process of feature extraction, and all other steps, are logged.

The next step is *Feature matching*, which offers a a whole new array of options and parameters to fine-tune to our use. Once again, we leave the matching options and parameters unchanged and run the matching process. COLMAP matching also includes the geometric verification step. Now with the feature point correspondences available, we perform the SfM-based reconstruction, with Bundle Adjustment. The output format is the same as in Automatic reconstruction (See Appendix), where the produced files are poses, triangulated 3D points and camera intrinsics.

The layout of the output camera poses is as such: There are two lines per pose. First line contains the Image ID, followed by four values representing the rotation in quaternions form, three values for translation, and the camera ID (For shared intrinsics it is always 1, since the same camera system is used to capture the entire 3D scene). The second line contains a list of feature points, more specifically their pixel coordinates (x, y) and the 3D point ID each keypoint represents. This is known as co-visibility info. The 3D points and their IDs can be found in the point cloud files. In addition to the essential values such as the 3D location and the RGB values per 3D point ID, each line also contains co-visibility info, i.e. which image planes they appear in (Image ID), as well as the index of keypoint each 3D point corresponds to.

The co-visibility info does not serve any interest when performing 3D metrics for recon-

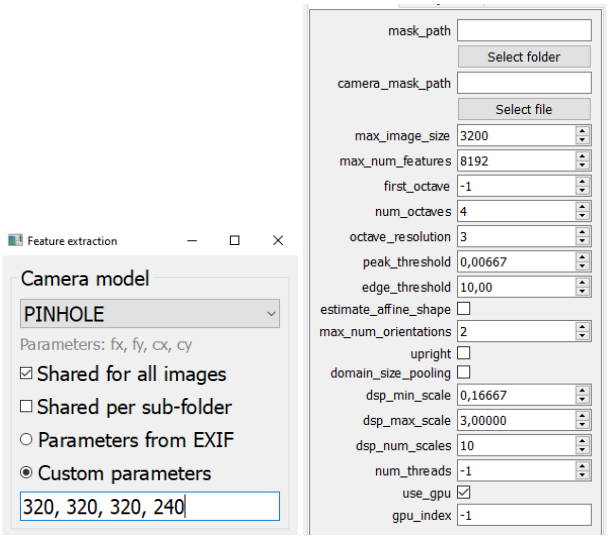


Figure 18: The *Feature extraction* dialog. *Left*: The upper pane, detailing the specifics of the RGB cameras(s) used. As an example, A single pinhole camera is used in all sequences of TartanAir dataset. So here we select *PINHOLE* from the drop-down menu, check in *Shared for all images* and since the intrinsics are provided, we select *Custom parameters* and fill in the text entry box, such as focal length f_x, f_y and principal point c_x, c_y *Right*: The lower pane, specifying the feature descriptors such as size, scale etc. using SIFT by default. Alternatively, feature descriptors can be imported. These are out of scope of this thesis project and so, we leave them untouched.

struction evaluations. So we trim these parts away when inputting the aforementioned files into the evaluation pipeline. The co-visibility info is however useful for other computer vision applications, such as creating a dense representation of the 3D environment via COLMAP MVS. Co-visibility is also useful for creating meshes

For our project, we decide to only densify the COLMAP sparse point cloud, as DROID-SLAM does not output the co-visibility info and it requires external pipeline for this, plus the densification step is already an exhaustive process even for a sparse point cloud.

3.4.2 DROID-SLAM

Setting up DROID-SLAM to run on custom datasets is a straightforward procedure. The required inputs for scene reconstruction include camera intrinsics containing focal lengths and principal points, with the option to include radial and tangential distortion parameters, as well as the path to the image folder. However, there are several adjustable flags within the pipeline that can be fine-tuned. In the initial experiments, the default flag values specified in the `demo.py` script were used. Nevertheless, certain parameters can be further refined within the pipeline. Specifically, in the `visualization.py` script responsible for rendering the reconstructions, the parameter `filter_thresh` was increased from `0.005` to `1.0` during the initial experiments. This particular parameter is used to scale a threshold value, which is used for filtering out 3D points during the visualization process. As discussed in Section 2.2.6, DROID-SLAM estimates depths on a per-pixel basis. Consequently, the authors employ a rather aggressive masking strategy based on the threshold value to remove estimates with excessively large depth values relative to the current frame. By setting `filter_thresh=1.0`, the aggressiveness in the masking process is reduced. Although this may be done at the expense of accuracy, it results in denser point clouds,

which is a trade-off that aligns with our goal of producing annotatable reconstructions.

By employing the default-valued parameters along with the modified parameter `filter_thresh`, we can successfully run DROID-SLAM on custom datasets, enabling the generation of dense point clouds and camera poses. These camera poses are represented by quaternions $\mathbf{Q} \in \mathbb{R}^4$ along with a translation vector $t \in \mathbb{R}^3$. The architecture of DROID-SLAM is built such that it exclusively generates poses for keyframes. To retrieve the poses for non-keyframes, the authors iteratively estimate the optical flow between each keyframe and its adjacent non-keyframes, and then perform a motion-only BA to the non-keyframes, a process which can be likened to the concept of interpolation. The resulting camera poses for all frames can simply be stored for later evaluation, requiring no further post-processing.

Furthermore, in order to extract the reconstructed point clouds from DROID-SLAM, the authors' rendering code in `visualization.py` is utilized. This is simply done by storing the generated 3D points that are passed to the Open3D-renderer, along with the points' respective RGB-values.

3.4.3 NeRF-SLAM

As mentioned in Section 3.3, NeRF methods typically require the additional input of camera poses in addition to RGB images and camera intrinsics. This is due to the fact that the poses provide information about the viewing direction which is used in the 5D input vector that is passed to the radiance field function, as introduced in Sections 2.3 and 2.3.1. Recall that for the case of NeRF-SLAM, poses are estimated via DROID-SLAM and eventually passed to Instant-NGP (see Figure 7). However, the architecture of NeRF-SLAM is designed in a manner such that the initial input format is similar to that of Instant-NGP. In practice, this meant that we had to write a script for creating JSON files containing information regarding the images and camera for every sequence that we wanted to reconstruct, with placeholders for the camera pose data. The JSON file contains the following information, in which the first four are shared for all images in a sequence:

- *Camera intrinsics*: Focal lengths, radial and tangential distortion parameters, and principal points. Given that we only have access to the focal lengths and principal points, we set the remaining of parameters to 0.
- *Image dimensions*: the width and height of the images.
- *Axis-aligned bounding box (AABB) - corner locations and scale*: defines the cube that constrains which 3D points to include in the reconstruction. To clarify, the reconstruction of points outside the AABB is omitted. We chose to maximize the size of our AABB for all sequences to ensure that the entire scene is captured. The AABB can then be reshaped and rotated manually after the scene has been reconstructed, which allows us to only extract the relevant parts of a scene. The initial cube that we set has its corners located in $[0, 0, 0],\ [1, 1, 1]$ and scaled is with a factor of 128.
- *Integer depth scale*: used to configure the quantization scale of depth maps. A lower value of this parameter yields higher precision quantization for the depth values. For the purpose of reconstructing road scenes, we experimentally found the value `integer_depth_scale = 0.001` to yield the best results.

  The following parameters are unique for each frame:

- *Image paths*: path to images.

- *Depth paths*: path to depth maps. Note that the depth maps are estimated and updated via the DROID-SLAM module, which is why we treat this parameter as a placeholder and set it to `null`.
- *Transform matrix*: the camera pose for the frame. Note that this is also treated as a placeholder, and thus set to be a 4×4 identity matrix.

After constructing a JSON file on the format explained above, we were able to run the NeRF-SLAM pipeline on any dataset. We proceeded by extracting the estimated camera poses outputted by the tracking module (see $\mathbf{T}$ in Figure 7), that is, before being passed to Instant-NGP. Note that, during runtime, NeRF-SLAM is designed to processes a maximum of 8 keyframes through a sliding window approach. Consequently, the poses and their corresponding frame indices need to be extracted iteratively and new poses are only stored when a new keyframe is encountered. Initially, new poses and their keyframe indices are stored, while a dictionary that contains a set of keyframe-frame key-value pairs is created and maintained. This dictionary is used in a post-processing stage during evaluation, where it is utilized to extract the correct ground truth poses through conversion.

For the extraction of point clouds, the first step is to manually align and reshape the AABB as previously mentioned. This is done using controls in the graphical user interface (*GUI*) which allows us to translate and rotate the box to fit the areas of interest of the scene that we wish to export. In Figure 7, a manually translated and rotated AABB in a NeRF-scene is illustrated, with the GUI controls highlighted in red. In an attempt to somewhat reduce the human error in this process, we always set the bounding box with the trajectory centered. The next step in the point cloud extraction process is to employ the function '`compute_marching_cubes_mesh()`' defined in the '`instant-ngp`' Python library to convert the scene representation into a mesh. Finally, the point cloud can be retrieved by extracting the color and positional information from the vertices in the mesh.

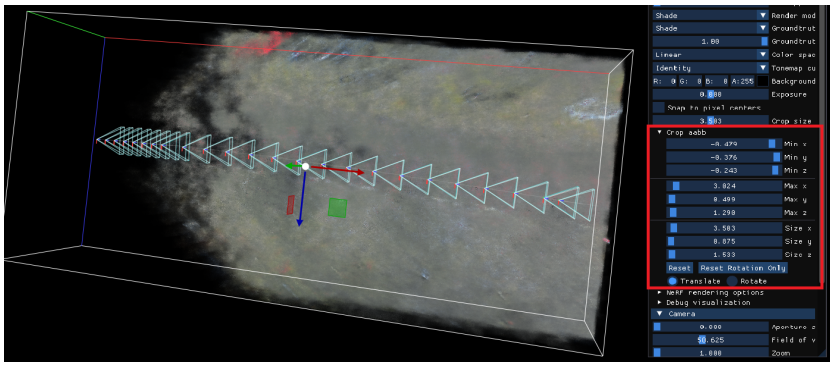


Figure 19: Bird's-eye view of a NeRF-SLAM scene showing the AABB, and the GUI controls which are used to manually set the constrains of the box.

3.5 Mesh Generation from Multi-Camera Point Clouds

Although the primary focus of this thesis lies on the generation and evaluation of point clouds, a portion of time was allocated on converting these into meshes. There are numerous advanced methods and software to generate meshes, such as the previously mentioned MVS techniques (Section 2.2.4). Given the objectives of this post-processing task

for this thesis, however, simplicity and runtime were prioritized. Mainly, this exploratory operation aimed at improving annotation capabilities and analysing the effectiveness of reconstructing key road features, such as parking lines. Initial observations indicated that some objects, which appeared ambiguous in the point cloud format, could potentially be more visually distinguishable after mesh conversion. More importantly, however, was the potential for improved visual analysis of the algorithms' ability to reconstruct uniformly flat road surfaces.

After conducting separate reconstructions using images from each camera in the multi-camera system of the Volvo vehicle, 7 point clouds and trajectories are generated in total. Given that the estimations are generated separately, and from cameras with different local coordinate systems and orientations, an assumption of a shared global coordinate system between the reconstructions can not be made. Thus, in order to ensure proper alignment and capture the entire scene captured by the vehicle, it is necessary to transform each sub-point cloud into a global coordinate system, resulting in a unified point cloud.

By leveraging the ground truth trajectories for each camera, the alignment transformation $\mathbf{S} \in \mathbb{R}^{4\times4}$ that aligns the estimation with the ground truth can be calculated utilizing the constructed pose evaluation tool. The alignment transformation can then be calculated for each trajectory estimate, and applied to its corresponding point cloud, similar to the initial steps made during point cloud evaluation (Section 3.2.2.1). Thus, the associated sub-point clouds are merged into a cohesive point cloud, covering the entire surroundings of what the cameras on the car have captured.

Leveraging the `Open3D` library in Python, Poisson surface reconstruction as detailed by [23] could be performed using only the cohesive point cloud and its color information. Prior to this meshification process, outliers whose distance to neighboring points exceeded the average distance between neighbors in the point cloud were removed. Though the resultant mesh could benefit from further optimization, for the purpose of analyzing road surface flatness and improving the visual distinction of objects, it is deemed sufficient for our objectives.

4 Results and Discussion

4.1 TartanAir

Table 1 provides the evaluation results of the methods' performance on the TartanAir sequence.

Table 1: Evaluation metrics for estimated camera poses and reconstructed point cloud for the Abandoned Factory (TartanAir) sequence.

	COLMAP	DROID-SLAM	NeRF-SLAM
APE [m]↓	0.0535	0.0521	**0.0055**
RPE [m]↓	0.3447	**0.3436**	1.4549
Accuracy [cm]↓	Sparse: **6.1182**, Dense: 8.1979	8.4530	129.42439
Completion [cm]↓	Sparse: 18.9534, Dense: **4.9264**	146.4535	234.2886
Completion Ratio [$< 5cm\%$*]*↑	Sparse: 40.0690, Dense: **83.8309**	54.7573	53.9628

As seen in Table 1, the RMSE values for both APE and RPE show close similarity for the COLMAP and DROID-SLAM methods, respectively. Note that from here on, when APE and RPE are mentioned, it is rather the RMSE value of the metric that is referred to. It has been previously mentioned that DROID-SLAM was exclusively trained on data from the TartanAir sequence, so the small magnitude of the RMSE, roughly 5.2 cm, is to be expected. Ultimately, DROID-SLAM was trained to minimize pose loss using ground truth poses from this very dataset. It is however important to note that details regarding which sequences from the TartanAir that were selected for training are omitted by the authors of [8], so we cannot know with certainty that this specific sequence was included in the training dataset.

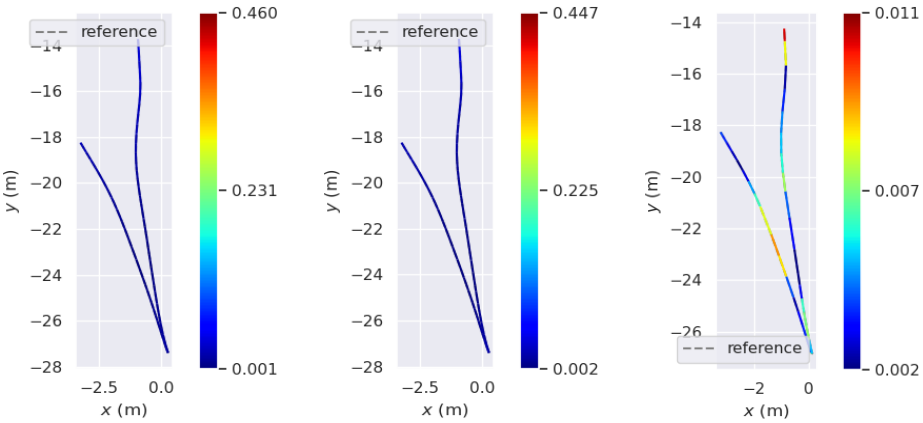
In terms of COLMAP pose error metrics, the result shows the robustness and accuracy of the incremental SfM-based approach. In Figures 21a and 21b, which visualize the RPE of COLMAP and DROID-SLAM respectively, it is clear that both estimations tend to drift from the ground truth trajectory as the camera progresses through the scene. To clarify, the start of the trajectory is located at $y \approx -18$ in the plots, and the scaling of the values varies across the figures. The RPE, which has a magnitude of roughly 34 cm as illustrated by these figures, could stem from the rapid camera motion and increased distance between camera views of that part of the trajectory.

In comparison to the other methods, NeRF-SLAM stands out with a significantly lower APE, as shown by the data in Table 1. Considering the exclusive generation of poses for keyframes in NeRF-SLAM, as discussed in Section 3.4.3, it offers a plausible explanation for this superior result. As discussed in 2.2.5, keyframes are selected to be representative of the 3D environment, usually containing features expected to yield reliable matching. Consequently, NeRF-SLAM has a higher confidence in its ability to estimate poses for these specific frames, compared to non-keyframes. Thus, the explicit evaluation of keyframe poses introduces a potential bias, a process of cherry-picking only the most accurate poses. Additionally, as detailed in Section 2.3.2, NeRF-SLAM employs the TartanAir-trained pose estimation core of DROID-SLAM. It is reasonable to believe that these two factors are the main contributing factors to NeRF-SLAM's 5.5 mm APE precision. It is however important to point out that the aspect of keyframe-only evaluation only pertains to the pose metrics. Thus, it is irrelevant to the point cloud evaluation.

At first glance, NeRF-SLAM's recorded RPE of 1.45 m indicates a notably inferior capa-

bility in capturing the relative movement in the trajectory, especially when compared to the 34 cm RPEs yielded by the other methods. This outcome, similar to the APE value produced by NeRF-SLAM, is likely to be a consequence of the evaluation being restricted solely to keyframes. As previously detailed in Section 3.2.1.1, the fixed time interval Δ is set to one frame. However, the terms "frames" and "poses" are synonymously used when discussing units in the context of RPE evaluation. Consequently, given that NeRF-SLAM does not generate poses for non-keyframes, these sub-trajectories cover larger distances relative to those evaluated by COLMAP and DROID-SLAM, which yield a pose for every frame.

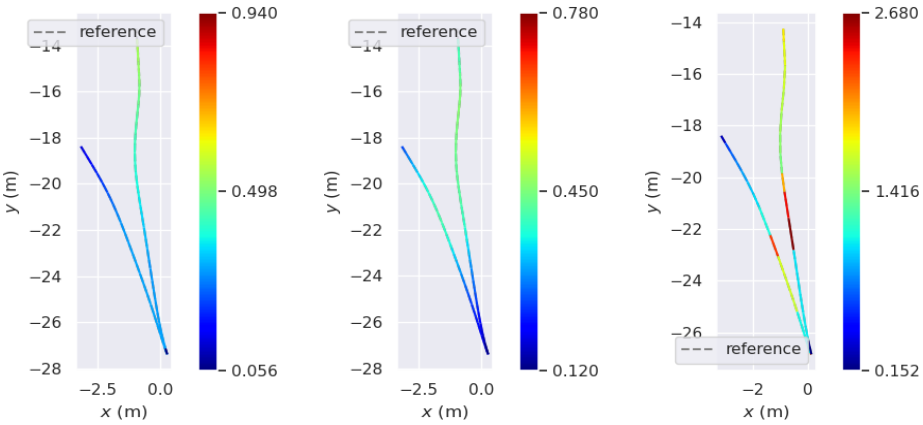
As depicted in Figure 22, the RPEs of NeRF-SLAM and DROID-SLAM are visualized from the xz-perspective, thereby illustrating the variation of RPE as a function of the camera's horizontal and vertical movement. In this representation, the sub-trajectories of NeRF-SLAM are quite distinguishable by their colors, suggesting that instances of rapid camera elevation for a sparse set of poses show up as significant errors in the RPE metric. In contrast, the denser pose set of DROID-SLAM yields more blended colors of the sub-trajectories in the figure, confirming the low RPE errors.



(a) COLMAP (b) DROID-SLAM (c) NeRF-SLAM

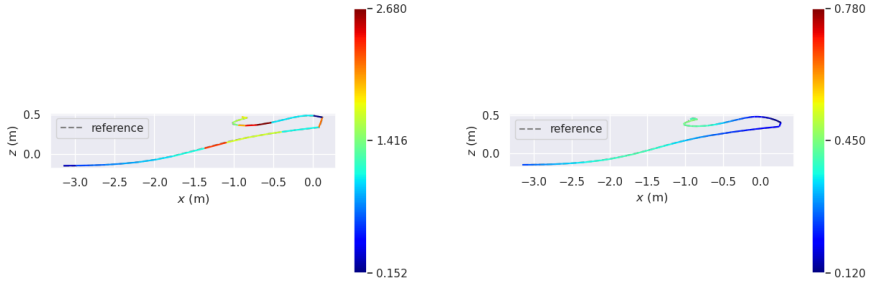
Figure 20: RMSE(APE) on the TartanAir sequence for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.

In terms of the point cloud evaluation in Table 1, a few noteworthy results can be pointed out here. The reconstruction yielded by DROID-SLAM on this sequence has a surprisingly large completion despite having a decent accuracy and completion, on par with COLMAP sparse and COLMAP dense reconstructions. But dissimilar to COLMAP, the reconstructed point cloud from DROID-SLAM covers a considerably smaller part of the ground truth in terms of area size, and here with very little to no outlier points. This phenomenon is illustrated by Figure 24 (bottom left), which shows the reconstructed point clouds (yellow) aligned to the ground truth (blue). This is because ground truth points are queried in the k-d tree of reconstructed points, which leads to high completion distances when the nearest neighbor to a ground truth point is still at a large distance. A completion ratio of 55%, confirms the fact that more than half of all completion distances are below the error tolerance. It is noteworthy that the reconstruction generated by DROID-SLAM in this scenario could have yielded a better completion value if it produced some errors and outlier points in those areas instead of filtering them out, which signifies one of the



(a) COLMAP (b) DROID-SLAM (c) NeRF-SLAM

Figure 21: RMSE(RPE) on the TartanAir sequence for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.



(a) NeRF-SLAM (b) DROID-SLAM

Figure 22: Comparison of RMSE(RPE) on the TartanAir sequence between NeRF-SLAM and DROID-SLAM as viewed from the xz-perspective, illustrating the vertical movement of the trajectories.

difficulties with point cloud evaluation.

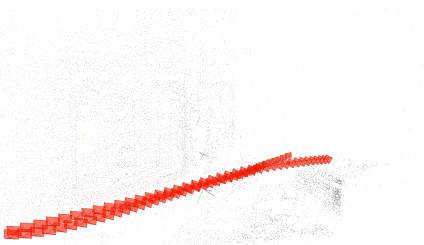
Another result that sticks out is the performance of NeRF-SLAM. It produces a lower completion score and completion ratio than DROID-SLAM, and it has an accuracy of 129cm, which is significantly worse than other algorithms. The low accuracy of NeRF-SLAM can be attributed to the highly dense block of 3D points that lays below the ground, and another dense segment of reconstructed points that end up behind the building facade. See 24 (Bottom Right). Since the nearest ground truth point to all these estimated points are still at a certain distance, it results in a low accuracy. So even though the point cloud captures the essence of the 3D environment well enough as seen in 23 (e), due to its bounding box property, it ends up with a small reconstruction segment which contributes to low completion, and a large volume of point mass which contributes to low accuracy.

The dense reconstruction of COLMAP outputted an impressive result on this sequence. The data shows a completion of 4.9 cm and a completion ratio of 84%, all while maintaining similar accuracy as for its initial sparse point cloud, and slightly better than DROID-SLAM. Since it passes all three 3D evaluation metrics to a satisfactory level, we can draw a preliminary conclusion that the reconstructed point cloud is adequately well-estimated.

The alignment of the COLMAP dense reconstruction can be seen in Figure 24 (top right). We can see that the point cloud is well-aligned on many surfaces and edges, but unlike DROID-SLAM, it covers a larger segment of the ground truth, reconstructing objects located further away from the trajectory. However, we may also notice that in Figure 24 (top right), especially the upper right part of the reconstruction segment, parts of the point cloud look stretched out (and sparser) when the dense model tries to reconstruct distant objects. As a result, there is a presence of a large influx of dense points horizontally, which may otherwise be unnoticeable when perceiving distant objects from a ground-level field of vision. As we will see later with other COLMAP dense reconstructions, this elongation or stretching effect in the dense point cloud is a common occurrence within a Multi-View Stereo reconstruction.

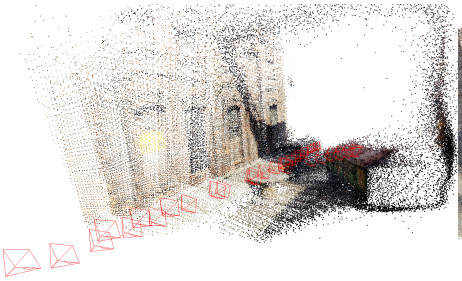
During the dense reconstruction process, depth is estimated by triangulating corresponding points in the virtual stereo image pair, created via COLMAP. Using the geometry between the epipolar lines coming from a pair of 2D points, their disparity, and the interception point in 3D space, a depth value can be accurately determined at close distances. However, the accuracy and density of the depth estimation decrease as the points move away from the distance between the viewpoints, known as the baseline. At a large distance, the angle between the epipolar lines becomes too acute, and the baseline is too small compared to the epipolar lines. This causes an apparent parallax, meaning there will be some depth ambiguity. So for multiple image pairs triangulating the same scene point at a large distance, the depth of the point is estimated differently with a large variance and little overlapping, which results in this stretching out effect.

(a) Ground Truth



(b) COLMAP Sparse Reconstruction

(c) COLMAP Dense Reconstruction



(d) DROID-SLAM

(e) NeRF-SLAM

Figure 23: Point cloud figure. Ground truth, followed by the reconstructions generated by the pipelines.

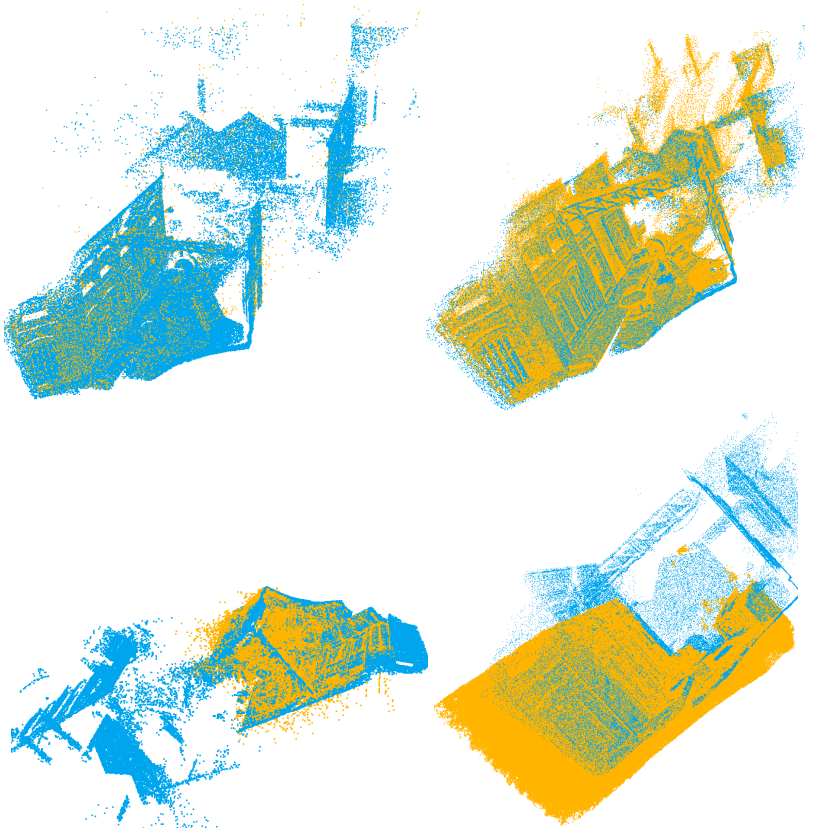


Figure 24: TartanAir, Abandoned Factory; Aligning the estimated point clouds (*orange*) to the ground truth (*blue*) via ICP Registration. *Top Left*: COLMAP Sparse. *Top Right*: COLMAP Dense. *Bottom Left*: DROID-SLAM. *Bottom Right*: NeRF-SLAM.

4.2 KITTI Visual Odometry / SLAM Evaluation 2012

Table 2 provides a detailed view of the evaluation metrics for the KITTI sequence, in which the vehicle traverses a curve.

Table 2: Evaluation metrics for estimated camera poses and reconstructed point cloud for the KITTI sequence.

	COLMAP	DROID-SLAM	NeRF-SLAM
APE [m] ↓	**0.0797**	0.2801	0.3036
RPE [m] ↓	0.9873	**0.0293**	1.2518
Accuracy [cm] ↓	Sparse: 69.2680 Dense: 104.0494	**45.6620**	61.4194
Completion [cm] ↓	Sparse: 281.3357 Dense: 275.9837	**239.9778**	317.4361
Completion Ratio [$<$ *5cm%]* ↑	Sparse: 2.3883 Dense: **23.3077**	3.9125	4.9065
Completion Ratio [$<$ *20cm%]* ↑	Sparse: 25.0881 Dense: **47.1160**	41.0308	24.5862

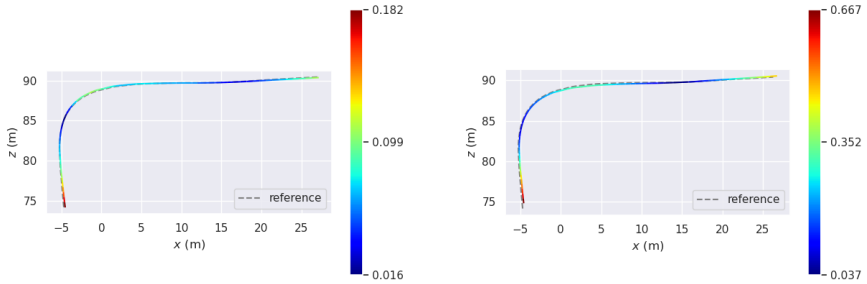
The data shows that COLMAP outperforms the other methods in terms of APE, yielding an RMSE of roughly 8 cm, thus implying a good overall performance. Somewhat unsurprisingly, DROID-SLAM and NeRF-SLAM yielded APE values similar to one another at a 28 cm and 30.4 cm margin respectively. Interestingly, these values are more than 3.5 times larger than the APE generated by COLMAP.

To analyze these results further, the color-mapped plots illustrated in Figure 25 can be examined. In this figure, a bird's eye view perspective of the trajectory is shown, where the start of the trajectory is located at $x \approx -5$. By comparing these plots it becomes evident that even after alignment, there is a significant offset present between the trajectories of DROID-SLAM and NeRF-SLAM and the ground truth. Notably, this offset is more distinct at the start and end points of the trajectories, which is indicated by red colors. Although this effect is somewhat apparent in COLMAP's plot, particularly at the start point of the trajectory, it is important to note the difference in scaling between the plots. For COLMAP, the maximum error (shown as red) yielded was 0.182m, whereas for DROID-SLAM and NeRF-SLAM, the corresponding errors are 0.667m and 0.487m respectively.

It is possible that the inferior APE results of the deep learning-based methods are related to the characteristics of the images in the KITTI sequence, and more specifically the image quality. A sample from this sequence is shown in Figure 27, showing images captured at three different (non-sequential) timesteps. It is notable that the color representation varies from frame to frame, despite being captured within seconds of each other. This variation, such as the shift in sky color from light blue in the first frame, to white in the third frame, could potentially obstruct the pose estimation process for the deep learning-based methods due to the downscaling that these methods perform.

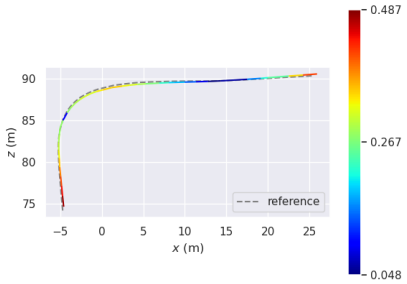
During the DROID-SLAM reconstruction, the KITTI images are downscaled from $W \times H : [1241, 376] \rightarrow [800, 240]$. Similarly, in NeRF-SLAM, the images are resized to $[640, 341]$. It is plausible that this dimension reduction could potentially present a challenge in the feature-matching process, as details in the image then are represented by a lesser number of pixels. Now add to this the fact that the colors may rapidly shift from frame to frame, and consequently, the matching process becomes considerably more challenging. Given the importance of this process for estimating accurate poses, it is plausible that these factors contribute to the high APE error for DROID-SLAM and NeRF-SLAM. For the case of COLMAP, no resizing of images is performed, and the method could thus be more robust to handle images with poor color representation.

In terms of RPE results, Table 2 shows RPEs of 0.99 m, 0.029 m, and 1.252 m for COLMAP, DROID-SLAM, and NeRF-SLAM respectively. The fact that DROID-SLAM achieves a low RPE value, despite its higher APE value, suggests that the estimated trajectory has a high local consistency, but a low accuracy globally. That is to say, the relative movements and orientations between successive poses are well-represented by the estimations, but the accuracy of the trajectory as a whole is lower.



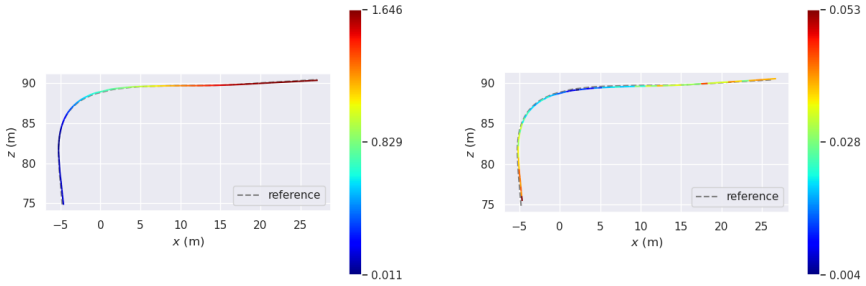
(a) COLMAP

(b) DROID-SLAM

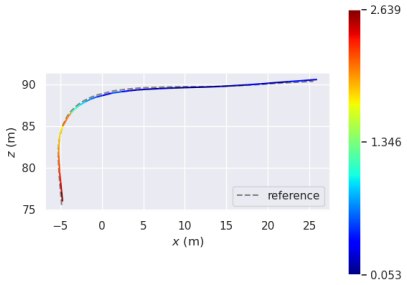


(c) NeRF-SLAM

Figure 25: RMSE(APE) for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.



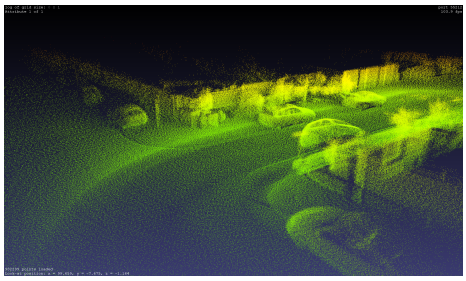
(a) COLMAP (b) DROID-SLAM



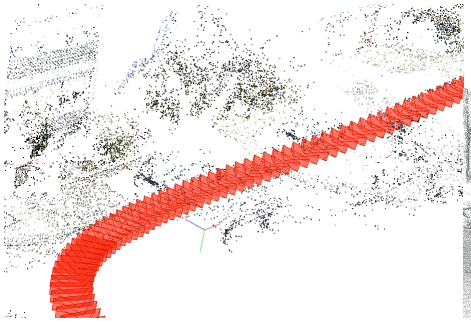
(c) NeRF-SLAM

Figure 26: RMSE(RPE) for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.

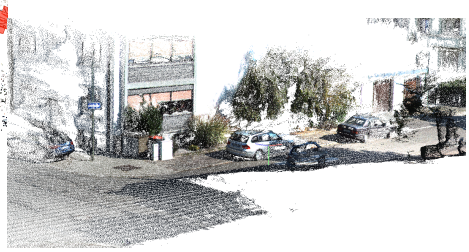
Figure 27: Three representative images from the KITTI sequence: the top image corresponds to the first image of the sequence; the second image captures the perspective from the car starting turning to the right; the final image shows the perspective from the car after it has navigated through the curve.



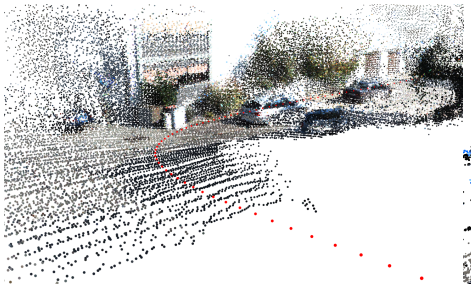
(a) Ground Truth



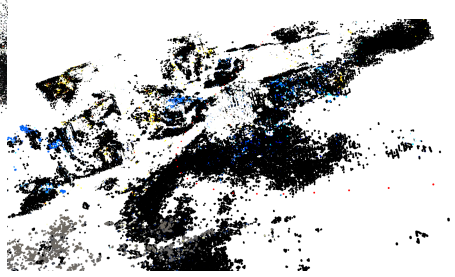
(b) COLMAP Sparse Reconstruction



(c) COLMAP Dense Reconstruction



(d) DROID-SLAM



(e) NeRF-SLAM

Figure 28: Point cloud figure. Ground truth, followed by the reconstructions generated by the pipelines.

In terms of the point cloud evaluation, we can perform the same comparative analysis on alignment and the reconstructed point clouds similar to the one conducted for the TartanAir results. See table 2. Since this is a large-scale real-life sequence, which introduces a range of new possible sources of error, the completion threshold was increased to reflect the scalability (object size) better, but also to examine how much the completion ratio would increase. Note that *cm* is still used as the length unit for the margin of error. DROID-SLAM seems to have the overall best accuracy by a considerable margin at 45.7 cm, the best completion score at 240.0 cm, and a satisfactory completion ratio with an increased threshold of 41.0%. In fact, DROID-SLAM catches up fairly quickly to COLMAP dense reconstruction in terms of completion ratio, where the dense reconstruction has a significantly better low-threshold completion ratio.

NeRF-SLAM outputs the lowest performance scores in some aspects of the point cloud evaluation, such as the lowest completion score(4.9cm). When the threshold is increased, the completion ratio improves for NeRF-SLAM as expected, but at a lesser rate than

DROID-SLAM. Comparing the performance of NeRF-SLAM with DROID-SLAM in terms of completion ratio, this means that there are many highly distant nearest neighbor points generated by NeRF-SLAM relative the ground truth points. This result can be expected due to the bounding box characteristic of NeRF-SLAM. In comparison, the relatively large distances in completion for DROID-SLAM are more evenly distributed, so it greatly benefits from an increased threshold. Once again, the result may vary substantially depending on how the LiDAR-based ground truth is created. These considerations will not however improve the verdict for the NeRF-SLAM reconstruction, as the reconstructed 3D scene from NeRF-SLAM looks undesirable and far from being viable for annotation purposes compared to other algorithms (see 28 (e)). This inefficiency could be attributed to the previously mentioned downsampling procedure that NeRF-SLAM performs, in conjunction with poor image quality.

Since the KITTI dataset is non-synthetic, we are inevitably introduced to new sources of error during the 3D reconstruction, even when obtaining a ground truth point cloud to evaluate against. One issue was addressed previously, where a jointly mounted LiDAR system and a camera must have a synchronous shutter. Such an issue is non-existent in synthetic data generation such as for the TartanAir dataset, since the virtual component representing the RGB camera is located at the exact same location as the depth camera, capturing the image and depth data simultaneously with zero shutter delay.

Another issue when constructing a LiDAR-based ground truth point cloud is the vertical field of view of the LiDAR system, which normally does not match that of a camera. The data for the KITTI sequence was collected using a LiDAR mounted on the roof of a car. Consequently, this placement restricts the LiDAR's ability to capture objects that exceed the height of its mounted position. As a result, the 3D reconstructions may capture the facade of the building, as illustrated in Figure 28, while the ground truth only contains points capturing the sidewalk and other ground-level objects on the left side of the trajectory. Moreover, the LiDAR-based ground truth point cloud represents a 360° view of the sequence, capturing objects outside the field of view of the camera, such as areas behind the car. In contrast, the reconstructed point cloud is limited by the field of view of the camera. This discrepancy is illustrated in Figure 29. Therefore, this will lead to a considerably larger completion distance compared to the results in the synthetic data evaluation (TartanAir) since the ground truth points on the left and front streets are also queried to the completion k-d tree. However, the performance of the reconstruction algorithms can still be compared relative to each other, with a lesser emphasis on the actual numerical values.

The stretching effect that was previously seen in COLMAP's reconstructed point cloud can be observed in this sequence as well. The ambiguous depth estimation is even more intensified here due to the fact that we have quasiparallel viewing rays from the cameras since the car moves in a straight line, before and after the leftmost turn, with little change in camera direction. This means that the apparent parallax occurs sooner at a closer range, leading to a long-distance stretching effect. See Figure 29 (top right).

Figure 29: KITTI sequence; Aligning the estimated point clouds (*orange*) to the ground truth (*blue*) via ICP Registration. *Top Left*: COLMAP Sparse. *Top Right*: COLMAP Dense. *Bottom Left*: DROID-SLAM. *Bottom Right*: NeRF-SLAM.

4.3 Volvo Cars Data 1: Single Camera

The evaluation results for the Volvo Cars Data 1: Single Camera sequence, containing images captured using the forward-looking long-range camera, are presented in Table 3. As noted in Section 3.1.3, the point cloud evaluation is omitted here due to the lack of LiDAR ground truth data for this particular sequence.

Table 3: Evaluation metrics for estimated camera poses and reconstructed point clouds for the first 80 frames in the Volvo sequence.

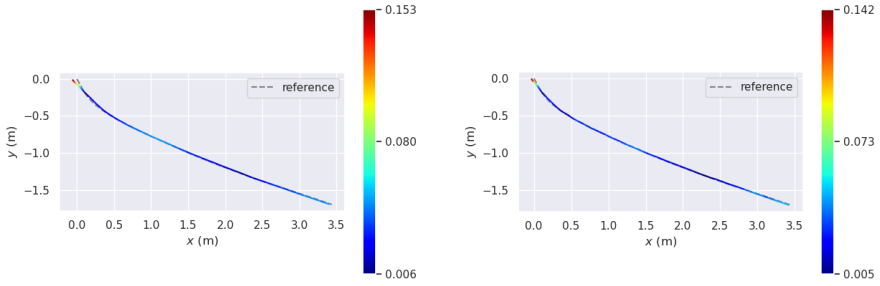
	COLMAP	DROID-SLAM	NeRF-SLAM
APE [m] ↓	0.0444	0.0374	**0.0348**
RPE [m] ↓	**0.5188**	1.1858	4.1197

Among the evaluated methods, NeRF-SLAM yielded the lowest APE error at approximately 3.5 cm, while COLMAP and DROID-SLAM generated comparable errors, at 4.4 cm and 3.7 cm respectively. Compared to the APE results on the KITTI dataset (see Table 2), where the data showed errors close to 30 cm for DROID-SLAM and NeRF-SLAM, it becomes apparent that their performance significantly improved for the Volvo Cars Data 1: Single Camera sequence. It is possible that this increase in performance relates to the improved image quality and lighting conditions. As illustrated in Figure 31, the color representation in this sequence appears to be significantly more realistic than that in the images from the KITTI sequence.

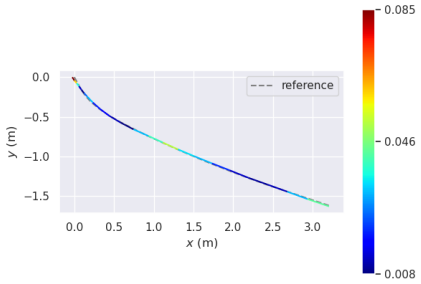
In Figure 30, it can be seen that all methods perform worse at the start of the trajectory (located at $[0, 0]$ in the plots), and stabilizes the errors rather quickly as the car traverses the trajectory.

Furthermore, an analysis of the RPE results from all methods, reveals some common traits with the performance on the TartanAir sequence. Particularly, NeRF-SLAM yielded a significant RPE error (approximately 4.1 m) despite its low APE value, which is presumably related to the exclusive evaluation of keyframes, as previously discussed in Section 4.1. For this 80-frame sequence, NeRF-SLAM generated a total of 25 poses. The first 9 poses correspond to frames $0, 2, 4, \ldots, 20$, followed by poses corresponding to frames $24, 28, 36, \ldots 76$, and finally, a pose at frame 78. The transition from one pose for every other frame, to one pose for every fourth frame (pose 20 to pose 24) contributed to a spike in the RPE, which could explain the significant RMSE value for this particular sequence. In Figure 33, the RPE as a function of pose n estimated by NeRF-SLAM is plotted, where the spike evidently appears at $n = 8$.

COLMAP and DROID-SLAM seem to accumulate more drift as the car progresses down the road, yielding RPE errors of approximately 0.5 m and 1.2 m.



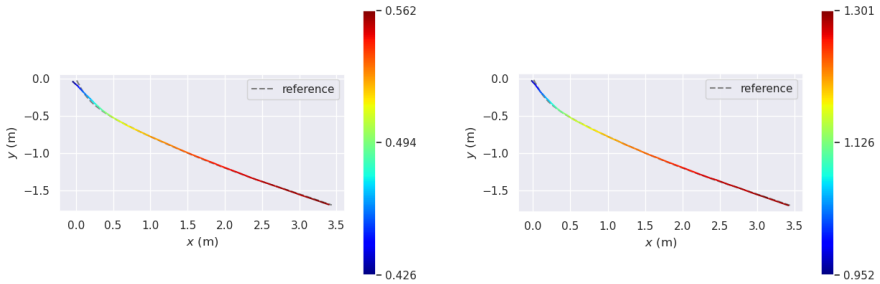
(a) COLMAP (b) DROID-SLAM



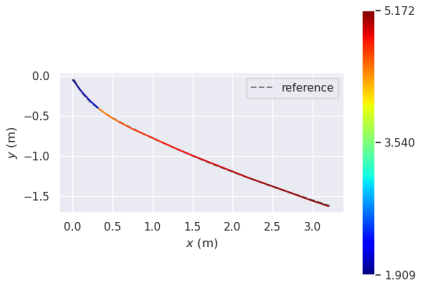
(c) NeRF-SLAM

Figure 30: RMSE(APE) for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.

Figure 31: Three representative images from the Volvo Cars Data 1 sequence: the top image corresponds to the first image of the sequence; the second image captures the perspective from the car as it has driven some distance; the final image shows the perspective from the car after having almost reaching the end of the trajectory.



(a) COLMAP (b) DROID-SLAM



(c) NeRF-SLAM

Figure 32: RMSE(RPE) for COLMAP, DROID-SLAM, and NeRF-SLAM respectively.

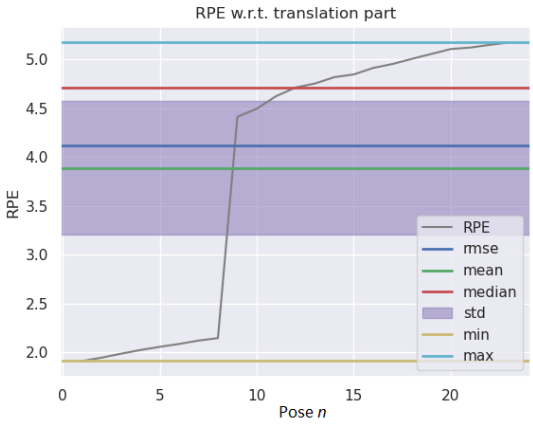
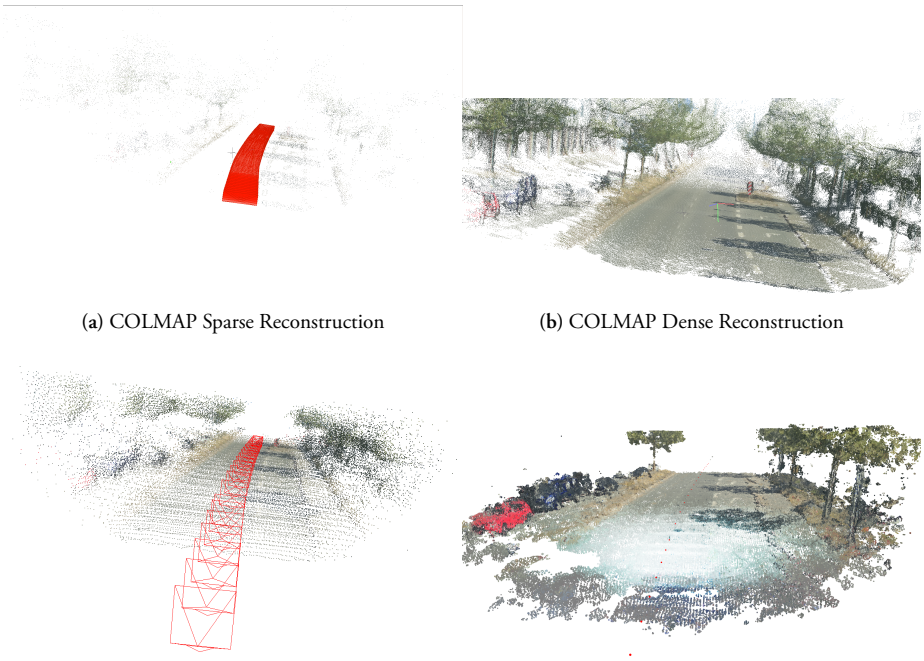


Figure 33: This plot depicts the RPE as a function of pose number n for poses estimated by NeRF-SLAM. The sequence demonstrates a pattern where NeRF-SLAM generates poses every second frame up to $n = 8$, then every fourth frame thereafter. This likely influenced the elevated RMSE of the RPE, one of the statistical metrics displayed in the image.

Figure 34 shows the point clouds generated by each method. In terms of reconstruction ability, NeRF-SLAM improved drastically from the unsuccessful results on the KITTI sequence (see Section 4.2). At first glance, the point cloud generated by NeRF-SLAM seems more accurate than DROID-SLAM's, and possibly even COLMAP's. However, the point cloud's quality is highly dependent on the viewing angle. As shown in Figure 35 (top left), a synthesized view of the scene in its NeRF representation is seen, approximately aligned with the same viewing angle as the figure showing the point cloud (Figure 34). This image shows that NeRF-SLAM managed to generate a rather accurate NeRF representation of this scene. Nevertheless, NeRF methods are limited by the coverage of the scene, meaning that the MLP (see Section 2.3) struggles to accurately learn the scene at angles that are absent in the image set. Consequently, when moving the camera to view directions that are far away from the directions in the training set, such as a side-view or bird's-eye view as seen in Figure 35, the result is considerably worse.

In contrast, it is evident that COLMAP dense reconstruction and DROID-SLAM manages to capture the geometry of the scene more accurately, albeit at different levels. As it has been previously noted, COLMAP succeeds in generating superior point clouds for this sequence as well. It is however important to discuss the runtime difference for the different methods. The generation of the sparse and then the dense COLMAP reconstructions for this 80-frame sequence took approximately $4.3 + 5.2 = 9.5$ minutes on a GeForce RTX 2080. In comparison, DROID-SLAM completed the reconstruction in approximately 20 seconds.

In the context of generating point clouds as training data, whether online or offline, the significantly longer runtime of COLMAP makes it unfeasible to use practically for large-scale projects. It is well-known within the fields of computer vision and machine learning that large datasets are crucial for effective model training. Although it might not be an absolute requirement to generate point clouds for sequences longer than 80 frames, the total runtime for several reconstructions would accumulate quickly, thus limiting its practical usability. DROID-SLAM on the other hand, which is approximately 96.49% faster, presents the possibility of real-time point cloud generation, given the appropriate hardware.



(a) COLMAP Sparse Reconstruction

(b) COLMAP Dense Reconstruction

(c) DROID-SLAM

(d) NeRF-SLAM

Figure 34: Point cloud figure, illustrating the reconstructions generated by the different methods.

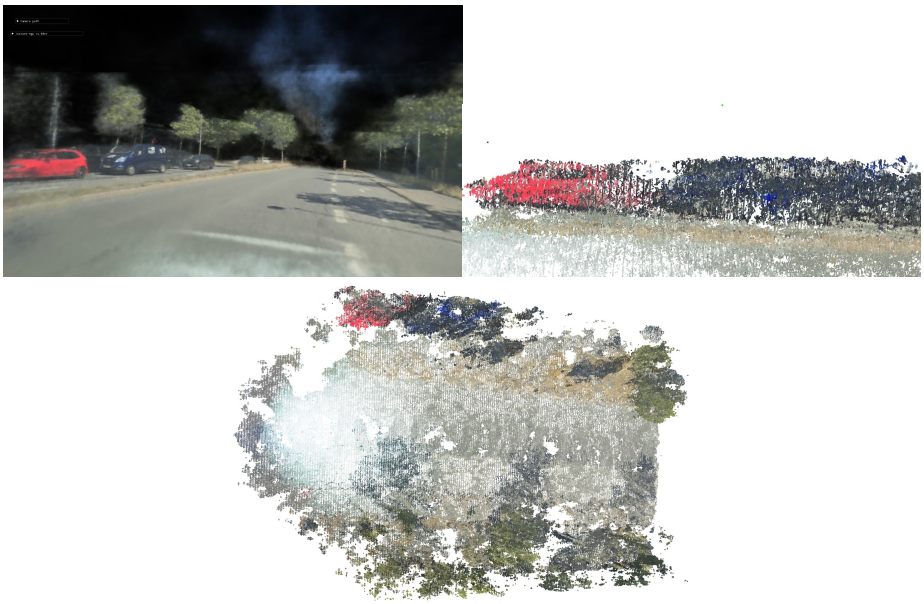


Figure 35: This figure presents three images from the Volvo Data 1: Single Camera sequence. The first image captures the scene in its NeRF representation. The second image shows a side-view of the point cloud, with a focus on the cars present in the scene. The last image provides a bird's eye view of the point cloud.

4.4 Volvo Cars Data 2: Multi-Camera System

Table 4 shows the evaluation results for COLMAP and DROID-SLAM on the Volvo Cars Data 2: Multi-Camera System sequence, in which the car maneuvers a turn. Metrics concerning pose estimation are presented individually for each camera and subsequently averaged to provide a comprehensive measure of overall performance. The first four rows (*FISH_*) present the pose metrics for each short-range fisheye camera, where L, R, F, and B stand for the left, right, front, and back. Subsequently, FC, LRBL, and LRBR correspond to the long-range forward-looking camera, and the two long-range backward-facing fisheye cameras mounted on each side-view mirror. The point clouds, however, were evaluated as one cohesive unified point cloud.

NeRF-SLAM was deliberately left out of this subsection as it failed to generate acceptable outcomes. Further details about this decision are provided in Section 4.5.

Table 4: Evaluation metrics for estimated camera poses and reconstructed point clouds for the Volvo Cars Data 2: Multi-Camera System sequence.

COLMAP	APE [m]↓	RPE [m]↓	DROID-SLAM	APE [m]↓	RPE [m]↓
FISHL	**0.0424**	**0.0538**	*FISHL*	0.0658	0.1502
FISHR	**0.1366**	**0.1065**	*FISHR*	0.1811	0.1154
FISHF	**0.0212**	**0.3783**	*FISHF*	0.1678	0.4064
FISHB	**0.0727**	**0.0830**	*FISHB*	0.1827	0.3932
FC	**0.0386**	**0.0227**	*FC*	0.0694	0.4039
LRBL	**0.0574**	**0.1770**	*LRBL*	0.0646	0.3262
LRBR	0.1005	**0.2771**	*LRBR*	**0.0266**	0.4037
Avg.	**0.0670**	**0.1569**	**Avg.**	0.1083	0.3141
Accuracy [cm]↓	92.4831		*Accuracy [cm]*↓	**61.9032**	
Completion [cm]↓	**69.5067**		*Completion [cm]*↓	504.9397	
Completion Ratio [$<$ *5cm%]*↑	**9.9948**		*Completion Ratio [*$<$ *5cm%]*↑	1.1788	
Completion Ratio [$<$ *20cm%]*↑	**37.2089**		*Completion Ratio [*$<$ *20cm%]*↑	12.8360	

The results show that COLMAP outperformed DROID-SLAM on this sequence in terms of APE by yielding lower values for all cameras except the LRBR camera, with an average of 6.7 cm, compared to DROID-SLAM's average of 10.8 cm. Reflecting on the previous results for the KITTI (refer to Section 4.2), a pattern can be seen once again in which COLMAP tends to outperform DROID-SLAM in pose estimation when the vehicle is turning.

The cameras that yielded the lowest APE values for DROID-SLAM were the two backward-facing long-range cameras, the forward-facing long-range camera, and the left fisheye camera, with RMSE values of approximately 6.5 cm, 2.7 cm, 6.9 cm, and 6.6 cm respectively. This might be attributed to their higher resolutions and positions. As has been seen in the previous results, the camera pose estimation of DROID-SLAM seems to be enhanced when provided higher-quality images that have better color representation.

In terms of RPE, COLMAP, and DROID-SLAM averaged errors at approximately 15.7 cm and 31.4 cm, respectively. Both of these results show a decent estimation of relative camera movement but indicate that drift in the trajectories is present.

For this sequence, there are numerous possible sources of error. To generate a ground truth trajectory, sensors such as the LiDAR, IMU, and GPS/GNSS receivers are utilized (See Section 3.1.3). In order to create a perfect ground truth trajectory for each camera, this

requires all sensors, including the cameras, to be completely synced. However as mentioned before, this can not be guaranteed in practice. Even if the positions of all sensors are known, effects such as rolling shutter might obstruct this process by capturing images with a slight time delta. Consequently, when the sensors calculate the pose at one time step, the images captured by the cameras might be slightly delayed, in relation to each other, but also to the pose-generating sensors.

In addition to a potential dependence related to image quality, it is possible that the pose estimation of DROID-SLAM did not generalize well to some of these sequences. Due to the exclusive training on synthetic images from the TartanAir dataset, it is plausible that the discrepancy is significant between images captured by some of the cameras in this sequence and the images on which DROID-SLAM was trained on.

Furthermore, the undistortion of the fisheye images introduces a rather significant source of error. As mentioned in Section 3.1.3, the images in the provided sequence underwent certain post-processing steps. It is notable that the steps taken are specific to the use case of the images, and the dataset provided for this thesis was not specifically optimized for visual SLAM purposes. When these images were captured and post-processed, a large emphasis was put on low runtimes. Consequently, some aspects of image quality that could have improved the reconstruction accuracy might have been lost out on in this case.

The point cloud evaluation results can be seen in the lower section of Table 4. DROID-SLAM outperforms COLMAP dense reconstruction in terms of accuracy, with a great margin as well, that being 61.9 cm versus 92.5 cm. COLMAP dense reconstruction yields a significantly better completion than DROID-SLAM, with a value of 69.5 cm compared to 504.9 cm of DROID-SLAM. Same goes for their completion ratio, as COLMAP outputs almost 10% with 5-cm threshold, which is considerably higher than 1.2% of DROID-SLAM. This disparity between the accuracy result and the completion results of COLMAP versus DROID-SLAM can be attributed to the fact that the LiDAR-based ground truth point cloud captures a relatively large area of this road scene, such as distant trees and objects, for which COLMAP dense reconstruction benefits the most in terms of completion and completion ratio. See Figure 36. On the other hand, DROID-SLAM estimates points at a higher accuracy, as a result of its distance-based filtering technique which excludes 3D points located at substantial distances. The DROID-SLAM reconstruction can be seen Figure 37 (b)-(e), where reconstructions of distant objects are mostly absent. However the road itself is well-reconstructed.

We may notice how the reconstructions from the fisheye cameras somewhat affect the overall 3D structure of the point cloud after transforming each reconstructed point cloud from all 7 cameras (pinhole + fisheyes) to one common coordinate system. For example in DROID-SLAM, there is a shifting effect that can be faintly observed in some objects, such as the rightmost white vehicle in 37 (b), where it looks like there is a point cloud of the same car overlaid on itself from another point cloud, where one is outputted from fish-eye cameras. We can also observe the rightmost white parking line in 37 (e) to be slightly curved.

The COLMAP dense reconstruction seems to yield a better color palette and contrast, with more details captured in the reconstructed point cloud, compared to the DROID-SLAM reconstruction. See Figure 38. For example, we see dense representations of trees and bushes on the right side of the trajectory, which can be useful for annotation purposes. We can also witness the same stretching effect of the MVS reconstruction. This effect is most noticeable from a bird's eye view, Figure 38 (b). The stretching effect occurs at a fairly large distance from the trajectory, to the point where it is almost unnoticeable from

other viewpoints. See Figure 38 (c), (d), (e) for comparison. So the stretched-out segment, located mostly behind the trees, does not occur on the road, nor on the sidewalks close to the trajectory. So for example, the right side of the trajectory (before the turn) can be easily identified, and potentially annotated, as a parking lot without having the stretching segment negatively influencing the scene.

While the unified DROID-SLAM point cloud may not look as impressive as COLMAP dense reconstruction, we have to keep in mind DROID-SLAM estimates the 3D points and mapping in a real-time manner, and it is considerably faster than COLMAP. The difference in runtime intensifies when large-scale datasets of high-quality images are used. For instance, the current DROID-SLAM point cloud can be converted into a mesh, and the accumulated runtime would still be lower than COLMAP dense reconstruction. See Figure 39.

We can clearly see some of the characteristics from the COLMAP dense reconstruction in the meshed representation of DROID-SLAM's reconstruction. Objects such as the parked cars, the green hedge and parking lines are clearly visible in this format. Thus, it could be argued that the meshed point cloud from DROID-SLAM is a viable option to COLMAP's dense reconstruction in terms of annotation capabilities. The COLMAP reconstruction could perhaps be converted into a mesh and then compared to the DROID-SLAM mesh for a more fair comparison, but since the dense point cloud is already based on MVS, the annotatability of objects only would improved by a small amount. Furthermore, given the density and size of the COLMAP dense point cloud, meshification would be a considerably time-consuming process. This automatically renders a meshifying post-process not feasible for this algorithm.

Another visible difference we notice in their reconstructions is that the COLMAP dense point cloud appears to have two distinct surfaces at the same plane, but with a void gap between them. This effect is the most noticeable in Figure 38 (c), (d) and (e). Meanwhile the meshified road in DROID-SLAM has preserved the flat surface property of a real-life street. This means that we have, for the most part, one single cohesive road surface, despite having utilized both fisheye and pinhole cameras, which can be easily annotated. See Figure 39 for an illustration of the 3D road scene in a mesh format.

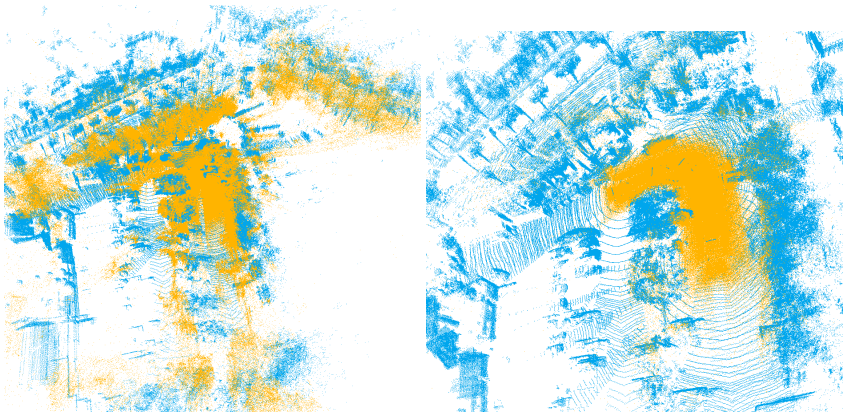
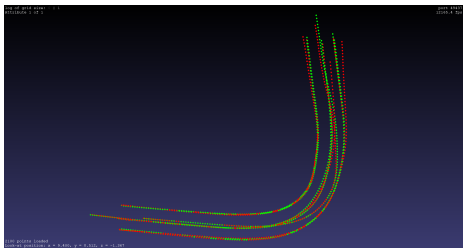
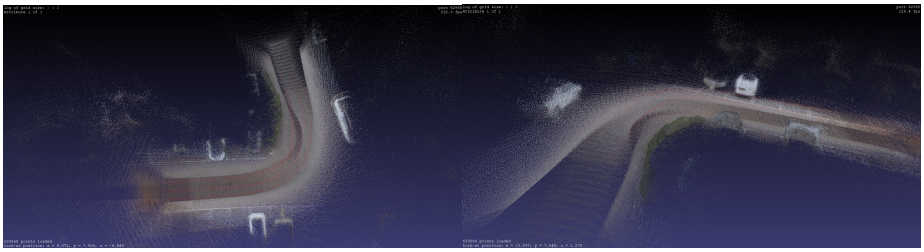


Figure 36: Volvo multi-cam system sequence; the alignment of the estimated point clouds (*orange*) to the ground truth (*blue*). The point clouds are downscaled for visualization purposes. *Left*: COLMAP dense. *Right*: DROID-SLAM.

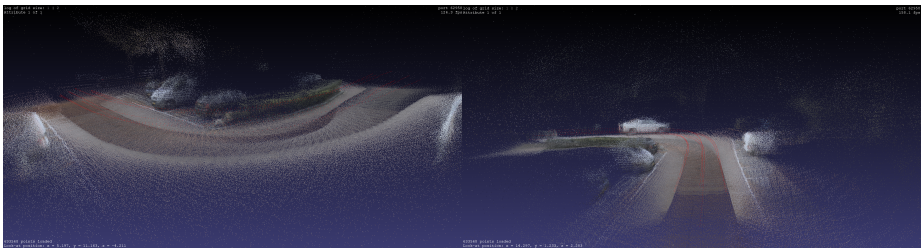


(a) Bird's eye view of ground truth trajectories from each camera in green, and corresponding estimations in red.



(b) Bird's eye view.

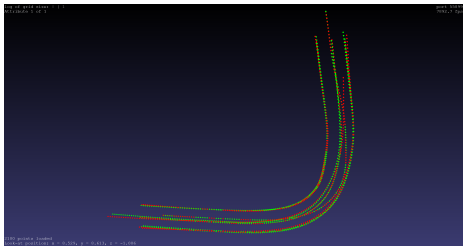
(c) Perspective from the end of the trajectory.



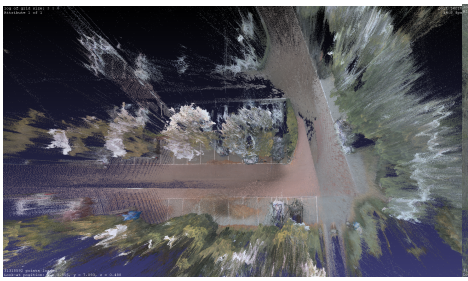
(d) Curve perspective

(e) Forward perspective

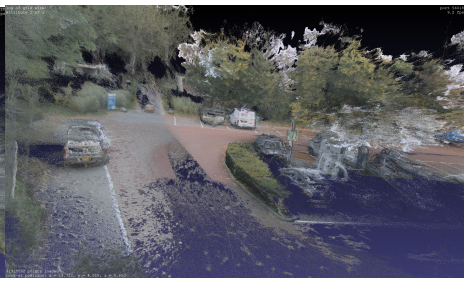
Figure 37: DROID-SLAM: Merged point cloud of the Volvo Data 2 sequence, generated from individual reconstructions from all 7 cameras.



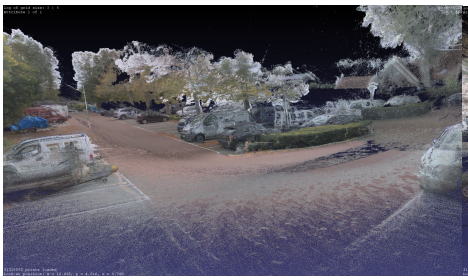
(a) Bird's eye view of ground truth trajectories from each camera in green, and corresponding estimations in red.



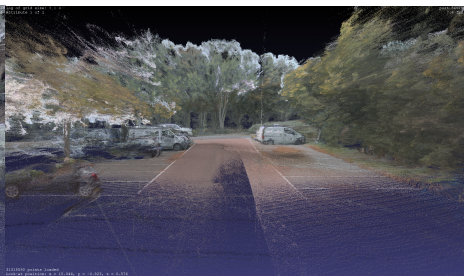
(b) Bird's eye view.



(c) Perspective from the end of the trajectory.

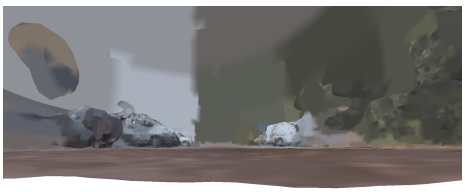


(d) Curve perspective

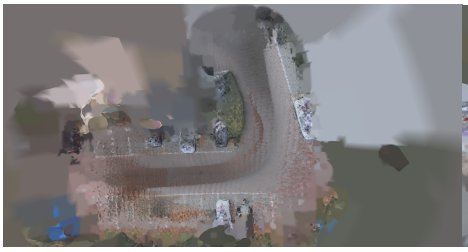


(e) Forward perspective

Figure 38: COLMAP Dense Reconstruction: Merged point cloud of the Volvo Data 2 sequence, generated from individual reconstructions from all 7 cameras.

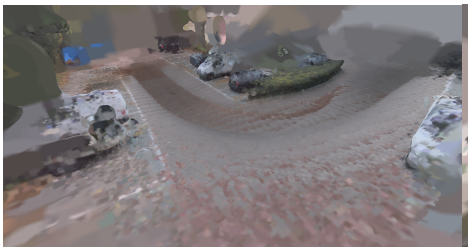


(**a**) Low angle perspective.

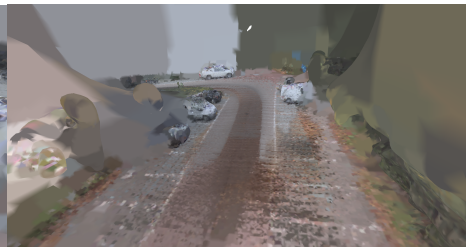


(**b**) Bird's eye view.

(**c**) Perspective from the end of the trajectory.



(**d**) Curve perspective.



(**e**) Forward perspective.

Figure 39: Meshified point cloud for all cameras, generated via DROID-SLAM.

4.5 NeRF-SLAM Limitations

Figure 40 shows some results generated by NeRF-SLAM using the right fisheye camera. The top image shows the scene in its NeRF representation, while the bottom two show the final point cloud for this sequence. In the images, a white van has been marked as somewhat of a landmark to allow for easier navigation across the images. As illustrated by the figure, the point cloud generated by NeRF-SLAM failed significantly, both in terms of capturing scene geometry and accurately representing colors. Reflecting back on the results from the KITTI dataset, similar traits in the point clouds become apparent where white, blue, and black colors are dominant.

Due to the poor performance of NeRF-SLAM on the Volvo Cars Data 2: Multi-Camera System sequences, the choice to omit including this method in the evaluation was obvious. Attempts were made to reconstruct the scene using images captured by other cameras, and by changing the values of parameters such as the `integer_depth_scale`, but similar results were yielded. Prior to these results, it could be argued that NeRF-SLAM might contend with the other algorithms in terms of exploration value, given its promising point cloud reconstructions on the TartanAir and Volvo Cars Data 1 Sequence: Single Camera. However, these results clearly indicate that the other two methods offer greater potential for Volvo Cars' use case than NeRF-SLAM.

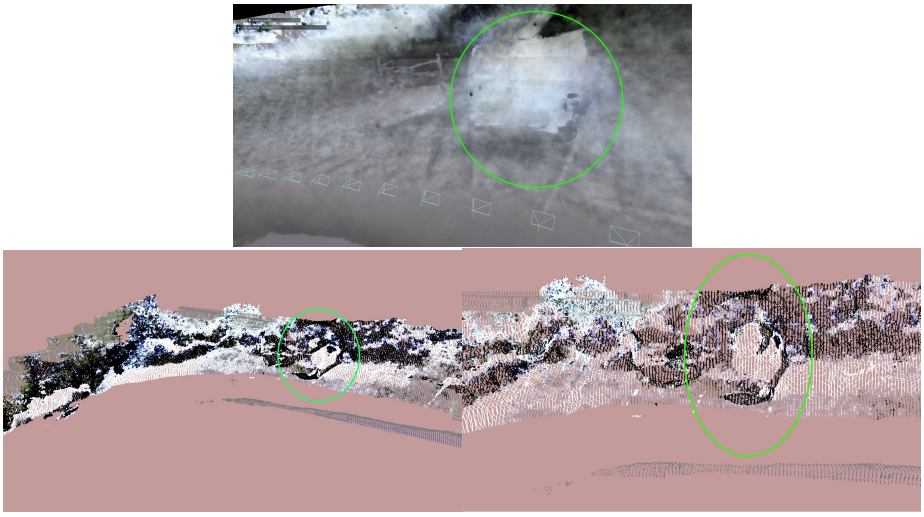


Figure 40: The figure illustrates the scene of the images captured using the right fisheye camera, as generated by NeRF-SLAM. The top image shows the scene in its NeRF representation. The two figures in the bottom row show the resulting point cloud from different angles, and with different point sizes.

5 Conclusions

The off-the-shelf implementations of COLMAP, DROID-SLAM, and NeRF-SLAM performed with varying efficiency across the different datasets used in this thesis. COLMAP generally outperformed the other methods when considering metrics from both the camera pose and point cloud evaluations. Although the performances on the public datasets provided important insights about what affected the different methods' reconstruction abilities, the primary emphasis should be put on the Volvo Cars Data 2: Multi-Camera System, as this reflects the Volvo Cars use case fully. From those results, it can be concluded that the classical 3D reconstruction approach taken in COLMAP yielded the most accurate pose estimations, while simultaneously generating dense, annotatable point clouds. However, due to its extensive reconstruction runtime, deploying COLMAP on larger datasets or to generate training data for large-scale deep learning applications is not practical. Therefore, it can be concluded that among the three algorithms evaluated in this thesis, COLMAP ranks second in terms of potential for Volvo Cars' use case.

The algorithm that demonstrated the most potential due to its ability to generate accurate point clouds and camera poses in near real-time was DROID-SLAM. The point clouds, fortified by their conversion into a mesh representation, clearly satisfied the condition of being annotatable. Despite generating larger pose estimation errors, and sparser point clouds for this sequence, it is important to recognize that DROID-SLAM was used in a predominantly off-the-shelf fashion. Consequently, there is potential for fine-tuning parameters to better align with Volvo Cars' setup, or to train DROID-SLAM further using custom datasets. This underlines one of the key insights from this comparative study: the viability of deep learning-based visual SLAM methods for Volvo Cars' use case, which can be further explored upon.

NeRF-SLAM showed the lowest performance overall out of the three methods evaluated in this thesis. While producing significant dense point clouds on datasets with appropriate image characteristics, NeRF-SLAM failed to produce adequate reconstructions on the Volvo Cars Data 2: Multi-Camera System dataset. Given that NeRF-SLAM was released as recently as late 2022, it is plausible that the limitations brought up in this thesis will be addressed, or indirectly solved, in future publications of NeRF methods. In its current state, however, NeRF-SLAM does not seem like the most viable option for Volvo Cars' to pursue.

In terms of exploring the effectiveness of utilizing multi-camera systems, it can be concluded that the introduction of several cameras improved the 3D reconstruction capabilities of the methods. This becomes evident when comparing the results from the two Volvo Cars sequences.

In retrospect, one aspect that affects the results of the evaluation pipeline which could have been improved is how the ground truth point cloud was constructed. While this was not an issue for the TartanAir sequence since synthetic data was used, it is certainly an important source of error that is worth addressing for the real-world data, such as the KITTI and Volvo Cars datasets. In future work, the ground truth creation could be improved by incorporating some level of pruning to ensure that only LiDAR points within the captured field of view of the cameras are selected before transforming them to a common coordinate system. Furthermore, the offset that occurred in the LiDAR-based point clouds could have been compensated for as a step before the final transformation to a global coordinate system.

References

[1] Carl Olsson. *Lecture Notes in Computer Vision*. Lund University, 2022.

[2] Deepak Geetha Viswanathan. Features from accelerated segment test (fast). In *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*, pages 6–8, 2009.

[3] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, pages 778–792. Springer, 2010.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[5] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[6] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[8] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras, 2022.

[9] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. *CoRR*, abs/2003.12039, 2020.

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding, 2022.

[12] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, volume 3, pages 47–54, 2003.

[13] Antoni Rosinol, John J. Leonard, and Luca Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields, 2022.

[14] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. Tartanair: A dataset to push the limits of visual slam, 2020.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite, 2012.

[16] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, oct 2015.

[17] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R. Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam, 2022.

[18] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison. imap: Implicit mapping and positioning in real-time, 2021.

[19] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[20] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[21] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[22] Songrit Maneewongvatana and David M. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. *ArXiv*, cs.CG/9901013, 1999.

[23] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, page 0, 2006.

Appendices

A SfM Ambiguity with Camera Intrinsics

Assume that we omit the camera intrinsics, i.e. we use uncalibrated cameras for Structure-from-Motion. The camera matrix is then $K_i \begin{bmatrix} R_i & t_i \end{bmatrix} = \begin{bmatrix} A_i & \tau_i \end{bmatrix}$, where K_i is the (unknown) intrinsics for camera i, which entails A_i being an unknown 3×3 matrix (not necessarily a rotation matrix). Now suppose we have found the 3D points $\mathbf{X}_j$ and camera matrices P_i such that the SfM camera equation $\lambda_{ij} \mathrm{x}_{ij} = P_i \mathbf{X}_j$ holds. Without the provided prior data, we can immediately find another solution to the camera equation by adding a projective transformation, i.e. inserting a 4×4 transformation matrix T, which need not to be an Euclidean, similarity or even an affine matrix, meaning the fourth row is arbitrary, not necessarily zeros and 1. We can reformulate the camera equation to find new solutions and thus new reconstructions, both poses and 3D points:

$$\lambda_{ij} \mathrm{x}_{ij} = P_i \mathbf{X}_j = P_i I \mathbf{X}_j = P_i (T T^{-1}) \mathbf{X}_j = (P_i T)(T^{-1} \mathbf{X}_j) = \tilde{P}_i \tilde{\mathbf{X}}_j, \qquad (24)$$

where $\tilde{\mathbf{X}}_j$ and $\tilde{P}_i$ are the newly obtained 3D points and poses if we were to find a solution for the projective transformed camera equation. For an illustration, See Figure 41.

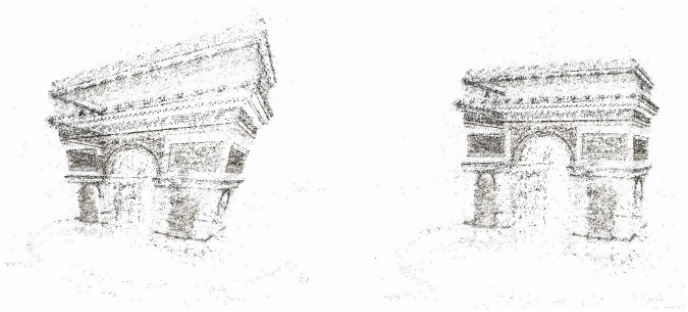


Figure 41: Two different 3D Reconstructions of Arc de Triomphe, but both provide the same projections, i.e. re-projecting the 3D points onto image planes match the ground truth 2D points. On the left, it looks distorted since angles or even parallel lines are not necessarily preserved in a projective reconstruction (this can be signified by the fourth row of a transformation matrix being arbitrary).

So without the known calibration, the solution is not only non-unique, but the solutions can be any projective transformation of our reconstruction, which is over-ambiguous. Supplying known camera intrinsics helps reducing this ambiguity by putting a constraint on the obtained camera matrices. More specifically, the SfM camera equation is deduced to:

$$\begin{aligned} \lambda_{ij} \mathrm{x}_{ij} = P_i \mathbf{X}_j = K_i \begin{bmatrix} R_i & t_i \end{bmatrix} \overset{\cdot K^{-1}}{\Longrightarrow} \lambda_{ij} K^{-1} \mathrm{x}_{ij} = K^{-1} K_i \begin{bmatrix} R_i & t_i \end{bmatrix} \mathbf{X}_j \Rightarrow \\ \Longrightarrow \lambda_{ij} \tilde{\mathrm{x}}_{ij} = \begin{bmatrix} R_i & t_i \end{bmatrix} \mathbf{X}_j \end{aligned} \qquad (25)$$

If we try to perform the same projective transformation as in 24, we quickly realize that it may violate the Euclidean/calibrated camera equation in 25, since multiplying $P_i = K_i \begin{bmatrix} R_i & t_i \end{bmatrix}$ with an arbitrary T may not necessarily yield a (new) rotation matrix in the first 3×3 block, which was the constraint for Euclidean reconstruction 25.

Now that we managed to contain the projective ambiguity, that being distortion, we can examine the uniqueness of the solution to the Structure-from-Motion problem. Instead of an arbitrary projective transformation matrix, we perform the same ambiguity test 24 using a similarity matrix T. The transformed (calibrated) camera equation is as follows:

$$\lambda_{ij}\tilde{\mathrm{x}}_{ij} = \begin{bmatrix} R_i & t_i \end{bmatrix} \begin{bmatrix} s\hat{R} & \hat{t} \\ 0 & 1 \end{bmatrix} T^{-1}\,\mathbf{X}_j = \overbrace{\begin{bmatrix} sR_i\hat{R} & R_i\hat{t} + t_i \end{bmatrix}}^{\tilde{P}_i} \tilde{\mathbf{X}}_j \tag{26}$$

The new camera matrix $\tilde{P}_i$ is valid since the first 3×3 block is the product of two rotations and thus, a new rotation matrix (with scaling). This implies that the 3D reconstruction can still be rotated, translated and re-scaled, without any projective distortion however as we saw in Figure 41. We can interpret this as if the scene is reconstructed in an arbitrary equidistantly spaced (linearly scaled) coordinate system, but the overall shape (i.e. angles) of the 3D object is still preserved, with ideally no projective ambiguity distortion. See Figure 41 (Right).

This is a trade-off we accept during our comparative analysis. More precisely, we are given the camera intrinsics, and provide such information as prior to our 3D reconstruction methods, and instead we tackle this specific ambiguity via applying suitable alignment methods (ICP Registration) on our 3D reconstructions before performing distance-sensitive evaluations on the point clouds.

B COLMAP Automatic Reconstruction and Output Files

Selecting GUI, we can choose a so-called *Reconstruction* > *Automatic Reconstruction* as a quickstart, usually aimed at beginners and intermediates. Here only the most essential parameters are specified, namely the *Image folder*, *Workspace folder*, *Quality* with limited implicit options (*Low*, *Medium*, *High* etc.) and a few other optional fields, such as Dense reconstruction and assumption of *shared* intrinsics. We then press on *Run* and COLMAP will perform all the necessary steps, from feature detection to sparse reconstruction and Bundle Adjustment. The process should be straightforward, and all the reconstruction steps can been seen in a log. A snippet of this procedure can be seen in Figure 42.

All COLMAP output, regardless of what options and parameters we use, will be stored in the previously specified working directory. For *Automatic Reconstruction*, COLMAP creates a database with all extracted data, and a subfolder named *sparse*, which contains three essential output files for estimated camera poses, intrinsics and of course, the reconstructed point cloud. These binary files are called `images.bin`, `cameras.bin` and `points3D.bin`, respectively. The project configuration file `project.ini` can also be found here, which can be later imported via *File* > *Open project* to open an existing project. An additional step we always choose to perform, irrespective of how we use COLMAP, is to select *File* > *Export model as text*. This will save the same reconstruction output information as before, but now in text format. The three files `cameras.txt`, `images.txt`, and `points3D.txt` display the relevant information in plain text, which can be easily comprehended, extracted and hence, used as input files in 3D evaluation pipeline.

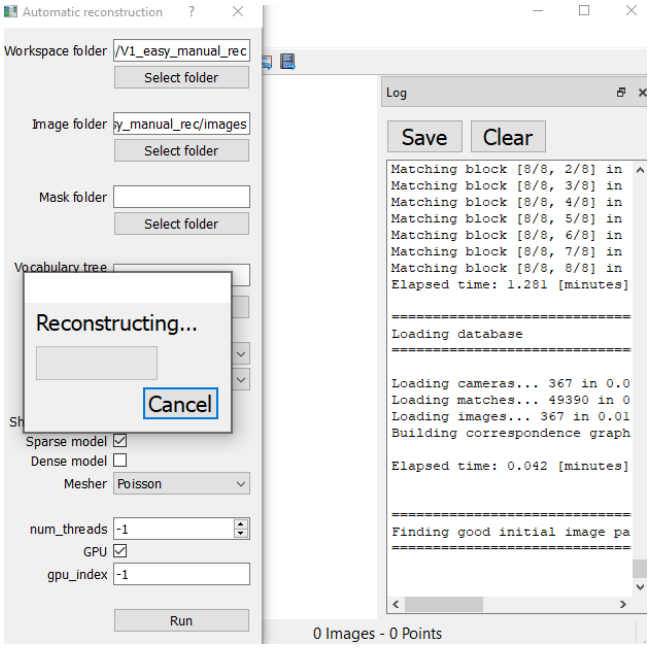


Figure 42: An excerpt from the GUI environment of COLMAP during reconstruction. The pop-up dialog appears after selecting *Reconstruction* > *Automatic Reconstruction*. A log of the entire 3D reconstruction process can also be viewed on the right side of the environment.

Master's Theses in Mathematical Sciences 2023:E35
ISSN 1404-6342

LUTFMA-3510-2023

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

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