Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Evaluating and Improving Depth Map Consistency in 3D Reconstructions Recorded by iPad Pro

Jan Ferbr

Supervisor: Ing. Michal Polic, Ph.D.
Study program: Open Informatics
Specialization: Computer Vision and Image Processing
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Ferbr Jan**	Personal ID number:	**483695**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Vision and Image Processing**		

II. Master’s thesis details

Master’s thesis title in English:

Evaluating and Improving Depth Map Consistency in 3D Reconstructions Recorded by iPad Pro

Master’s thesis title in Czech:

Evaluace a zlepšení konzistence hloubkových map nahrávaných pomocí iPad Pro

Guidelines:

1. Familiarize yourself with the developed RGB-D recording app on the iPad Pro 2023.
2. Validate and convert coordinate systems into a single one.
3. Record real-world indoor/outdoor environments and evaluate the consistency of merged dense point clouds.
4. Compose a dense point cloud from RGB-D images and iPad tracking for initial quality assessment.
5. Identify and visualize issues causing inconsistencies in dense point clouds.
6. Utilize external SfM techniques for more accurate camera poses and re-evaluate point cloud consistency.
7. Investigate state-of-the-art solutions for improving depth map consistency and list related literature.
8. Explore neural network approaches to enhance depth map quality and consistency

a) use pixelwise approach and artificially generated Groud Truth data,

b) [BONUS] - define a loss function based on the reprojection error between multiple depthmaps,

- use a CNN assuming patch of the RGB-D pixels reprojected from multiple images.

Bibliography / sources:

[1] SWong, Alex, and Stefano Soatto. "Unsupervised depth completion with calibrated backprojection layers." Proceedings of the IEEE/CVF ICCV 2021.

[2] Kerbl, Bernhard, et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering." ACM Transactions on Graphics 42.4 (2023).

[3] Schonberger, Johannes L., and Jan-Michael Frahm. "Structure-from-motion revisited." CVPR2016.

Name and workplace of master’s thesis supervisor:

Ing. Michal Polic, Ph.D. Applied Algebra and Geometry CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **22.01.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Michal Polic, Ph.D.	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to sincerely thank my supervisor Michal Polic for all the advices and the time he was willing to offer me. Also I would like to thank my family, my friends and my girlfriend for moral support and never ending motivation during the writing process.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

24. May 2024

Abstract

This thesis explores a new research direction of enhancing the consistency of depth maps in 3D reconstructions, by using machine learning technologies. High-quality 3D reconstructions require advanced correction techniques to address inaccuracies of depth maps captured by low-cost devices. The common approach is to statistically select the most probable surface by averaging measured values, e.g., Signed Distance Function (SDF), and not assume dependence on depth maps acquisition method, e.g., by LIDAR or Multi-View Stereo (MVS). This work introduces two neural network-based approaches, the pixel-wise and convolutional neural network to refine depth map quality. The pixel-wise approach processes depth maps on a per-pixel basis, optimizing depth values across multiple views to enhance consistency. Meanwhile, the convolutional approach utilizes a convolutional neural network to ensure depth accuracy and consistency from different viewpoints, effectively merging depth data from LiDAR and Multi-View Stereo systems. Experimental results demonstrate that these methods improve the consistency of depth maps.

Keywords: Depth map enhancement, Neural network, Machine Learning, Computer vision, Lidar, Multi view stereo, 3D reconstruction, RGB-D scanning

Supervisor: Ing. Michal Polic, Ph.D.
Contact: michal.polic@cvut.cz

Abstrakt

Tato diplomová práce zkoumá nový výzkumný směr zvyšování konzistence hloubkových map ve 3D rekonstrukcích pomocí technologií strojového učení. Vysoce kvalitní 3D rekonstrukce vyžadují pokročilé korekční techniky k řešení nepřesností hloubkových map zaznamenaných nízkonákladovými zařízeními. Běžným přístupem je statisticky vybrat nejpravděpodobnější povrch průměrováním naměřených hodnot, např. pomocí Signed Distance Function (SDF), a nepředpokládat závislost na metodě získávání hloubkových map, např. pomocí LIDAR nebo Multi-View Stereo (MVS). Tato práce představuje dva přístupy založené na neuronových sítích, pixelový a konvoluční, pro zlepšení kvality hloubkových map. Pixelový přístup zpracovává hloubkové mapy na bázi jednotlivých pixelů, optimalizuje hloubkové hodnoty napříč více pohledy a zvyšuje tak konzistenci. Mezitím konvoluční přístup využívá konvoluční neuronovou síť k zajištění přesnosti a konzistence hloubky z různých pohledů, efektivně slučuje hloubková data z LIDAR a Multi-View Stereo systémů. Experimentální výsledky ukazují, že tyto metody zlepšují konzistenci hloubkových map.

Klíčová slova: Vylepšení hloubkové mapy, Neuronová síť, Strojové učení, Počítačové vidění, Lidar, Multi view stereo, 3D rekonstrukce, RGB-D skenování

Překlad názvu: Evaluace a zlepšení konzistence hloubkových map nahrávaných pomocí iPad Pro

Contents

Chapter 1

Notation

In this document, specific notational conventions are employed to represent mathematical and conceptual entities crucial for clarity and precision in discussing algorithms and theories. Vectors and matrices are denoted by small bold letters, e.g., $\mathbf{v}$ for vectors, and capital letters, e.g., $\mathbf{M}$ for matrices, with special matrices such as the calibration matrix $\mathbf{K}$, projection matrix $\mathbf{P}$, rotation matrix $\mathbf{R}$, translation vector $\mathbf{t}$, and the identity matrix $\mathbf{I}$. Points in 2D and 3D space are represented as $\mathbf{x}$ and $\mathbf{X}$ (we denote 3D points the same like matrices) respectively, with their homogeneous coordinates given by $\underline{\mathbf{x}}$ and $\underline{\mathbf{X}}$. Elements within vectors and matrices are accessed by subscripts, such as $\mathbf{v_j}$ for the j-th element of a vector and $\mathbf{P_{ij}}$ for the (i, j) element of a matrix. A special 4x4 transformation matrix $\mathbf{E}$ combines a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ in a specific format that includes the identity for transformation operations.

Abbreviations such as SfM (Structure from Motion), MVS (Multi-View Stereo), RGB-D (Red, Green, Blue - Depth), GT (Ground Truth), NN (Neural Network), SGD (Stochastic Gradient Descent), and MSE (Mean Squared Error) are used to succinctly convey concepts related to image processing and machine learning.

$\mathbf{v}$	Vector
$\mathbf{M}$	Matrix
$\mathbf{K}$	Calibration matrix
$\mathbf{P}$	Projection matrix
$\mathbf{R}$	Rotation matrix
$\mathbf{t}$	Translation vector
$\mathbf{I}$	Identity matrix
$\mathbf{x}$	2D point
$\mathbf{X}$	3D point
$\underline{\mathbf{x}}$	Homogeneous coordinates of a 2D point
$\underline{\mathbf{X}}$	Homogeneous coordinates of a 3D point
$\mathbf{v_j}$	j-th element of vector $\mathbf{v}$
$\mathbf{P_{ij}}$	element i,j of matrix $\mathbf{P}$
$\mathbf{E}$	4x4 matrix in form $\mathbf{E} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0_3}^\top & 1 \end{pmatrix}$

SfM	Structure from Motion
MVS	Multi-View Stereo
RGB-D	Red, Green, Blue - Depth
GT	Ground Truth
NN	Neural Network
CNN	Convolutional Neural Network
SGD	Stochastic gradient descent
MSE	Mean Squared Error
MAE	Mean Absolute Error
GPU	Graphics processing unit

Chapter 2

Introduction

The field of 3D reconstruction has been significantly impacted by the advent of RGB-D scanning technologies, such as those available on devices like the iPad Pro 2023. Despite technological advances, challenges such as inconsistencies and inacuracies in 3D reconstructions persist. High-quality 3D models are typically reliant on expensive scanning equipment, limiting accessibility. This master thesis seeks to address the challenge of achieving high-quality 3D reconstructions with scanners that are broadly accessible to the public. We aim to explore and propose methods to correct inaccuracies in scanned 3D data, thereby enhancing the consistency and quality of the models.

2.1 Scope

This thesis primarily focuses on the development and refinement of algorithms for 3D reconstruction to address and resolve the inconsistencies and inaccuracies observed in depth maps an point clouds. We aim to critically analyze reconstructions obtained from RGB-D images, identify potential issues, and suggest improvements to enhance the 3D reconstruction quality. Our efforts concentrate on improving the depth maps obtained from the LiDAR scanner of the iPad Pro 2023 and those generated using the Multi-View Stereo (MVS) algorithm. Figures 2.1 and 2.2 show examples of 3D reconstructions errors made using both approaches. In 2.1 we can see artifacts in the 3D reconstruction, specifically deformation of windows. On the other hand, in 2.2 we can see holes in the 3D reconstruction either caused by lack of texture in given areas or insufficient number of images used.

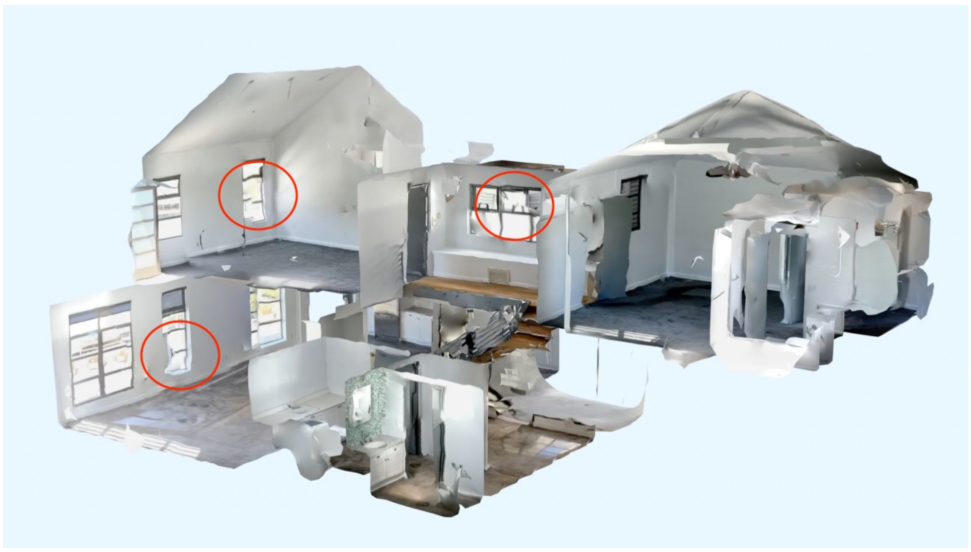


Figure 2.1: Example 3D reconstruction from Polycam app [44] created using RGB-D scanner on apple device, highlighting artifacts in the model.

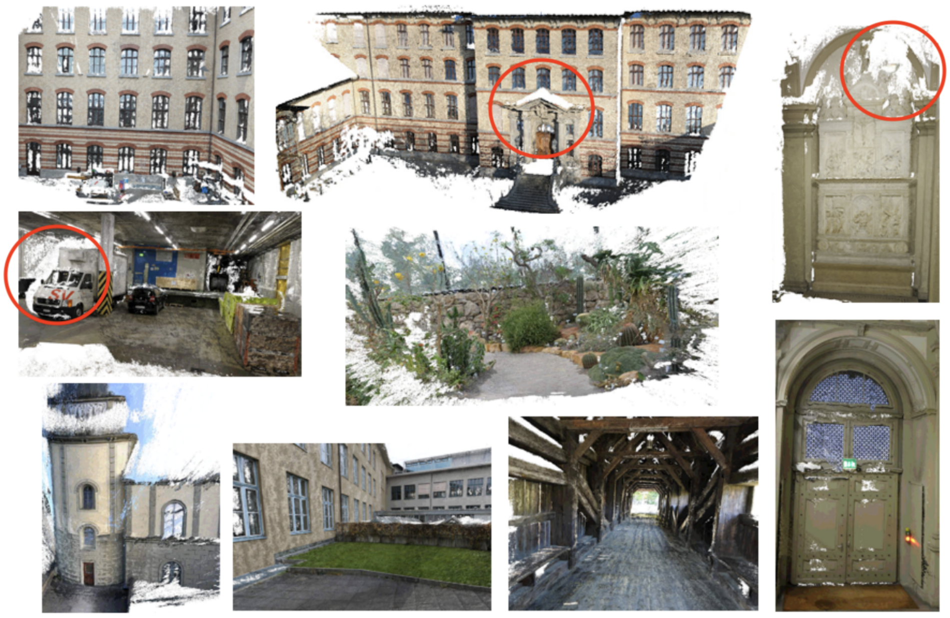


Figure 2.2: Example 3D reconstructions using PatchMatchNet [60] (MVS) on ETH3D benchmark [49], highlighting artifacts in the model.

2.2 Objectives

The primary objective of this research is to enhance the quality of depth maps obtained during the scanning process. The objective was formed because the EU H2020 ARtwin project running in Czech Institute of Cybernetics and informatics (CIIRC) required precise scanning using low cost devices like iPad PRO. While current standard 3D reconstruction pipelines yield satisfactory results, there remains room for improvement, particularly in the finer details and error minimization. Our focus has been on identifying and rectifying these areas to elevate the overall quality of the generated 3D models.

To achieve this, we investigated methods for optimizing scanned depth data through the application of neural networks. This approach leverages the capabilities of machine learning to substantially improve both the quality and consistency of depth maps, representing a significant advancement in the fields of spatial analysis and computer vision. We propose a method that would use two approaches that each showed promising results either in outdoor (MVS) or indoor (LiDAR) environments and combine them together via machine learning.

Despite utilizing modern technology known for its relatively high precision, inaccuracies in the scans persist, leading to a lack of reliable ground truth (GT) data. The only way to obtain GT data would be using expensive scanner devices, that were unfortunately not available to us. This presents a major challenge, as enhancing the data without a clear GT benchmark complicates the process. Neural networks typically require well-labeled reference data to train effectively. Two pivotal questions we address in this study is how to generate such reference data when the precision of our scanned inputs remains suboptimal, and how to avoid using GT data and use the imprecise data for our advantage.

2.3 Limitations

Despite our objectives to advance affordable 3D scanning technology, the project acknowledges several inherent limitations. A significant constraint is the computational demand of our methods, which predominantly rely on GPU processing and are not feasible with CPU-only setups. Additionally, our experimental dataset, which simulates a construction site environment, may not be representative of larger or outdoor scenes, potentially limiting the generalizability of our findings. Nextly, the precision of the scanning devices used, including the iPad Pro 2023, is not flawless, and some measurement errors are still observable. Lastly, the biggest limitation of our work is the lack of ground truth data, because our scanned data are imprecise, and require fine tuning. These limitations define the boundaries of our project and frame the context in which our findings should be interpreted.

2.4 Contributions

This project extends beyond the basic goal of refining dense point clouds. Utilizing the iPad Pro, a device prevalent in the consumer market, this study investigates how mainstream RGB-D scanners can be enhanced for routine use, potentially democratizing advanced 3D reconstruction technologies.

Despite the limited scope of our data set, we observed enhanced consistency in the reconstructed models, even with inputs that were novel to the neural network. This suggests robustness in our approach, indicative of its applicability to a broader range of real-world scenarios.

Additionally, this thesis identifies several options for further refinement of our techniques. These suggestions not only underscore the potential for incremental advances within our framework but also highlight opportunities for future investigations and applications into 3D reconstruction methodologies.

Chapter 3

Related Work

In the rapidly evolving field of computer vision and 3D reconstruction , the development and refinement of data capture and processing technologies have been essential in pushing the boundaries of what is achievable. Among these advancements, Structure from Motion (SfM), Multi-View Stereo (MVS), RGB-D scanning technologies, and depth map fusion are some of the main concepts we will discuss in this section. This chapter provides an overview of the main approaches in these areas, outlining their foundational principles and their contributions to both academic research and practical applications.

The following concepts that are discussed, are described in depth in Hartley's Multiple view geometry in computer vision book [20] or in [61, 12, 63].

3.1 Structure from Motion (SfM)

Structure from motion (SfM) is a method employed to generate 3D models from 2D RGB images [20]. This process is divided into several key phases: extracting features, matching these features between images, mapping into 3D and optimizing the resulting matches through bundle adjustment. These steps form a process that results in a sparse point cloud (providing us with intrinsics and extrinsics parameters for each camera). The subsequent section will delve into each of these stages in more depth, outlining their contributions to the construction of the sparse model. For our implementation of SfM, we utilized COLMAP [47, 48], a software that facilitates each of these operations, to assemble our sparse 3D model. For MVS implementation we used PatchmatchNet [60].

3.1.1 Feature Extraction

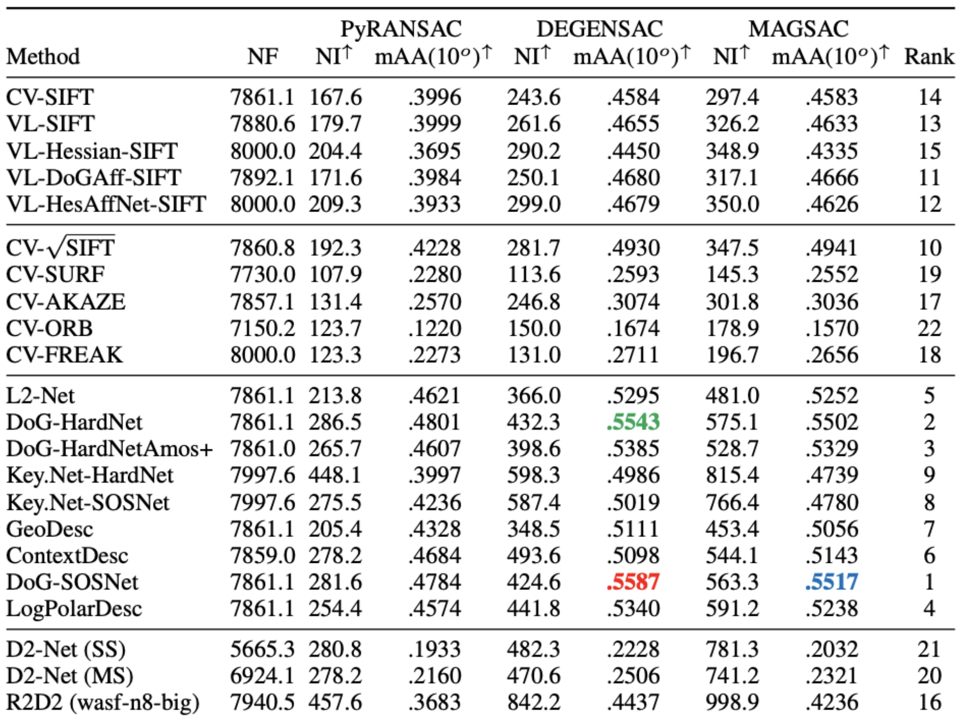
Feature extraction in the context of Structure from Motion (SfM) is an initial step where distinctive patches within images are identified and characterized. This process involves describing these patches by the coordinates of their centers and encoding their unique information into distinctive feature vectors. The objective here is to distill the images into a manageable set of features, effectively reducing data redundancy and enhancing computational efficiency. These features, or feature vectors, are crucial as they represent specific image patches and are designed or trained to be invariant to scale changes, rotations, and translations. This invariance ensures that the features remain recognizable across different images despite variations in size, orientation, or position.

A cornerstone in the field of feature extraction is the SIFT (Scale-Invariant Feature Transform) algorithm [26], which is valued for its robustness in detecting and describing image features. SIFT's ability to maintain consistency in feature identification across diverse conditions makes it a perfect tool to meet the requirements of SfM, where images captured from various angles and distances are synthesized into a cohesive 3D structure.

The paper "Image Matching across Wide Baselines: From Paper to Practice" published by Dmitro Mishkin [21], contains extensive comparison of feature detectors and matches.

The features can be categorized into three families. Firstly, the full "classical" pipelines (handcrafted methods) [27, 3, 6, 46, 2, 1, 24, 58, 7, 32, 30, 5, 34], secondly the descriptors learned on DoG keypoints [51, 33, 29, 52, 28, 15], and lastly we also have pipelines that are learned end to end [13, 38, 14, 45].

The table below (figure 3.1) shows the number of feature detected (NF), number of inliers produced by RANSAC [16] (NI) and mean average accuracy (mAA) of each of the approaches. The three highlighted numbers show the three best detector strategies.



		PyRANSAC		DEGENSAC		MAGSAC		
Method	NF	NI$^{\uparrow}$	mAA(10^{o})$^{\uparrow}$	NI$^{\uparrow}$	mAA(10^{o})$^{\uparrow}$	NI$^{\uparrow}$	mAA(10^{o})$^{\uparrow}$	Rank
CV-SIFT	7861.1	167.6	.3996	243.6	.4584	297.4	.4583	14
VL-SIFT	7880.6	179.7	.3999	261.6	.4655	326.2	.4633	13
VL-Hessian-SIFT	8000.0	204.4	.3695	290.2	.4450	348.9	.4335	15
VL-DoGAff-SIFT	7892.1	171.6	.3984	250.1	.4680	317.1	.4666	11
VL-HesAffNet-SIFT	8000.0	209.3	.3933	299.0	.4679	350.0	.4626	12
CV-$\sqrt{\text{SIFT}}$	7860.8	192.3	.4228	281.7	.4930	347.5	.4941	10
CV-SURF	7730.0	107.9	.2280	113.6	.2593	145.3	.2552	19
CV-AKAZE	7857.1	131.4	.2570	246.8	.3074	301.8	.3036	17
CV-ORB	7150.2	123.7	.1220	150.0	.1674	178.9	.1570	22
CV-FREAK	8000.0	123.3	.2273	131.0	.2711	196.7	.2656	18
L2-Net	7861.1	213.8	.4621	366.0	.5295	481.0	.5252	5
DoG-HardNet	7861.1	286.5	.4801	432.3	**.5543**	575.1	.5502	2
DoG-HardNetAmos+	7861.0	265.7	.4607	398.6	.5385	528.7	.5329	3
Key.Net-HardNet	7997.6	448.1	.3997	598.3	.4986	815.4	.4739	9
Key.Net-SOSNet	7997.6	275.5	.4236	587.4	.5019	766.4	.4780	8
GeoDesc	7861.1	205.4	.4328	348.5	.5111	453.4	.5056	7
ContextDesc	7859.0	278.2	.4684	493.6	.5098	544.1	.5143	6
DoG-SOSNet	7861.1	281.6	.4784	424.6	**.5587**	563.3	**.5517**	1
LogPolarDesc	7861.1	254.4	.4574	441.8	.5340	591.2	.5238	4
D2-Net (SS)	5665.3	280.8	.1933	482.3	.2228	781.3	.2032	21
D2-Net (MS)	6924.1	278.2	.2160	470.6	.2506	741.2	.2321	20
R2D2 (wasf-n8-big)	7940.5	457.6	.3683	842.2	.4437	998.9	.4236	16

Figure 3.1: Comparison of stereo results of different detector strategies with 8k features [21].

In the presented stereo task, deep descriptors extracted from DoG keypoints achieved top performance in terms of mean Average Accuracy (mAA). The handcrafted descriptor SIFT (specifically RootSIFT) also remained competitive, ranking 10 on the stereo task and within 13.1% of the top-performing method. However, other classical feature descriptors did not fare as well, with inconsistent performance between validation and test sets. R2D2, an end-to-end method, produced a much larger number of "inliers" (both correct and incorrect) than most other methods, suggesting that its incompatibility with the ratio test could pose challenges when combined with sample-based robust estimators.

The following visualization (figure 3.2) shows the reconstructed point cloud using COLMAP while utilizing three different local features: SIFT, SuperPoint and R2D2. The reconstructions using SIFT and R2D2 are dense, though they differ in some aspects. While the reconstruction with SuperPoint is fairly less dense, given its ability to extract a limited number of features effectively and its poses seem less accurate.

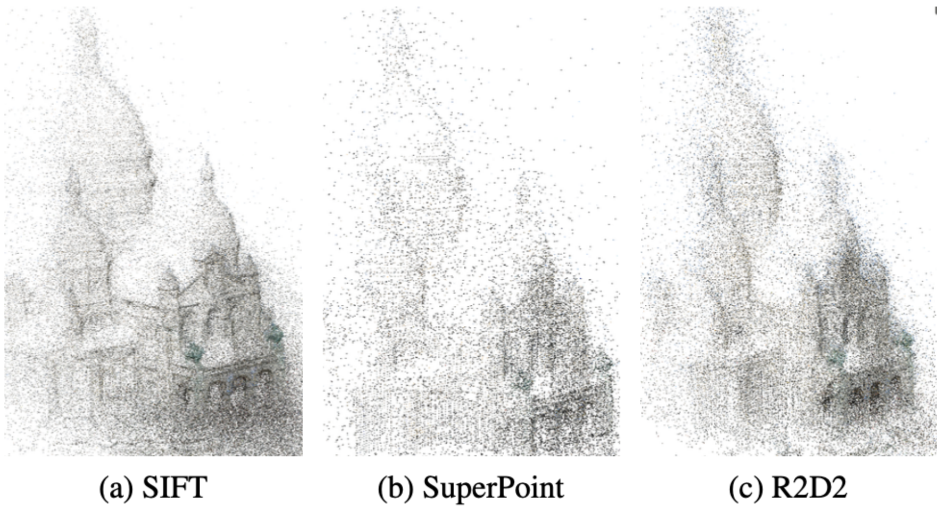


Figure 3.2: Results of COLMAP with different local features on scene "Sacre Coeur" from [21].

3.1.2 Feature Matching

Feature matching represents a critical phase in the Structure from Motion (SfM) workflow, serving as the bridge between isolated features extracted from individual images and the cohesive 3D reconstruction they will form. This stage begins with the identification of tentative matches among features across image pairs. This is accomplished by comparing feature vectors and selecting pairs that exhibit the smallest distance between them. This distance is usually described by cosine similarity, thereby establishing a preliminary set of correspondences among the distinct patches detected in different images.

Once tentative matches are established, their validity is assessed through the verification of geometric constraints derived from epipolar geometry. This step ensures that only those matches that satisfy the expected constraints within a threshold—given the camera's movement and orientation between shots—are accepted as true correspondences. These true correspondences are referred to as inliers. It serves to filter out inconsistent matches that, although close in feature space, do not adhere to the physical transformation between multiple viewpoints.

Some recent feature matching pipelines leverage pre-detected features using detectors such as [13, 65, 56]. Examples of such pipelines include LightGlue [25] and SGMNet [64]. There are also pipelines that operate without traditional detectors, like LoFTR [50] and ASpanFormer [10]. The following figure 3.3, taken from Xu's paper [62], provides a comparison of these various feature matching pipelines.

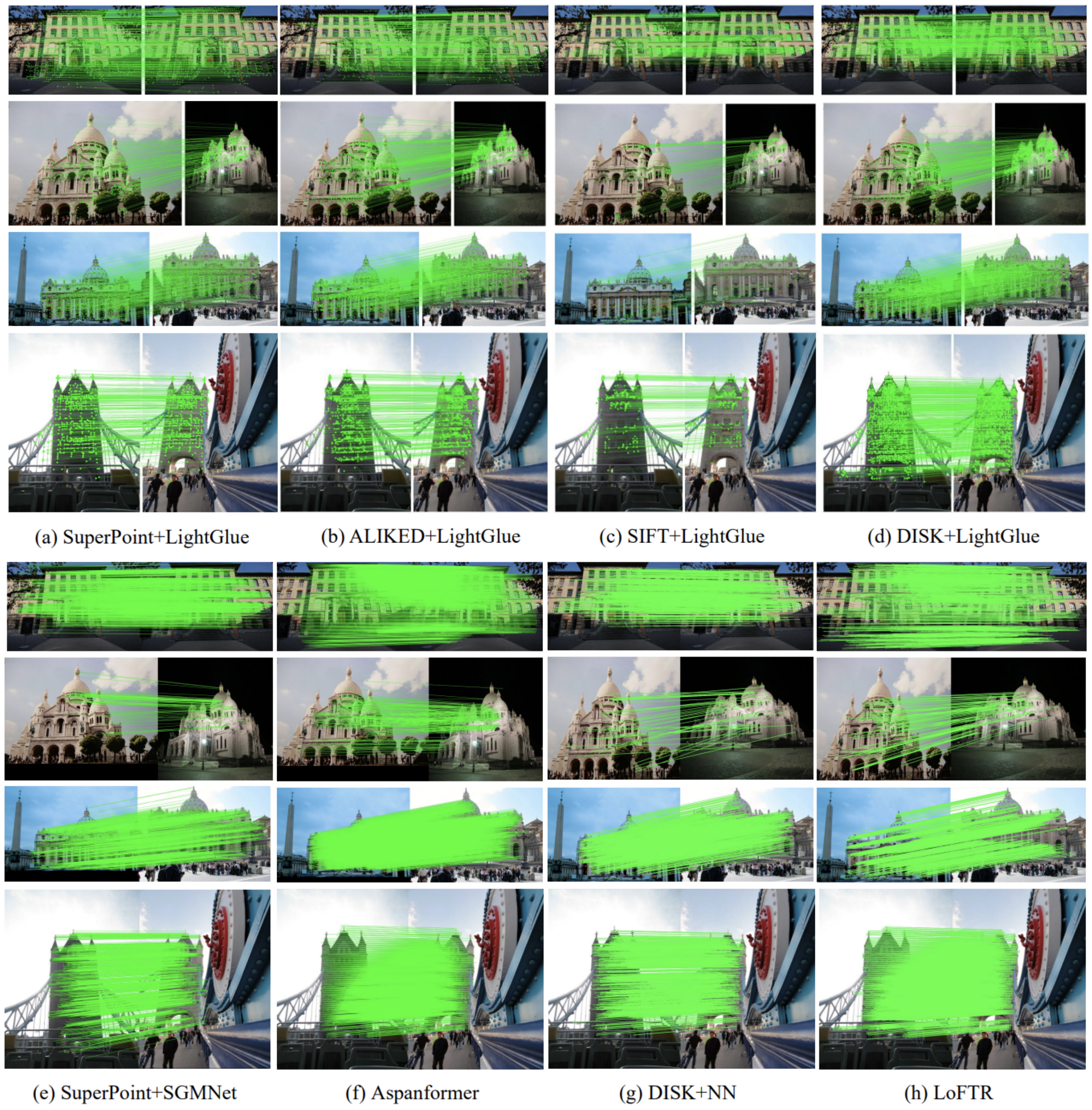


Figure 3.3: Matching results for outdoor images using different matching techniques [62].

The primary parameters of each detector are robustness and precision. Handcrafted methods such as SIFT generally yield fewer keypoints but with higher precision. On the other hand, trained detectors typically generate a larger number of features that are more robust, although their precision is not as high. For a more detailed analysis, refer to the bachelor's thesis by Ondřej Kafka [22].

3.1.3 Mapping

Mapping in the Structure from Motion (SfM) process is the next step that transforms the found correspondences of 2D image feature points into a 3D world coordinate system. The procedure begins with the selection of an initial pair of cameras, which serve as the reference for the mapping process. Usually, the coordinate system of the first camera defines the origin and orientation of the world coordinate system and the baseline length of the initial pair is scaled to one, i.e., it defines the scale of the world coordinates.

Following this, a new image is registered to the reconstruction based on found correspondences between the new image and the cameras that are already registered. The 2D-2D correspondence between already registered (reference) cameras and the new camera is essential for establishing the connections between 2D image feature points and their corresponding points in the 3D space. With these correspondences in hand, the absolute pose of the newly registered camera can be accurately computed, anchoring it within the emerging 3D reconstruction.

This mapping cycle of finding 2D to 3D correspondences to calculate the absolute pose is repeated for each new camera added. It is also necessary to triangulate the remaining 2D to 2D correspondences in order to obtain 3D points. Through this iterative process, the SfM method registers 2D images to construct accurate 3D reconstruction of the scene.

These stitching processes are usually made iteratively using sampling schemes like RANSAC [16] relying on solutions for pose solving such as the 5- [37], 7- [18] or 8-point algorithm [19]. Optimization can be achieved using likelihood instead of reprojection for example using MLESAC. MLESAC adopts the same sampling strategy as RANSAC to generate putative solutions but chooses the solution to maximize the likelihood rather than just the number of inliers [53].

Incorporating Simultaneous Localization and Mapping (SLAM) into Structure from Motion (SfM) [36, 9] provides several valuable benefits. The real-time processing and incremental mapping capabilities are vital for applications that require immediate spatial awareness. SLAM's loop closure detection helps minimize drift by recognizing previously visited locations, leading to more accurate maps. Plus, the map-based optimization simultaneously refines the 3D map and trajectory for improved localization, ensuring robust performance even in changing environments. This makes SLAM essential in enhancing the accuracy and reliability of SfM.

3.1.4 Bundle Adjustment

The last step of SfM is the bundle adjustment [54]. Bundle adjustment is a optimization method used to refine the 3D reconstruction created in the mapping step of the SfM. The mapping algorithm provides initial estimates of camera positions and the 3D points corresponding to matched features within the scene. However, these initial estimates may contain inaccuracies due to various sources of error, such as imprecise feature matching or outliers.

Bundle adjustment adjusts these initial estimates to minimize the overall re-projection error, which is the discrepancy between the observed feature positions in the images and the projections of 3D points using estimated camera poses and intrinsic parameters. This optimization is conducted by simultaneously refining the camera parameters and the 3D point coordinates.

To achieve this refinement, bundle adjustment employs algorithms such as Levenberg-Marquardt [35], which iteratively improve the accuracy of the camera and 3D points.

Through bundle adjustment, the resulting output includes the optimized positions of cameras and the refined 3D coordinates of features, leading to a more accurate and coherent sparse model.

3.2 Multi View Stereo (MVS)

Following the generation of the sparse model, the subsequent phase in 3D reconstruction is the Multi-View Stereo (MVS) [48]. While Structure from

Motion (SfM) produces the initial sparse model along with the camera orientations, MVS further refines the 3D representation by estimating additional 3D points that were not identified during the initial feature extraction process in SfM.

3.2.1 Depth Map Creation

The creation of depth maps begins with a principle similar to the feature extraction stage in Structure from Motion (SfM), aiming to identify corresponding pixels in images. Modern approaches usually employ patch matching techniques [4]. These methods involve comparing small sections across the complete images, assessing their similarity based on photometric consistency to ensure uniform appearance from different viewpoints.

Followingly images are rectified to align corresponding points along the same image rows or columns, simplifying the subsequent computation of disparities. Next is the calculation of disparity, which involves determining the difference in the horizontal positions of matching points between stereo images. This disparity is inversely proportional to the scene depth at each point, allowing closer objects to exhibit greater disparity. These disparities are then used to construct depth maps, where each disparity value is converted into a depth measurement based on the stereo setup's geometry, specifically the cameras' baseline and focal lengths.

Subsequently, these 3D points generated from disparity maps undergo a consistency check from various viewpoints, confirming the accuracy of the depth estimates across all views. The final depth estimates are then interpolated and smoothed in each view to eliminate holes and achieve a realistic appearance.

3.2.2 Mesh Construction

The subsequent standard step in 3D reconstruction can be transitioning from the dense point cloud, obtained in the preceding phase, to constructing a mesh. This dense point cloud is generated by projecting the estimated depth values into 3D space from each viewpoint using algorithms like [42], [43], [59] or [31]. To create the mesh from these 3D points, we employ algorithms such as the Poisson Surface Reconstruction [23] or Delaunay Triangulation [8].

To enhance the visual fidelity of the model, texturization can be applied. This process involves mapping the RGB data from the original images onto the mesh to create textures that accurately mirror the scene's appearance, utilizing the camera parameters for precise alignment.

The mesh creation is a standard step during 3D reconstruction, however, for our project, we focused solely on analyzing the depth maps produced by MVS, comparing these with depth maps acquired from iOS LiDAR. As mentioned earlier, the execution of the SfM algorithm was carried out using COLMAP while MVS was carried out using PatchmatchNet [60].

3.3 RGB-D Scanning

The data acquisition phase of our project was conducted exclusively with an iPad Pro 2023, chosen for its integrated LiDAR sensor and the customer choice, i.e., in EU H2020 ARTwin project, the device was specified by companies managing the construction progress. However, it's worth noting that any Apple device equipped with a LiDAR sensor could be employed for similar data collection purposes, as they are capable of capturing depth information. The comprehensive procedure for scanning and data collection is detailed further in 4.2 and 4.1.

By harnessing depth data from both the iPad Pro's LiDAR sensor and the Multi-View Stereo (MVS) approach, we succeeded in gathering a diverse set of depth measurements. This diversity in data sources allowed us to achieve a greater variance in our dataset, providing a broader scope for data analysis due to the increased variability of the information collected, while allowing us to identify the fact that some parts of the 3D representation are more accurate and reliable from LiDAR and some from MVS.

Chapter 4

Methodology/Prerequisites

This chapter details the methodology used to capture, analyze, and align RGB-D data to assess defects of our scanned environment. It outlines the data collection process, the alignment of different coordinate systems, and derives the reprojection formula. The chapter also discusses the evaluation metrics used to identify inconsistencies and gauge the quality of 3D point clouds.

4.1 Recording Application

This chapter outlines the application we utilized for data collection, pivotal to this thesis. The application captures RGB-D images, yielding color and depth data for each pixel. It also integrates GPS and gyroscope technologies to track the camera's location and orientation. We selected a construction site for data collection because the app is specifically tailored for identifying defects in such environments.

4.1.1 Data Collection Setup

Our dataset scanned by our partner Artefacto during the EU H2020 ARtwin project comprises a total of 193 RGB-D images. The dataset was scanned at an Eiffage construction site. Each RGB image has a corresponding depth image, captured from the identical camera position, with each image boasting a resolution of 1920 x 1440 pixels.

However, it is important to note that the default scanning resolution of LiDAR is not 1920x1440 pixels. LiDAR scans are conducted at a much lower resolution and then upsampled. This upsampling process can introduce some errors to the accuracy of the depth map.

4.1.2 Definition of Coordinate Systems

The sparse model constructed using COLMAP lacks a metric scale; hence, all distances within the sparse model are geometrically consistent but do have a different scale. The aim was to employ the camera positions derived from the app, which provides correct scale but is loaded with drift of the camera poses, and synchronize them with the refined camera models from COLMAP. This alignment would ensure that the refined camera centers from COLMAP correspond to real-world metric system.

4.1.3 Alignment of Coordinate Systems

Upon acquiring the 3D positions of the camera centers, we applied the Procrustes function in Matlab, which facilitates the alignment of point sets. The Procrustes function calculates the Procrustes transformation, an optimal shape-preserving Euclidean transformation (including rotation, reflection, scaling, and translation) between two datasets, $\mathbf{X}$ and $\mathbf{Y}$. This transformation is designed to minimize the sum of squared differences between camera centers from iPad tracking and the COLMAP reconstruction.

To align the coordinate systems between COLMAP and the iPad, we accounted for the fact that the iPad uses a left-handed coordinate system, while COLMAP employs a right-handed system. Additionally, in the iPad's coordinate system, the z-axis is inverted compared to COLMAP. To reconcile these differences, we flipped the direction of the x-axis in COLMAP's coordinate system to match the one of iPad. This adjustment was also applied to the updated projection matrix to ensure alignment between the two coordinate system.

Despite the alignment of coordinate systems and subsequent refinement of camera positions, when reprojecting points from depth maps into 3D space using references 4.12 and 4.13, the point clouds exhibited an error margin of approximately ±2cm. As an example let's look at two 3D reconstructions of manipulators reconstructed using the same pipeline as explained prior (figures 4.1 and 4.2), showing imprecisions of the point clouds.

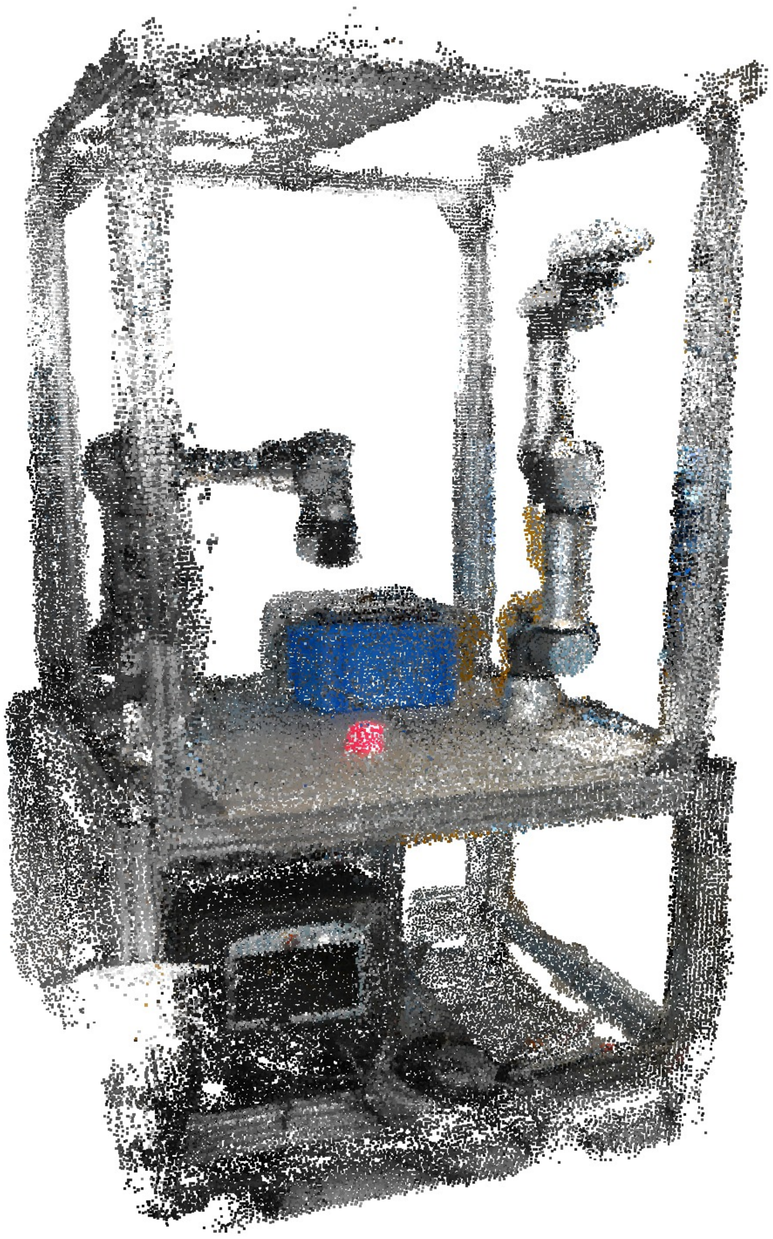


Figure 4.1: Example of error on the refined pointcloud 1.

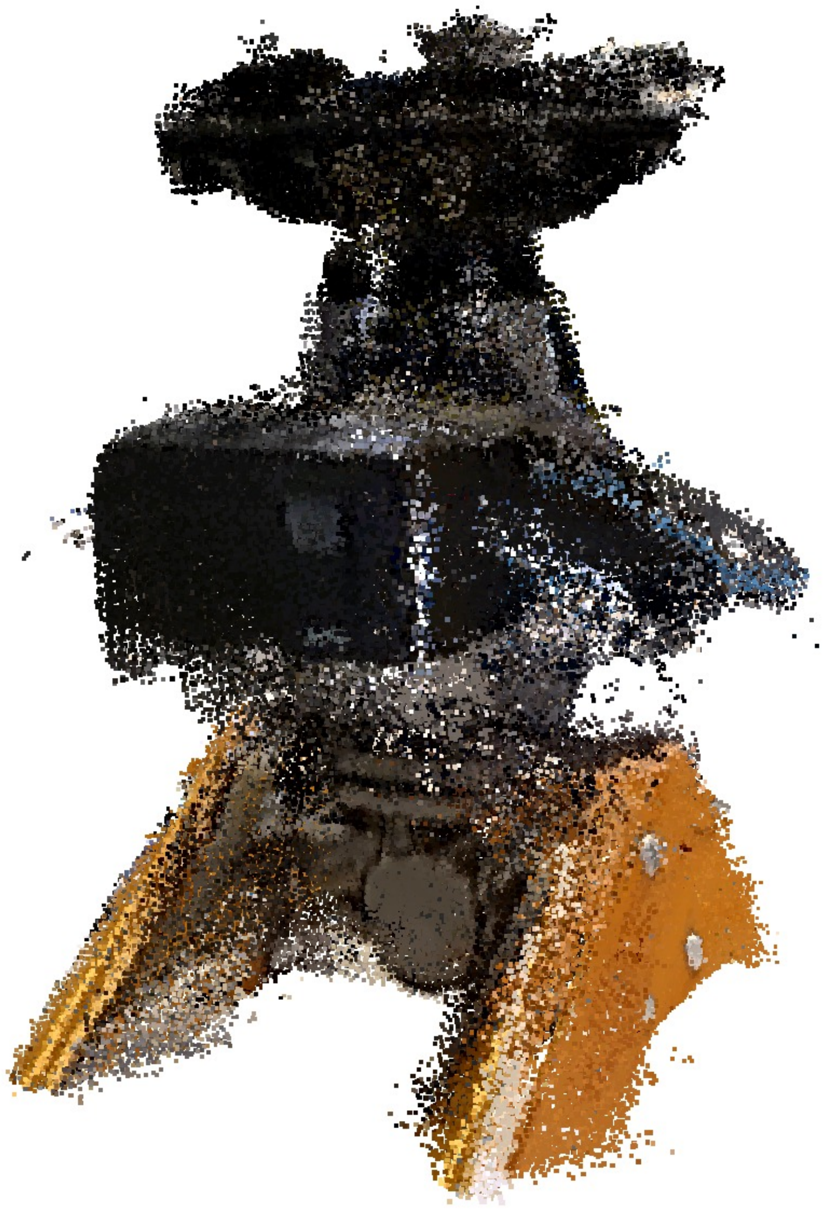


Figure 4.2: Example of error on the refined pointcloud 2.

4.2 Real-world Indoor/Outdoor Scanning

The choice of scanning environment is important in evaluating the performance and applicability of different depth sensing technologies such as LiDAR and Multi-View Stereo (MVS). Our project focuses on a construction site, an environment that uniquely combines aspects of both indoor and outdoor settings. This mixed setting presents unique challenges and opportunities for assessing and comparing these technologies.

4.2.1 Environment Selection

The construction site was selected as the scanning environment because it inherently contains both indoor and outdoor elements. This choice was strategic, aimed at evaluating the technologies under varying conditions.

Outdoor environments often impair the effectiveness of LiDAR technology. The primary issue with outdoor LiDAR scanning arises from the interference of natural sunlight with the LiDAR sensors. Sunlight can overpower the laser beams emitted by LiDAR, making the beams nearly invisible and significantly reducing the precision of the depth measurements. This phenomenon is particularly problematic in bright conditions [41].

In contrast, Multi-View Stereo (MVS) does not rely on light emission from the device but rather on capturing images from multiple viewpoints to reconstruct a 3D model. MVS can be more adaptable to different lighting conditions, making it potentially more effective in outdoor environments where LiDAR struggles. On the other hand, MVS does struggle with texture-less areas, because of the absence of feature points, also leading to imprecisions in the 3D reconstruction.

4.3 Geometry of Reprojections

This chapter introduces and derives the reprojection formula extensively used throughout this thesis. This formula translates image values into a reference frame of another camera, providing a mathematical foundation for evaluating the 3D reconstruction.

4.3.1 Used Matrices and Vectors

In this section, we describe the calibration and projection matrices essential for our reprojection process. The calibration matrix **K** adjusts the 3D coordinates according to the camera's internal characteristics and they are mapped onto the 2D image plane. This includes scaling by the focal length, accounting for the optical center, and correcting for skew in the image sensor. A visualization and in depth explanation of these transformations can be found in the book Elements of Geometry for Computer Vision and Computer Graphics [40] written by Tomas Pajdla, more specifically on page 45, figure 7.2. b). For simplicity, our model assumes zero radial distortion. Radial distortion typically affects real-world lenses, causing straight lines to appear curved. By neglecting this, we simplify the mathematical model and focus on transformations, which are computationally less intensive and more straightforward to manage.

The calibration matrix, often denoted as **K**, encapsulates the intrinsic parameters of the camera. These parameters include the focal lengths along the x and y axes (f_x and f_y) and the optical center coordinates (c_x and c_y) [40]. The matrix is defined as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

This matrix simplifies our approach by assuming the camera exhibits no skew between the axes.

The projection matrix **P** is a 3x4 matrix that combines the camera's intrinsic parameters with its extrinsic parameters (rotation **R** and translation **t**), representing the camera's position and orientation in the world:

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{4.2}$$

In our derivation of reprojection formula, we will define matrix **E** which is essentially the Euclidean transformation from world coordinates into the camera coordinates. It transforms 3D points in their homogeneous coordinates. **E** therefore has form:

$$\mathbf{E} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^\top & 1 \end{pmatrix} \tag{4.3}$$

4.3.2 Reprojection Formula

One of the main equations employed in the dataset creation phase was the reprojection formula, enabling the transfer of pixel values from a reference image to a source image through spatial transformation. This derivation outlines the underlying mathematical procedure.

We start with the coordinates of a pixel in the reference camera, denoted as $\mathbf{x} \in \mathcal{R}^2$. The initial step involves transforming these 2D coordinates into the camera coordinate system via:

$$\underline{\mathbf{v}} = \mathbf{K}^{-1}\underline{\mathbf{x}} \tag{4.4}$$

Note that both the vectors $\underline{\mathbf{x}}$ and $\underline{\mathbf{v}}$ are in homogeneous coordinates.

To convert from homogeneous coordinates of $\underline{\mathbf{V}}$ and $\underline{\mathbf{v}}$ to non-homogeneous coordinates $\mathbf{V}$ and $\mathbf{v}$ we use:

$$\mathbf{V} = \left(\frac{V_x}{V_w}, \frac{V_y}{V_w}, \frac{V_z}{V_w}\right)^T \quad \text{where } \underline{\mathbf{V}} = \begin{bmatrix} V_x \\ V_y \\ V_z \\ V_w \end{bmatrix} \tag{4.5}$$

$$\mathbf{v} = \left(\frac{v_x}{v_w}, \frac{v_y}{v_w}\right)^T \quad \text{where } \underline{\mathbf{v}} = \begin{bmatrix} v_x \\ v_y \\ v_w \end{bmatrix} \tag{4.6}$$

Subsequently, to project the point from 2D into 3D in the camera coordinate system, we scale this vector by the depth value d of the pixel $\mathbf{x}$:

$$\mathbf{v}' = \mathbf{v}d \tag{4.7}$$

Transitioning the point from the camera coordinate system to the world coordinate system requires multiplication by the inverse of the extrinsic matrix $\mathbf{E}_{\text{ref}}^{-1}$ defined in 4.3:

$$\underline{\mathbf{X}} = \mathbf{E}_{\text{ref}}^{-1}(\underline{\mathbf{v}'}) \tag{4.8}$$

Having derived the world coordinates $\mathbf{X}$, the reprojection to corresponding pixel coordinates $\mathbf{y}$ in the source image reverses this transformation process, involving $\mathbf{E}_{\text{src}}$ and the calibration matrix $\mathbf{K}$:

$$\underline{\mathbf{V}} = \mathbf{E}_{\text{src}}\mathbf{E}_{\text{ref}}^{-1}(\underline{\mathbf{v}'}) \tag{4.9}$$

$$\underline{\mathbf{y}} = \mathbf{K}\mathbf{V} \tag{4.10}$$

The coordinates of $\mathbf{y}$ indicate the position within the source image that geometrically corresponds to the value in the reference image. This allows for the alignment of RGB-D values between images:

$$d = \mathbf{D}_{\text{ref}}(\mathbf{x}), \qquad d' = \mathbf{D}_{\text{src}}(\mathbf{y}) \tag{4.11}$$

where $\mathbf{D}_{\text{ref}}(\mathbf{x})$ is the depth value at position $\mathbf{x}$ of the depth map of the reference image and $\mathbf{D}_{\text{src}}(\mathbf{y})$ is the depth value at position $\mathbf{y}$ of the depth map of the source image.

To find the corresponding 3D point representations from both images, the following formulations are employed:

$$\underline{\mathbf{X}} = \mathbf{E}_{\text{ref}}^{-1}\left(\mathbf{K}^{-1}\underline{\mathbf{x}}d\right) \tag{4.12}$$

$$\underline{\mathbf{X}'} = \mathbf{E}_{\text{src}}^{-1}\left(\mathbf{K}^{-1}\underline{\mathbf{y}}d'\right) \tag{4.13}$$

In an ideal scenario $\mathbf{X}$ and $\mathbf{X}'$ should be the same up to the inconsistency introduces by the parameters $\mathbf{E}$, $\mathbf{K}$, d, $\mathbf{x}$, and $\mathbf{y}$. Therefore, we need to simplify the mathematical problem by assuming that some values are correctly determined. Otherwise, we have under constrained problem and the variables can acquire any values. In the following text, we assume that $\mathbf{E}$ and $\mathbf{K}$ are fixed, i.e., not loaded with an error.

4.4 Identification and Visualization of Inconsistencies

The next step of the thesis was to evaluate the inconsistencies of our model. We consider fixed camera intrinsic and extrinsic parameters that were generated by COLMAP SfM, after their alignment to iPad coordinate system to fix the scale.

To further evaluate the point cloud quality, we used various metrics that offer insights into the accuracy and integrity of the 3D reconstructions, as we assume the lack of ground truth (GT) data, caused by the inaccuracies of the scans (more details in section 4.4.1). Therefore the metrics used evaluate the model without GT and they included reprojection error, which measures the discrepancy between the observed image points and the corresponding projections of the reconstructed 3D points; photometric consistency, assessing the similarity in appearance of a point across multiple views; and a direct comparison with depths obtained from LiDAR scans. Additionally, the completeness of the point cloud was scrutinized to evaluate the coverage and density of the reconstructed model.

Upon analysis, it became evident that the most challenging areas in the reconstruction process were those subjected to occlusion. Occlusions, caused by physical barriers such as walls, corners, and pillars, pose significant challenges for both depth estimation and point cloud reconstruction. These obstacles can obstruct the line of sight necessary for accurate depth measurement and feature matching, leading to gaps in the data or inaccuracies in the reconstructed model. Thats why we chose a specific image 4.3 for analysis.

Further investigation into these problematic areas revealed several patterns and insights. For instance, occlusions often resulted in sparse or incomplete sections within the point cloud (more in section 4.4.1), highlighting the need for advanced strategies to predict or infer the missing information based on the available data. Moreover, the analysis of photometric consistency (more in section 4.4.1) in occluded regions underscored the challenges in maintaining visual coherence across different viewpoints, especially in complex scenes with varying levels of light and shadow.

4.4.1 Analysis Tools and Techniques

In examining the inconsistencies, a specific image was selected to demonstrate the various metrics due to its depth complexity from the chosen viewpoint. This particular image 4.3 features a corridor, which is prone to occlusions when observed from slightly different angles. This characteristic made it an ideal candidate for illustrating the challenges associated with depth measurement and feature matching in complex scenes.

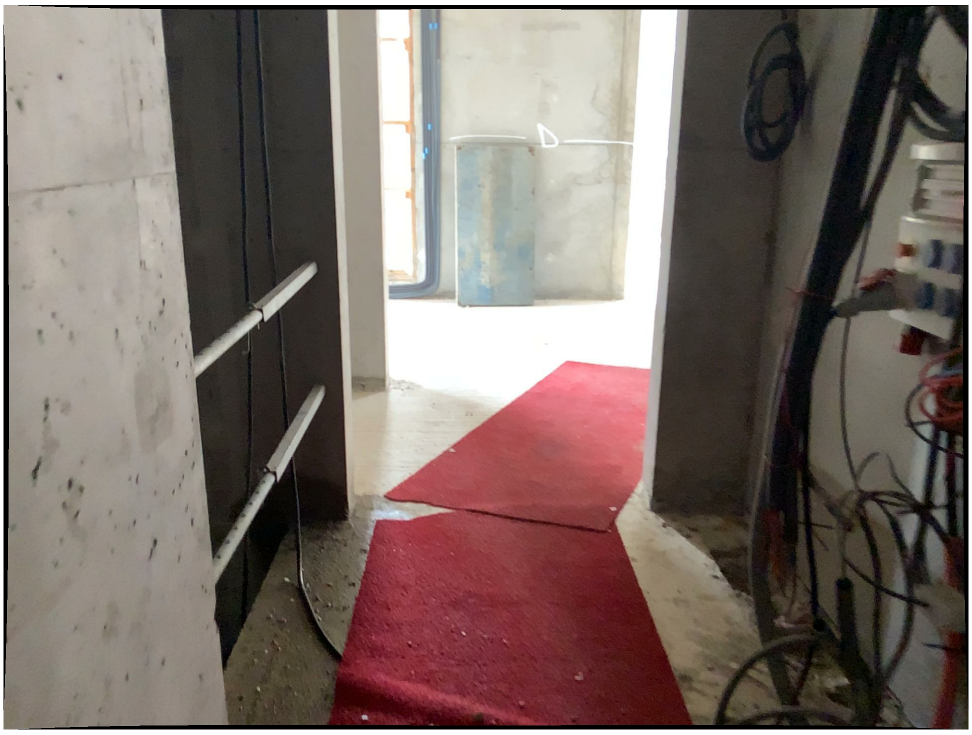


Figure 4.3: Undistorted RGB image chosen for depth consistency analysis because it is prone to occlusions.

Comparison with LiDAR

The initial assessment of the constructed point cloud and camera placements involved a direct comparison between the depth data derived from Multi-View Stereo (MVS) and that acquired via LiDAR technology.

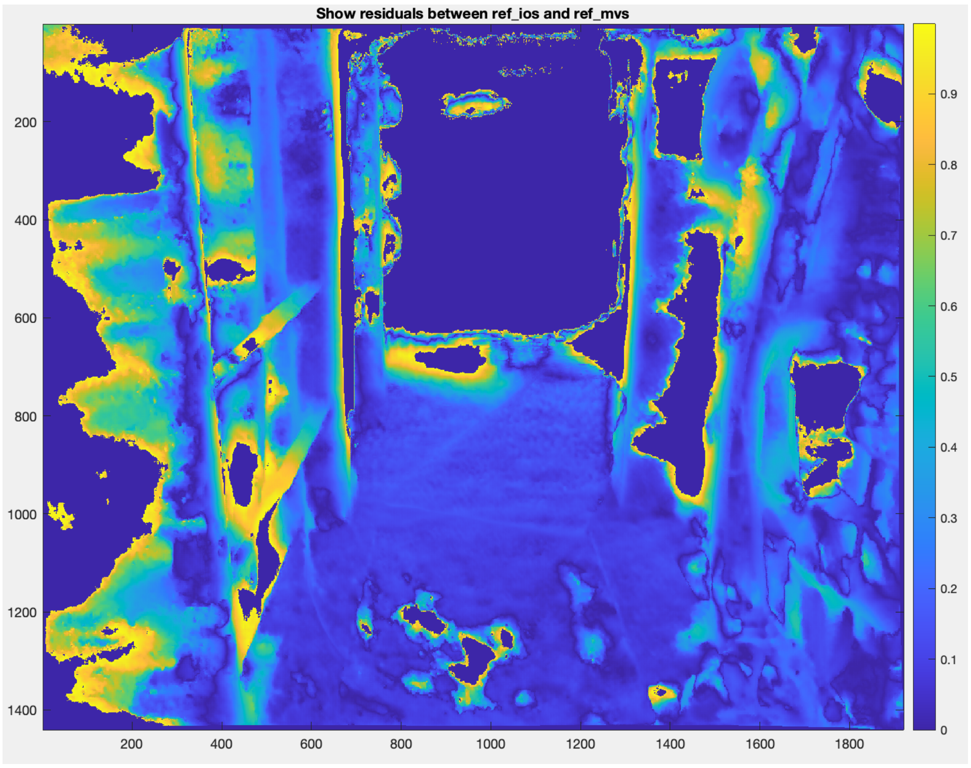


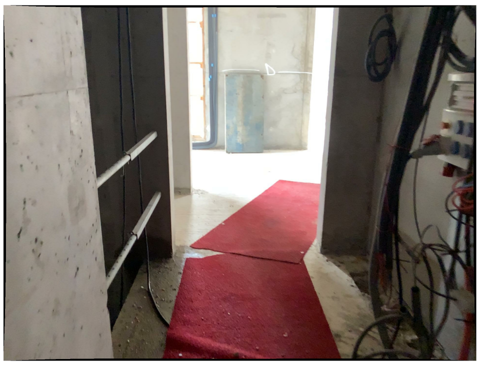
Figure 4.4: Absolute error between LiDAR and MVS depth in meters. Values of the absolute error higher than one meter were discarded for visualization purposes.

The graphical representation 4.4 clearly reveals discrepancies between the depth maps, with some of the errors reaching significant levels (up to approximately 3 meters). However, the overall Mean Absolute Error (MAE) stands at 0.633 meters, indicating the average deviation across the dataset. It is also necessary to explain, why we previously mentioned errors of approximately ±2cm in 3D points when we see an average error of 63cm. The reason lies in the filtering of inconsistencies and the preservation of areas with a high density of points, that is, filtering out points that were not observed in similar locations repeatedly. This filtering does not occur here.

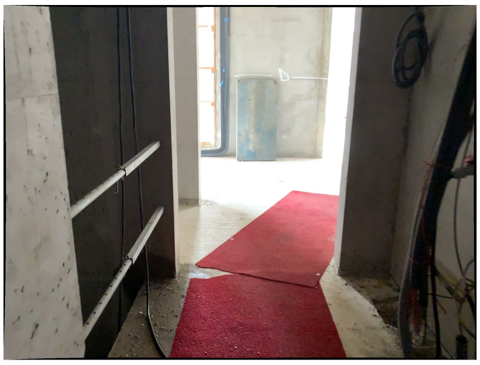
Reprojection Error

To assess the reprojection error, we focused on an image that was reprojected according to the formula referenced in 4.11. This particular image was chosen due to its proximity to the reference image. The figures below display the reprojection error comparing both the LiDAR-captured depth values 4.6a and those estimated by MVS 4.6b.

Reprojection error is important because it measures the difference between the expected projection of a 3D point onto a 2D image plane and its actual observed position. This error helps assess the accuracy of 3D models and camera calibrations. A smaller reprojection error means the model or system is accurately estimating the spatial relationships between the camera and the scene, while a larger error can indicate problems in feature matching or calibration.

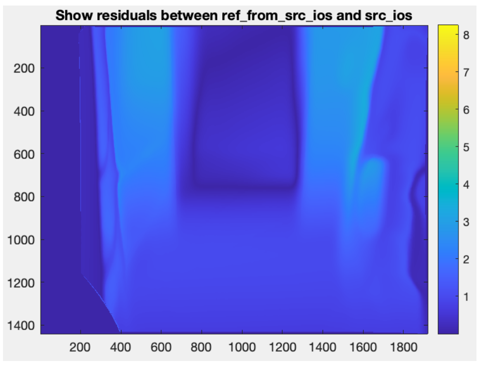


(a) : Reference image RGB

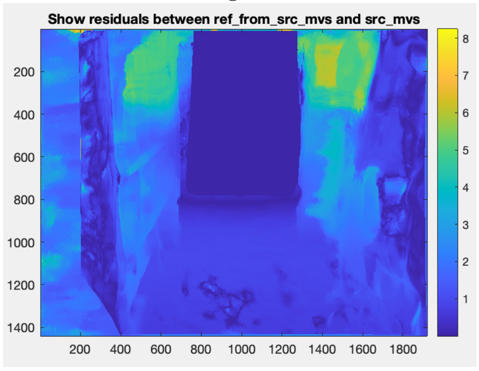


(b) : Source image RGB

Figure 4.5: Reference and source RGB images.



(a) : Reprojection error for LiDAR-captured values in meters.



(b) : Reprojection error for MVS-estimated values in meters.

Figure 4.6: Reprojection errors between LiDAR and MVS.

Analyzing the discrepancies from the LiDAR, it is observed that the error can escalate to as much as 3.5 meters, notably around the corridor area.

Conversely, the reprojection error for the MVS-derived depth values peaks at 6 meters, indicating significant inaccuracies in the estimated depth cloud. This aligns with the statement that MVS has problems accurately estimating the depth for texture-less areas.

Photometric Consistency

To analyse the photometric consistency of the reprojections, we measured Normalized Cross-Correlation (NCC) and Structural Similarity Index (SSIM) of the reprojections on the selected picture.

NCC is a statistical method for measuring the similarity of signals or images. NCC is robust to changes in amplitude between the two compared datasets, making it ideal for comparing images that may have been taken under different lighting conditions or that have differences in exposure. The formula for calculating NCC is:

$$NCC(A,B) = \frac{\sum_{x,y}[A(x,y)-\bar{A}][B(x,y)-\bar{B}]}{\sqrt{\sum_{x,y}[A(x,y)-\bar{A}]^2 \sum_{x,y}[B(x,y)-\bar{B}]^2}} \tag{4.14}$$

where A and B are the images compared, $A(x,y)$ and $B(x,y)$ are the intensity (depth) values in the images and $\bar{A}$ and $\bar{B}$ are the mean values of the images A and B.

SSIM is a method used to measure the similarity between two images. It is designed to provide a more comprehensive assessment of image quality by considering changes in structural information, luminance, and contrast, rather than focusing solely on pixel-by-pixel differences. The SSIM index is based on the perception that images are highly structured and that the human visual system is highly sensitive to structural variations. Here's the definition and formula for SSIM:

$$\mathrm{SSIM}(A,B) = \frac{(2\mu_A\mu_B + c_1)(2\sigma_{AB} + c_2)}{(\mu_A^2 + \mu_B^2 + c_1)(\sigma_A^2 + \sigma_B^2 + c_2)} \tag{4.15}$$

where A and B are the images being compared, μ_A and μ_B are the average pixel intesity (depth) values in images A and B, σ_A^2 and σ_B^2 are the variances of A and B, σ_{AB} is a covariance of A and B, and c_1 and c_2 are defined as:

$$c_1 = (k_1 L)^2 \tag{4.16}$$

$$c_2 = (k_2 L)^2 \tag{4.17}$$

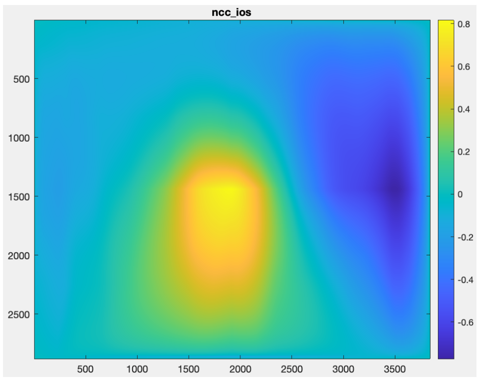
where L is the dynamic range of the pixel values and $k_1 = 0.01$ and $k_2 = 0.03$ by default.

The SSIM values are:

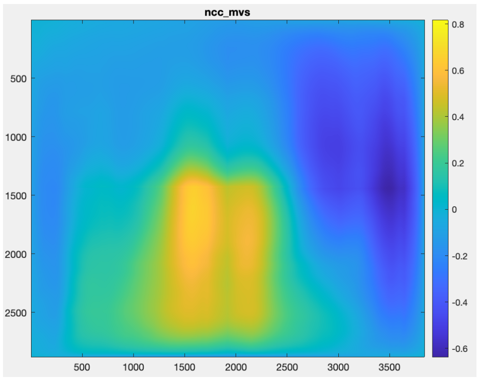
Method	Average SSIM ↑
LiDAR	0.7022
MVS	0.4105

Table 4.1: Comparison of Average SSIM values for LiDAR and MVS values.

The NCC values are:



(a) : NCC of reprojection of LiDAR depth values in meters.



(b) : NCC of reprojection of MVS depth values in meters.

Figure 4.7: NCC of reprojections from LiDAR and MVS.

The visualization for LiDAR depth values demonstrates a relatively higher NCC 4.7a, suggesting a stronger correlation and, hence, a more accurate depth estimation when compared to the original imaging data. In contrast, the MVS depth values exhibit lower NCC scores 4.7b, indicating reduced photometric consistency. This disparity suggests that the reprojected MVS depth values are less aligned with the original images, possibly due to inherent limitations in capturing fine details or dealing with occlusions in the MVS approach.

Chapter 5

Depth Map Consistency

This chapter focuses on improving the depth maps gathered during data collection. The main goal is to merge the iPad and MVS depth maps, improving the consistency across multiple views, ensuring they accurately match the scene's reprojections into different viewpoints. Achieving consistent depth maps is crucial for their reliable use in 3D modeling and other applications. We will explore two techniques to refine depth data, addressing common issues such as noise and inaccuracies, to produce depth maps that have smaller inconsistencies.

5.1 Pixel-Wise Approach

To enhance depth map consistency, we start with a pixel-by-pixel neural network approach. The network takes reprojected depth values as input and outputs the optimal mean depth value along with the standard deviation for the sample. The main challenge of this approach is the lack of GT data, therefore we explored how to compose a dataset for training the depth map consistency. The main motivation behind this is that MVS sometimes ignores or fails to reconstruct some parts of surfaces, as we can see in figure 5.1.

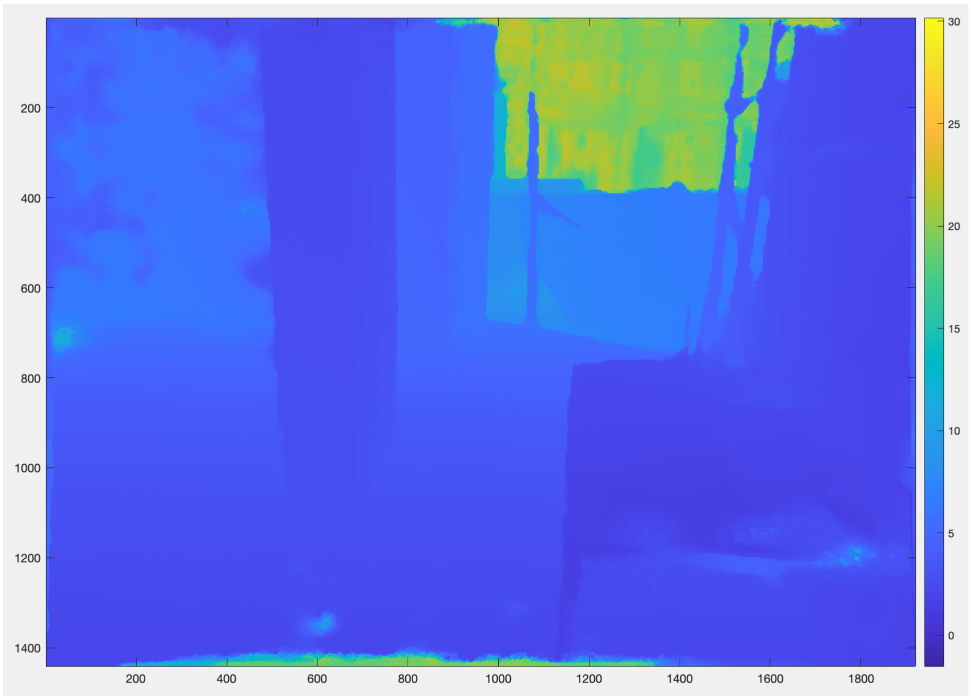


Figure 5.1: Example of error of MVS depth maps - we can see that the table in the lower right corner is not reconstructed perfectly and some artifacts occur.

LiDAR does also impose some problems, specifically with reconstructing black surfaces like PC monitors, because of the light absorption of the surface. Therefore this pixel-wise approach, should consider these imperfections produced by both MVS and LiDAR, and allow for higher quality depth maps by combining the data together.

The following parts explain the creation of the dataset, structure of the neural network, its training and testing.

5.1.1 Dataset

To identify inconsistencies, specifically depth inaccuracies, we first reprojected each depth value from the reference image into n source images, obtaining the geometrically corresponding pixel coordinates in each of the source images. Subsequently, we extracted the corresponding depth values from the source images. Utilizing these depth values, along with multiple depth measurements obtained from the iOS LiDAR scanner and Multi-View Stereo (MVS) systems, we reconstructed the 3D points, using 4.13, in the world coordinate system for each source image. This approach allowed for a comprehensive assessment of depth information consistency by comparing these reconstructed 3D points against the 3D point reconstructed from the reference image's (LiDAR) depth value, using 4.12.

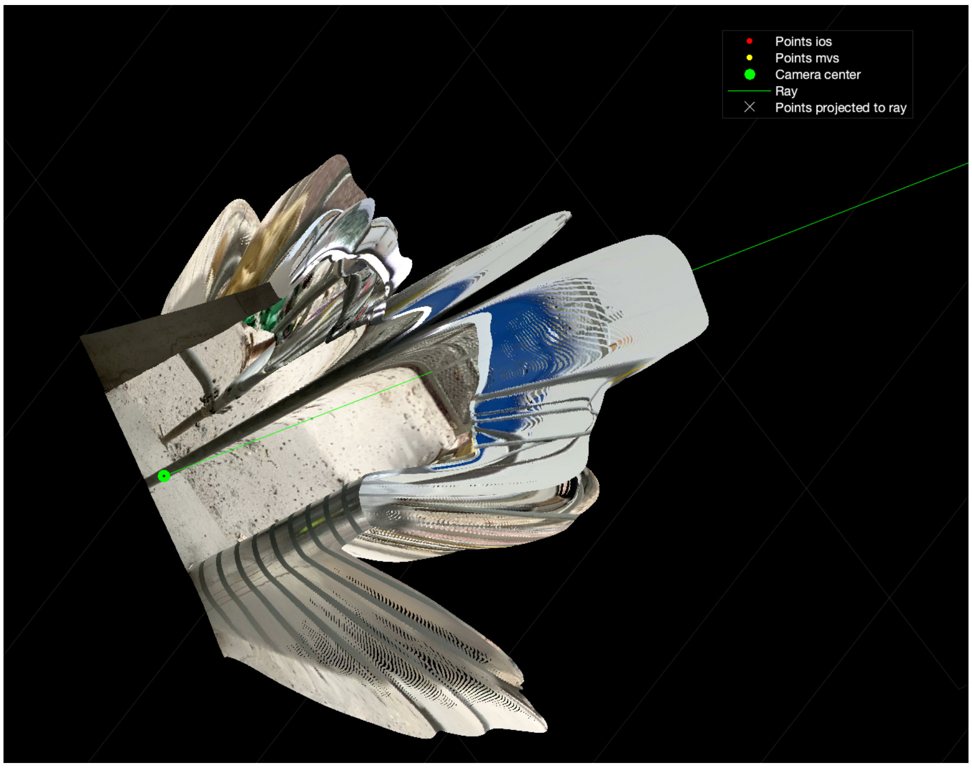


Figure 5.2: Ray in the point cloud connecting camera center, aiming at our chosen pixel. All the original and reprojected depth values from MVS and LiDAR are aligned on to the ray, which could be seen better on 5.3.

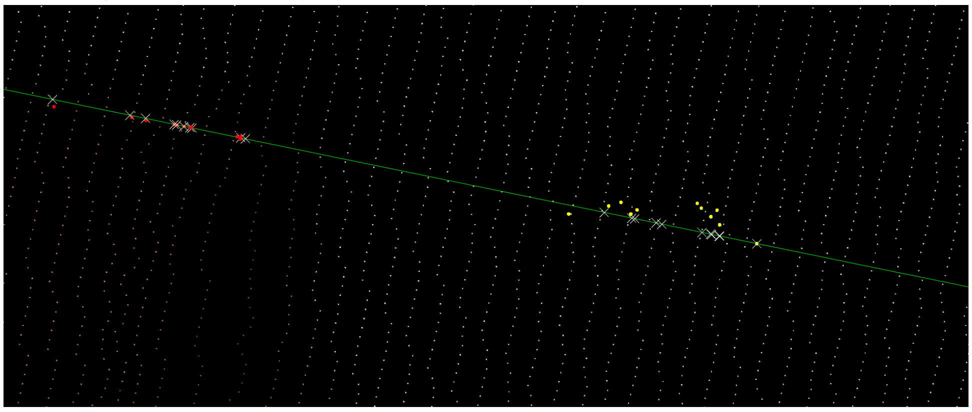


Figure 5.3: Ray in the point cloud from 5.2 - zoomed in.

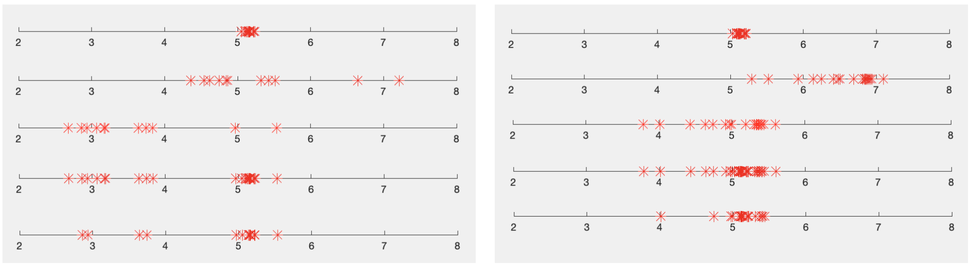
To systematically evaluate the distances and inconsistencies among the reconstructed 3D points, we projected them onto a ray originating from the camera center and aimed at the 3D point reconstructed from the reference image depth value, as visualized in Figures 5.2 and 5.3. This projection facilitated a more straightforward analysis by aligning the points along a single dimension.

Using the projections of 3D points onto the ray, we analyzed the mean and standard deviation of the distances from the camera center, employing the standard deviation as a criterion to identify outlier—points significantly deviating from the reference 3D point on the ray. Since we did not know what can be classified as inlier or outlier in the samples, we assumed that the clusters of points falling into a experimentally selected threshold of standard deviation are correct (containing no outliers) and clusters not falling into this threshold were classified as incorrect (containing outliers). That led us to the idea to manually craft artificial samples that will follow the same distribution of data and train a neural network to filter out the outlier values.

The learning dataset was crafted by combining correct samples with incorrect samples and subsequently subsampling. This strategy ensured the generation of good enough training data from non-ideal inputs (artificial samples with outliers) aligned with correct outputs (mean of samples not containing outliers), laying a solid foundation for our analysis of depth information consistency across images. To prepare the samples for neural network processing, we applied a softmax transformation to normalize the data, ensuring that each input vector sums to one, thereby resembling a probability distribution.

The MATLAB function that calculates the softmax, creates a normalized histogram for a list of distances within a specified range from 0 to 25.5 meters (256 bins), using bins of 0.1 units each. It determines the appropriate bin for each distance, caps the maximum bin index at 256, and accumulates counts. After processing all distances, it normalizes these counts by the total number of distances to form a probability distribution, where each entry in the resultant counts vector represents the proportion of distances that fall into each bin. This effectively provides a binned density estimate of the distance distribution. An example can be seen in 5.5.

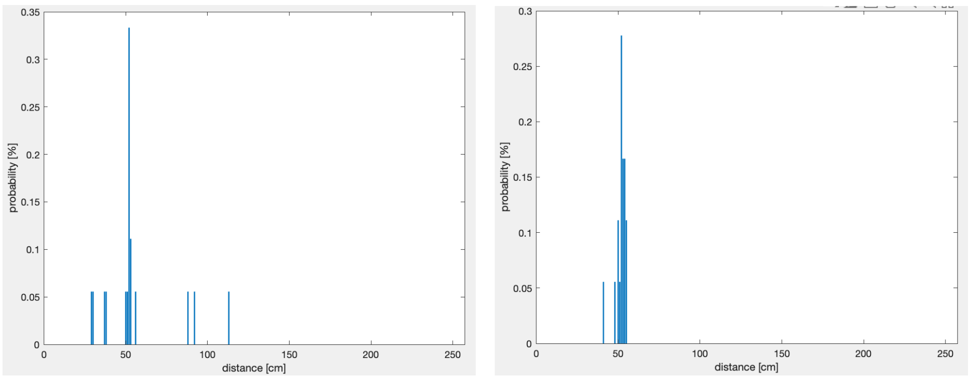
The following image 5.4 shows the construction of two dataset input samples.



(a) : sample a) **(b) :** sample b)

Figure 5.4: Data sample creation
1) 'no outlier' sample
2) sample with outliers
3) outlier sample 2) shifted to match mean and median of 'no outlier' sample
4) 'no outlier' sample 1) and shifted outlier sample 3) joined
5) subsampled version of 4) (finalized input before softmax)

Then the finalized input sample is ran through softmax to create the input for the neural network (figure 5.5).



(a) : sample a) **(b) :** sample b)

Figure 5.5: Softmaxes of samples a) and b) from 5.4.

For each input generated the training output is only mean and standard deviation of the 'no outlier' sample 1) in 5.4.

While generating the testing data a reference image is chosen, and n depth maps are reprojected into the reference camera. For each pixel, a ray is constructed, and the corresponding reprojected 3D values are collected and stored as distances on the ray.

The final dataset was obtained from 4 random images, creating a total of 50 611 samples.

5.1.2 Neural Network

The neural network runs pixel by pixel, aiming to predict the optimal mean depth value and standard deviation from the ray distances, i.e„ we can see it as a filter of outlier values that were artificially added into the training samples.

Structure

Our model, implemented using PyTorch's neural network module (nn.Module), consists of a simple feed-forward neural network designed for tasks requiring an output of two values, based on an input vector of size 256. Since this approach is pixel-wise, we process each pixel independently, therefore for each image we need to process 1920x1440 pixels. The network architecture is detailed as follows:

- **Input Layer:** Accepts input vectors of 256 units for each pixel. These 256 units represent the softmax distribution of point distances from camera center.
- **First Hidden Layer:** Comprises 128 neurons, employing the ReLU (Rectified Linear Unit) activation function to introduce non-linearity, enabling the model to learn data patterns.
- **Second Hidden Layer:** Similar to the first, this layer contains 128 neurons with ReLU activation, augmenting the model's ability to capture intricate relationships in the data.
- **Output Layer:** A fully connected layer with 2 output neurons. This layer outputs mean and standard deviation for each data sample (pixel).

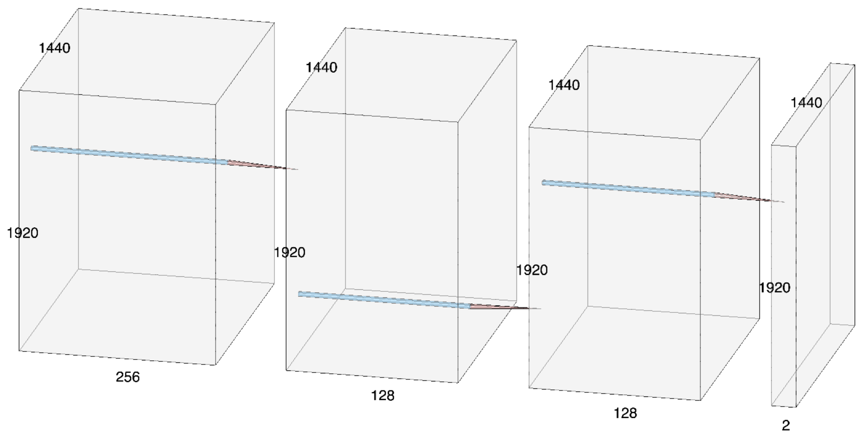


Figure 5.6: Structure of pixel-wise neural network.

Loss Function

To evaluate the performance of our network, we devised a loss function that incorporates both the discrepancy in the mean and the standard deviation between the network's predictions and the ground truth. Specifically, the loss function calculates the absolute difference in the standard deviation (Δstd) and the mean ($\Delta \overline{mean}$) values, combining these two metrics into a single measure of error. The weight assigned to each component was empirically determined to ensure that both the variance and the mean discrepancies contribute equally to the overall loss.

Another option would be using a already known metric called Kullback–Leibler divergence [11], however while experimentally testing with this loss, we observed a non converging trend during training. Therefore, we chose a more straightforward loss function in order to compare the data generated from the NN with the reference.

The variables are defined as follows:

- std_{GT} and $\overline{mean_{GT}}$ represent the "ground truth" standard deviation and mean, respectively.
- std_{NN} and $\overline{mean_{NN}}$ denote the standard deviation and mean predicted by the neural network.

The components of the loss function are calculated as:

$$\Delta std = |std_{GT} - std_{NN}|, \tag{5.1}$$

$$\Delta \overline{mean} = |\overline{mean_{GT}} - \overline{mean_{NN}}|. \tag{5.2}$$

The overall loss is then defined as:

$$Loss = \Delta std + \omega \Delta \overline{mean}, \tag{5.3}$$

where ω=0.05 is the weight applied to the mean difference, chosen through experimental validation.

Training and Other Parameters

For our model's optimization, we utilized the Stochastic Gradient Descent (SGD) technique. SGD differs from traditional gradient descent by updating parameters iteratively using the gradient of the loss function calculated from a single sample rather than the entire dataset. This approach significantly reduces memory usage and computational demand.

Experimentation led us to select a learning rate of 0.01, as it facilitated the most rapid convergence compared to other tested values, which either failed to converge or did so more slowly.

Our neural network underwent training over 100 epochs, with a batch size of 50,611 samples. Each epoch shuffled the samples to ensure the model did not learn any potential sequence patterns in the training data. The accompanying graph 5.7 illustrates the progression of the loss metric throughout the training process, showcasing the model's learning curve.

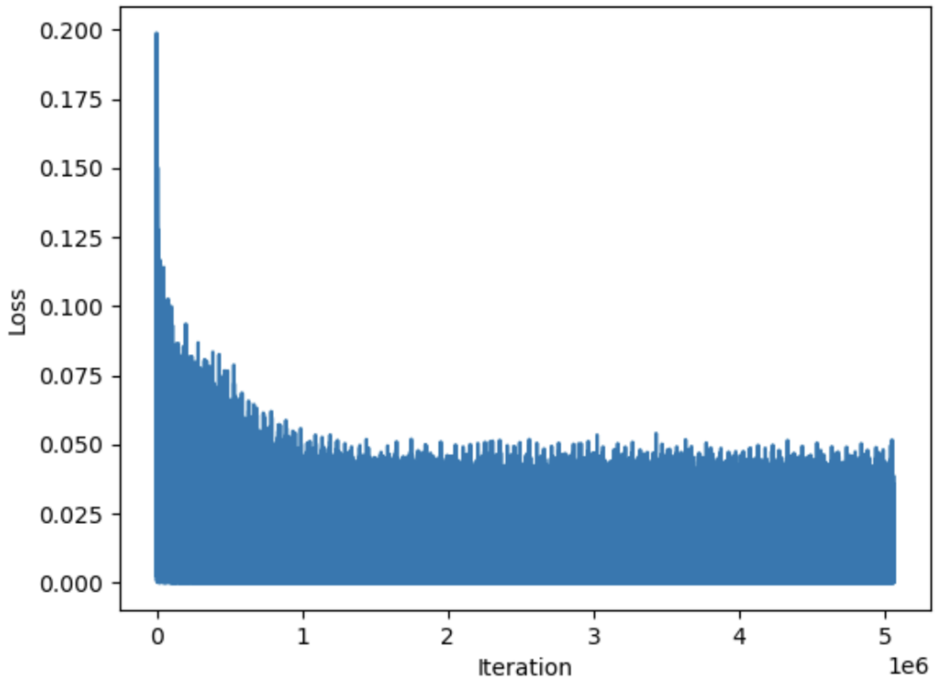


Figure 5.7: Loss vs. iteration during training of the pixel-wise NN on training dataset.

Validation and Testing

For the evaluation phase, we chose an image entirely outside the training dataset to test the network's generalization ability. I processed this image through our model, computing distances for each pixel to obtain a thorough overview. As a result, the network provided a unique mean and standard deviation for the image as a whole. The subsequent images 5.8 5.9 illustrate the outcomes generated by our neural network, highlighting its capability to interpret and summarize unseen data. The following images are visual examples and the analytical evaluation can be found in Section 6.

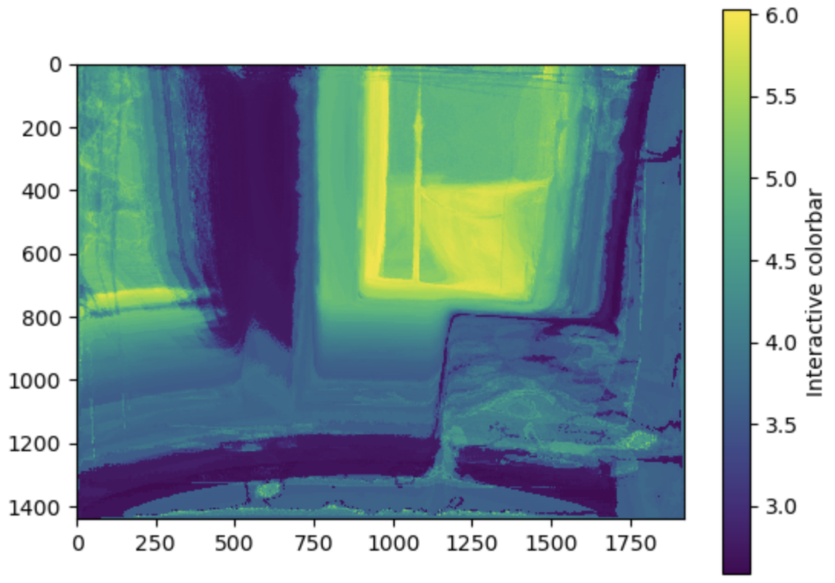


Figure 5.8: Mean estimated by the NN on testing dataset.

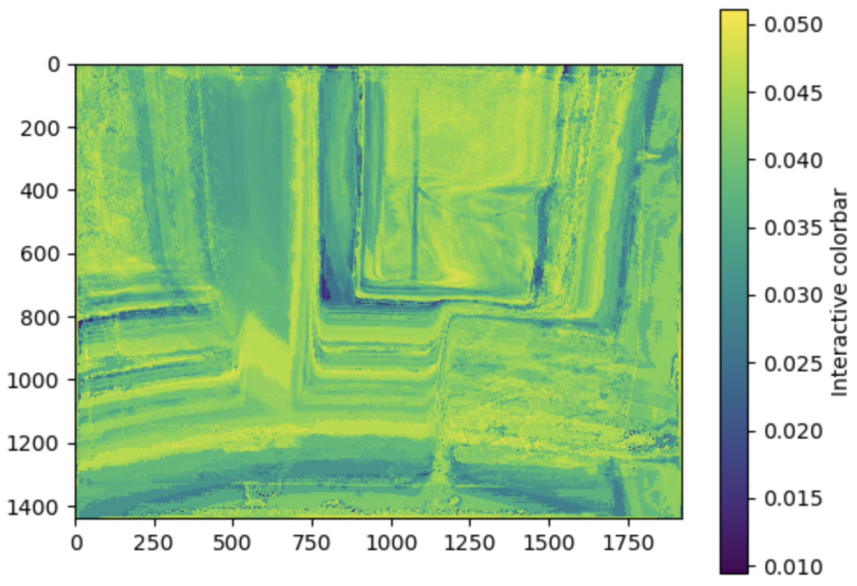


Figure 5.9: Standard deviation estimated by the NN on testing dataset.

5.2 Convolution Approach

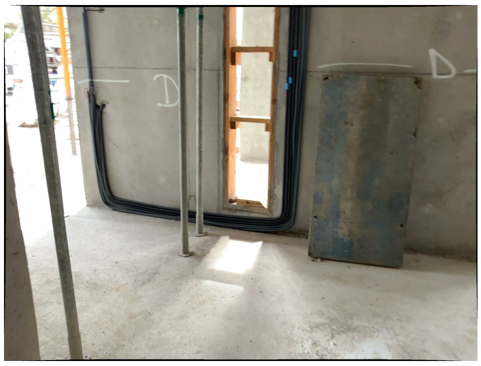
In our exploration of depth map consistency, we introduced a second technique involving a convolutional neural network (CNN). This method entails processing a reference image alongside source images that have been repositioned into the reference camera's coordinates. The goal is to produce a depth map that aligns with all the updated depth maps. The following sections explain the approach in more detail.

5.2.1 Dataset

The dataset was crafted using the reprojection formulas 4.9, 4.10 and 4.11. The key idea of the dataset creation was to create n-tuples of images with high enough overlap of keypoints. For each set, every image once served as the reference image, with the remaining images reprojected to its camera coordinates.

CNN input includes images featuring five distinct layers: the conventional RGB channels, a depth layer from LiDAR data, and a depth estimation from MVS. These layers from source images were then aligned to the reference image. Consequently, for a single image, the input dimensionality scales to $n \times 5 \times \textit{image width} \times \textit{image height}$, given n images per batch. Therefore, the entirety of the batch expands to $n \times n \times 5 \times \textit{image width} \times \textit{image height}$

To better illustrate the dataset's structure, we have included figures that depict the reprojection process for both RGB and depth values from a randomly chosen sample within our RGB-D image collection 5.11. The figures showcase the transformation of RGB values from the source image to match the reference frame 5.11a and similarly, the adjustment of depth values sourced from LiDAR data to align with the reference perspective 5.11b.


(a) : Reference image RGB.

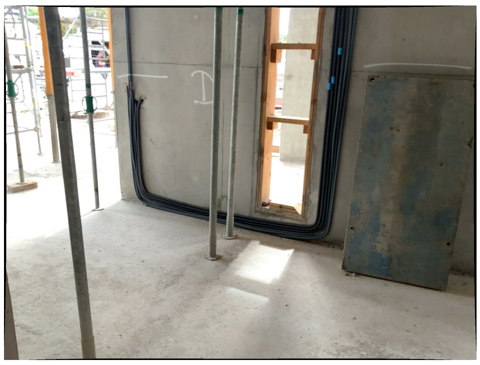
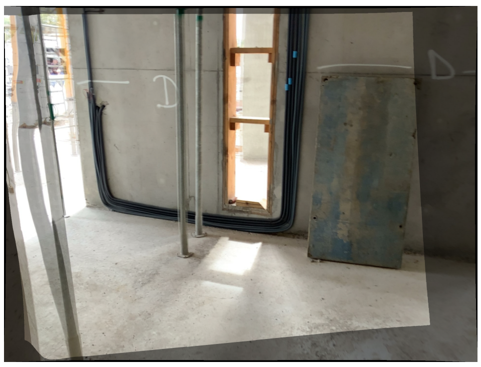

(b) : Source image RGB.

Figure 5.10: Reference and source RGB images.


(a) : RGB reprojected from source to reference.

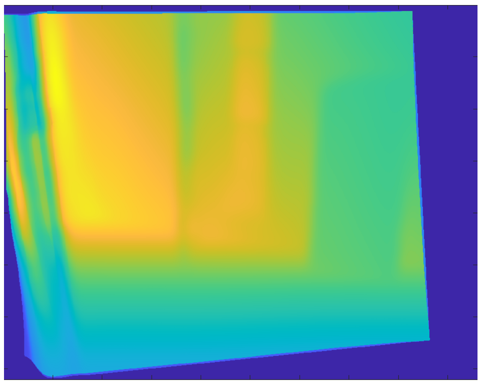

(b) : Depth (LiDAR) reprojected from source to reference.

Figure 5.11: Reprojections from source to reference.

In addition, Figure 5.12 provides a visualization of the input configuration when handling a batch size of three:

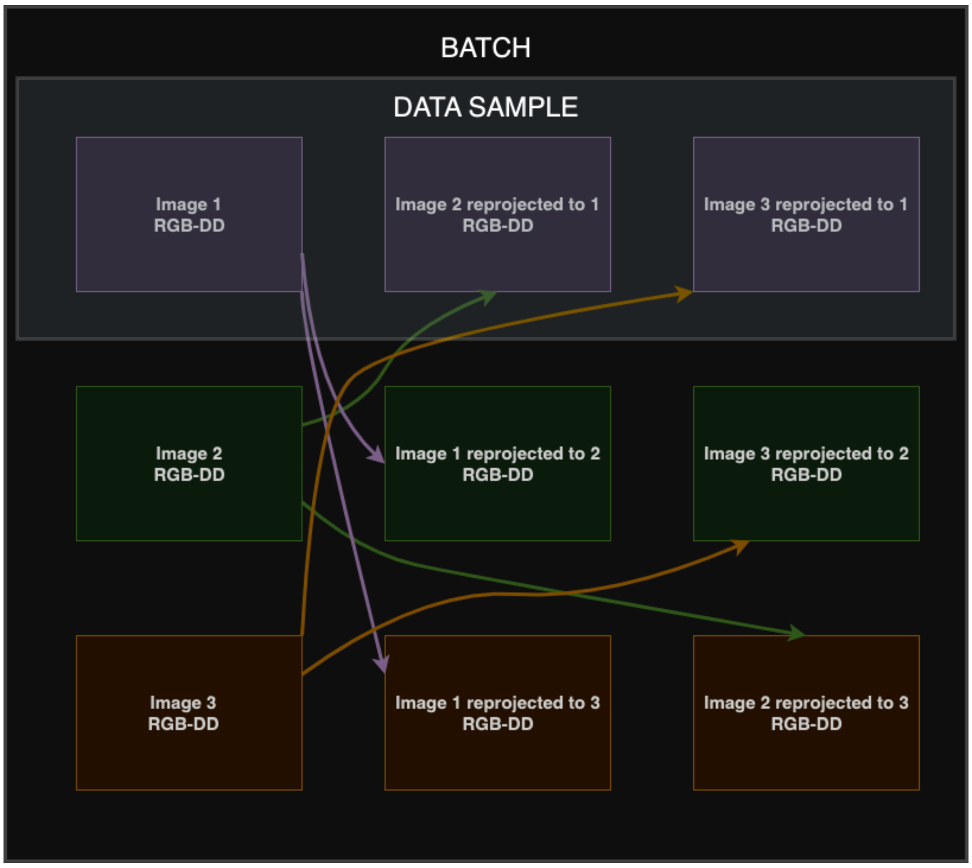

BATCH
DATA SAMPLE
Image 1 RGB-DD
Image 2 reprojected to 1 RGB-DD
Image 3 reprojected to 1 RGB-DD
Image 2 RGB-DD
Image 1 reprojected to 2 RGB-DD
Image 3 reprojected to 2 RGB-DD
Image 3 RGB-DD
Image 1 reprojected to 3 RGB-DD
Image 2 reprojected to 3 RGB-DD

Figure 5.12: The figure show the structure of the data that are passed to the CNN . We show one batch of 3 data samples composed of RGB-DD images. RGB-DD refers to 5 channels: Red, Green, Blue, Depth MVS and Depth LiDAR. We reproject all five channels of each image into the other images from the batch. Batch size of 3 therefore includes 3 original images and 6 reprojection images

5.2.2 Neural Network

In this section, we describe the architecture and training process of our convolutional neural network (CNN) designed to analyze spatial relationships within images and predict depth with high precision. The CNN processes data through multiple layers, starting from an input layer that handles diverse features to several convolutional layers aimed at effective feature extraction. The detailed structure and visualization of the CNN architecture are presented below (Figure 5.13).

Structure

Our CNN is learning to analyze spatial relationships within images to predict depth with higher precision than the input. Given its complexity, the network processes data through multiple layers, starting from an input that encapsulates a diverse set of features to a series of convolutional layers designed for better feature extraction. The architecture is discussed in the following text and visualized in Figure 5.13.

- **Input Layer:** Accepts multi-channel input corresponding to the number of images times five, incorporating RGB channels, LiDAR depth, and MVS depth. The input is processed through 3x3 convolutional filters to capture initial spatial relationships and features.
- **Convolutional Layer 1:** Employs 32 filters of size 3x3 with padding of 1 to preserve the spatial dimensions of the input, followed by ReLU activation and batch normalization for faster convergence and regularization.
- **Convolutional Layer 2:** Utilizes 64 filters of size 3x3 with padding of 1, including ReLU activation and batch normalization.
- **Convolutional Layer 3:** Contains 128 filters of size 3x3 with padding of 1, including ReLU activation and batch normalization.
- **1x1 Convolutional Layer:** A layer using 128 filters of size 1x1 designed to integrate features without altering spatial dimensions, including ReLU activation and batch normalization..
- **Convolutional Layer 4:** Increases depth with 256 filters of size 3x3 and padding of 1, including ReLU activation and batch normalization.
- **Convolutional Layer 5:** Features 256 filters of a larger size (5x5) with padding of 2, enabling the capture of wider spatial contexts, including ReLU activation and batch normalization.
- **Output Layer:** The final convolutional layer reduces the feature maps to a single output channel using a 3x3 filter with padding of 1, designed to output the estimated depth value for each pixel.

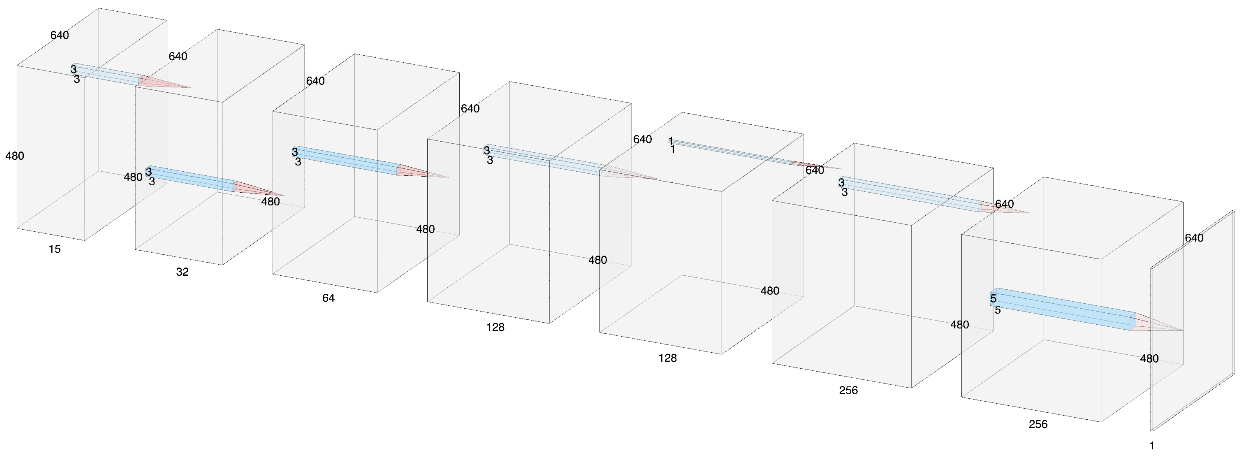


Figure 5.13: Structure of convolutional neural network

Loss Function

In order to evaluate the performance of our CNN, we designed a custom loss function to minimize the difference between all possible reprojections. This involves calculating the mean square error (MSE) not only between each generated image and its original counterpart but also for reprojections of the output depth map onto the other images in the batch. The aggregation of MSEs from each reprojected instance forms our overall loss function.

The diagram 5.14 shows an example for a batch with 3 images.

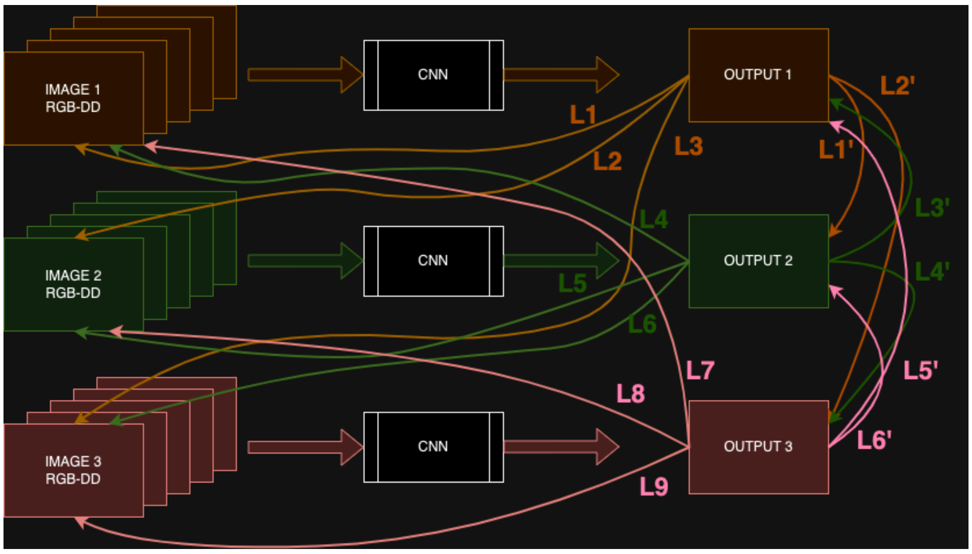


Figure 5.14: CNN loss function diagram. Values $L1', ..., L6'$ account for reprojections between outputs and values $L1, ..., L9$ values account for reprojections into the original image, to ensure similarity of the generated image with the input.

The first data sample produced a depth map (OUTPUT 1) which was then compared to its original input using mean square error (MSE, denoted as L_1). Additionally, this output depth map was reprojected into the camera coordinate systems of the second and third images, with the MSEs of these reprojections being calculated (L_2, L_3). This process was similarly applied to the depth maps generated for the second and third images by the CNN.

We also needed to ensure reprojection consistency between the outputs. Therefore, we reprojected the outputs into each other and calculated the MSE between those reprojections $(L'_i,\ i \in \{1, 2, 3, 4, 5, 6\})$.

The total loss function can be then calculated as:

$$Loss = \gamma \sum_{i=1}^{n\times n} L_i + (1-\gamma) \sum_{i=1}^{n\times(n-1)} L'_i \tag{5.4}$$

where n represent the number of images and γ is a parameter to scale weights of the losses based on their importance.

The formula for loss function can be therefore generalized as:

$$Loss = \gamma(\sum_{i=1}^{n}\sum_{j=1}^{n} L_{i\to j}) + (1-\gamma)(\sum_{k=1}^{n}\sum_{l=1}^{n} L'_{k\to l}),\ \mathit{where}\ k \neq l \tag{5.5}$$

where:

$$L_{i\to j} = \mathrm{MSE}(D_j, D_{\bar{i}\to j}) \tag{5.6}$$

where D_i is the LiDAR depth map of the i-th image, $D_{\bar{i}}$ is the CNN depth map of the i-th image and $D_{\bar{i}\to j}$ is the CNN output depth map from image i reprojected to camera coordinates of j-th image and MSE is the function for mean square error and γ is a weight used to find a balance between similarity of generated depth maps and depth consistency of the outputs. Consequently:

$$L'_{k\to l} = \mathrm{MSE}(D_{\bar{l}}, D_{\bar{k}\to l}) \tag{5.7}$$

where $D_{\bar{l}}$ is the depth map of the l-th image generated by the CNN and $D_{\bar{k}\to l}$ is the CNN output depth map of k-th image reprojected to camera coordinates of l-th image and MSE is the function for mean square error.

In summary L represents the comparison of CNN outputs with the input depth map, while L' represents the comparison of CNN outputs with the remaining CNN outputs. It is important to notice that the complete loss function accounts for total of 15 reprojections (that are also weighted by some coefficient γ) for each batch, when the number of images in batch is equal to 3. Therefore the final values of loss function does not directly correspond to the mean squared error of the reprojections in meters. If the average MSE per reprojection is be desired as an indicator, it is necessary to rescale the total loss accordingly by the number of reprojections and their corresponding weights.

Training and Other Parameters

The CNN had two training strategies. The first strategy, that can be interpreted as a depth map optimization, involved training on a single batch of n images, each reprojected to match the others, aiming to derive an optimal depth map consistent across all reprojections. The second strategy expanded the scope to multiple batches, encompassing various viewpoints. This approach intended for the CNN to assimilate information about the entire scene, rather than focusing solely on consistency from a singular viewpoint.

Single Batch Training

To speed up the training process, we utilized GPU acceleration, faced with the limitation of our graphics card's 11GB memory capacity, which introduced additional challenges. More specifically the GPU we trained on was NVIDIA GeForce GTX 1080 Ti. By training on local PC, we made the creation of virtual environment more simple and we did not need to create docker or singularity containers or deal with any similar problems. Initially, we aimed to train with batches of 5 images at the full resolution of 1920x1440, but this ambition was quickly curtailed by memory constraints, causing CUDA to run out of memory. Our first adjustment involved reducing the batch size to 3 images, but even then, the memory was insufficient. Consequently, we were compelled to downsample the images to 640x480 pixels, a reduction to one-third of their original size, which finally allowed the data to be accommodated by the GPU.

The training was conducted over 2000 epochs, employing Stochastic Gradient Descent (SGD) with a learning rate of 0.01. Although increasing the learning rate to 0.1 hastened convergence, it resulted in a notably higher loss function value. A learning rate of 0.01, on the other hand, struck a balance between convergence speed and optimal loss function value at the end of the training. SGD was used for the same reasons as outlined in 5.1.2. The value of γ was set to 0.1 during the whole training in order for the model, to keep higher emphasis on the reprojection error, than the similarity of the images.

The accompanying graph 5.15 illustrates the progression of the loss metric throughout the training process, showcasing the model's learning curve.

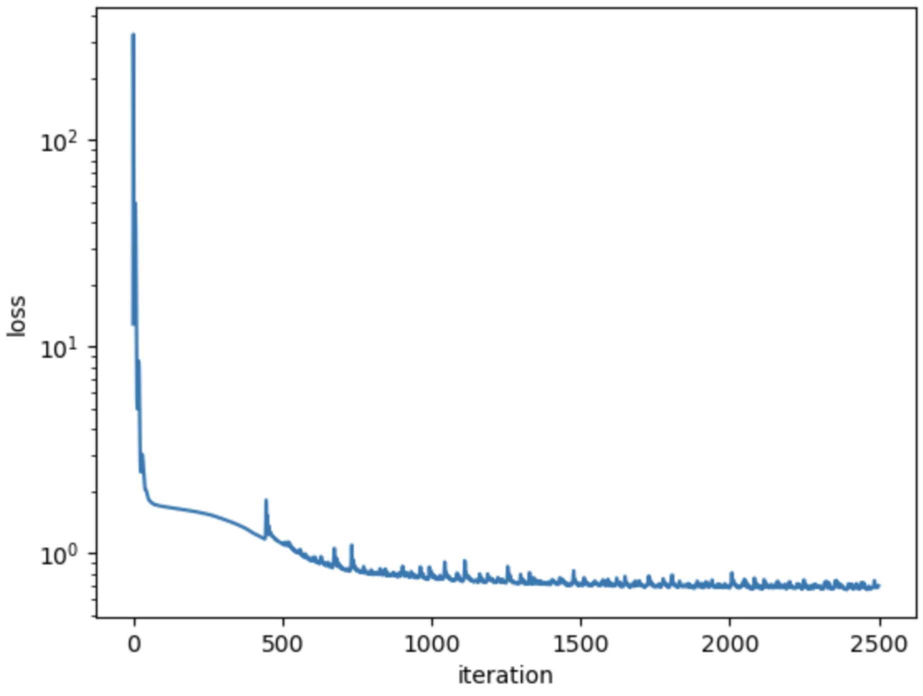


Figure 5.15: Loss vs. iteration during training of the CNN on the training dataset - single batch.

Multi Batch Training

For the multi batch training, we trained on 10 batches of 3 images each. We only trained on this amount of data, since our training process took around 50 hours on the GPU available. Therefore, training on more data would be possible with bigger GPU memory available. We used equal configurations as in the single batch training, only increasing the number of epochs to 2500. The following figure 5.16 shows the learning curve of the CNN.

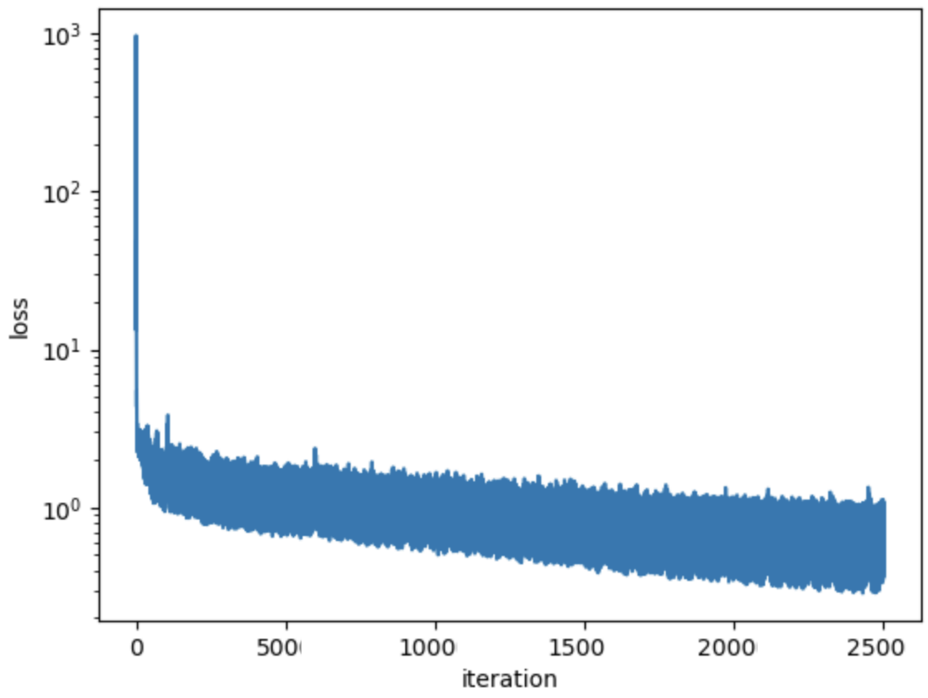
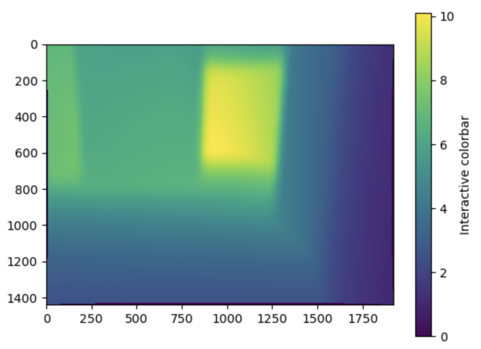


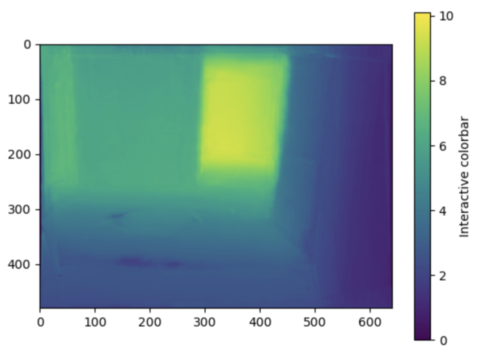
Figure 5.16: Loss vs. iteration during training of the CNN on the training dataset - multi batch.

Validation and Testing

Following the training phase with a single batch of images, we conducted a test by processing one image from the batch through the model to verify the accuracy of the output. Before delving into an analysis and discussion of the outcomes, we will first showcase the CNN-generated depth maps alongside the original depth maps acquired through iPad for comparison in Figures 5.17 and 5.18.



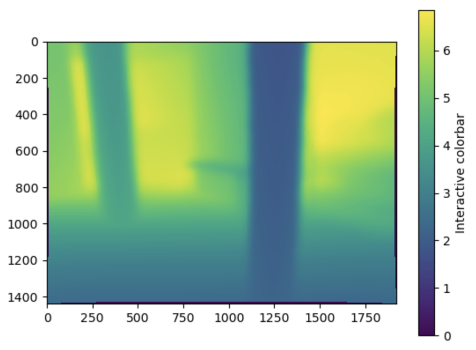
(a) : Depth map captured using LiDAR.



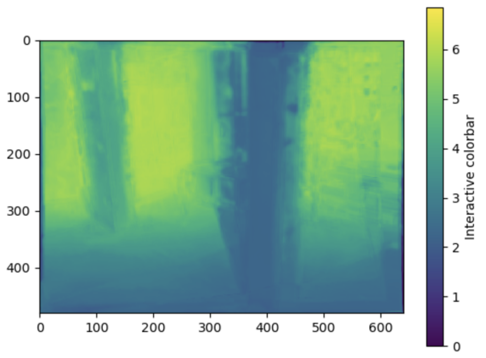
(b) : Depth map produced by the CNN on an image from testing dataset.

Figure 5.17: Comparison of original depth map with the depth map produced by the CNN - known data.

After training on multiple batches, we tested how the model performs on unseen data. The following images show the example comparison of the original depth map and depth map generated by the CNN. More in-depth evaluation can be found in 6.



(a) : Depth map captured using LiDAR.



(b) : Depth map produced by the CNN on an image from testing dataset.

Figure 5.18: Comparison of original depth map with the depth map produced by the CNN - unknown data.

Chapter 6

Experimental Results

This section of the thesis presents analytical results comprising mostly of tables and visualizations. Visualisation of the depth maps generated by the NNs can be found at figures 5.8, 5.9, 5.17 and 5.18.

6.1 Pixel-Wise Approach

In the process of evaluating the effectiveness of our pixel-wise neural network model for depth map enhancement, we conducted a comparison against the depth maps obtained from LiDAR measurements. This comparative analysis aimed to quantify the accuracy of our model-generated depth maps in relation to the LiDAR data. To achieve this, we employed two principal metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE). The following figure 6.3 visualises the process of measuring MSE and MAE.

The Mean Squared Error emphasizes the penalization of larger errors, offering a perspective on the average of the squares of errors. Conversely, the Mean Absolute Error offers an intuitive measure of the average magnitude of errors between the predicted and actual values.

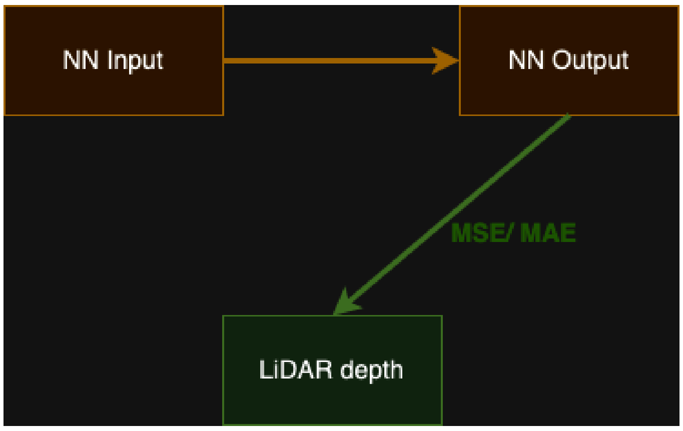


Figure 6.1: Visualisation of computation of MAE and MSE from pixel-wise approach.

Findings from the comparison are summarized in Table 6.1.

Metric	**Value**
MSE $[m^2] \downarrow$	2.2557
MAE $[m] \downarrow$	1.1131

Table 6.1: Comparison of Neural Network-Generated Depth Maps against LiDAR Data.

The findings summarized in Table 6.1 reveal certain limitations in the model's ability to replicate depth maps with the same fidelity as those obtained from LiDAR measurements. The MSE and MAE values, while providing a measure of the model's performance, indicate notable discrepancies between the generated depth maps and the LiDAR data.

We also selected four random samples from the validation dataset and visualized the performance of the NN (figure 6.2). The green bars depict the original histogram (or parts of it, since the x axis was limited for viewing clarity) that was the input for the NN. Red line shows the estimated depth value and blue line represents the LiDAR depth value. The red dotted lines represent the estimated standard deviation of the sample.

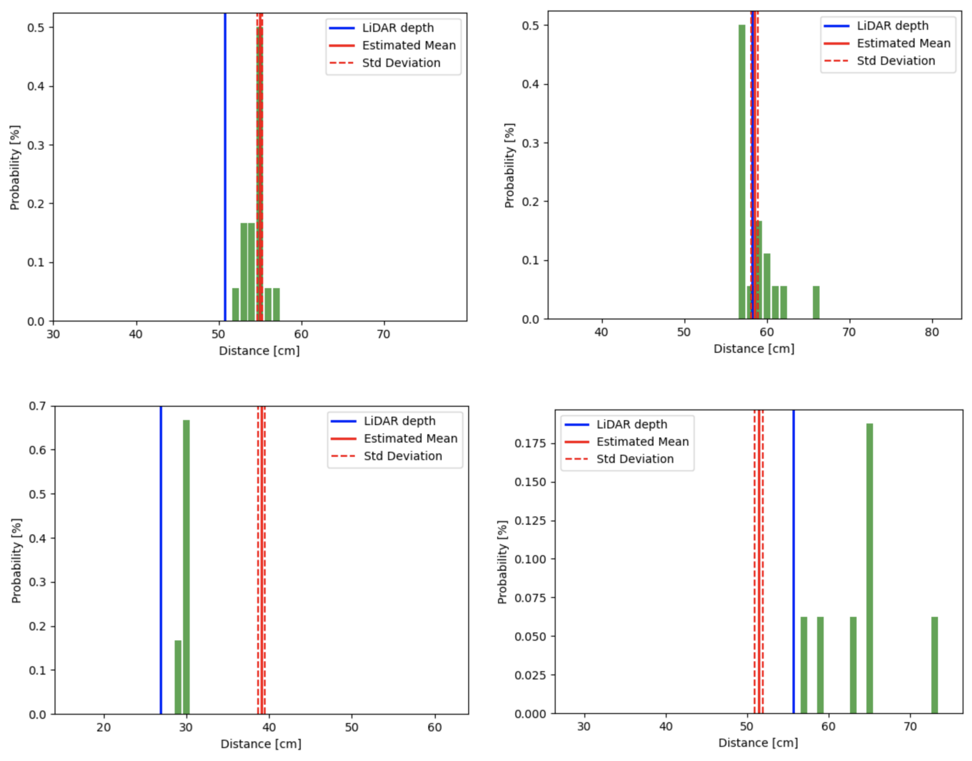


Figure 6.2: Four random samples ran through the NN. Green bars represent input softmax histogram, blue line represent depth value from LiDAR and red line represents estimated depth value (the estimated mean in top left figure is exactly covering the highest peak of the input histogram).

From the figure 6.2 we can see that sometimes the predictions are statistically more accurate than the LiDAR depth, however, in some samples the estimated mean does not correspond to reality and the estimation is wrong.

The next figure 6.3 visualizes the error between the LiDAR data and the generated depth maps by thresholding differences. Pixels marked in yellow indicate locations where the discrepancy is less than 10 centimeters, providing a clear visual representation of areas where the model achieves an error margin below this threshold.

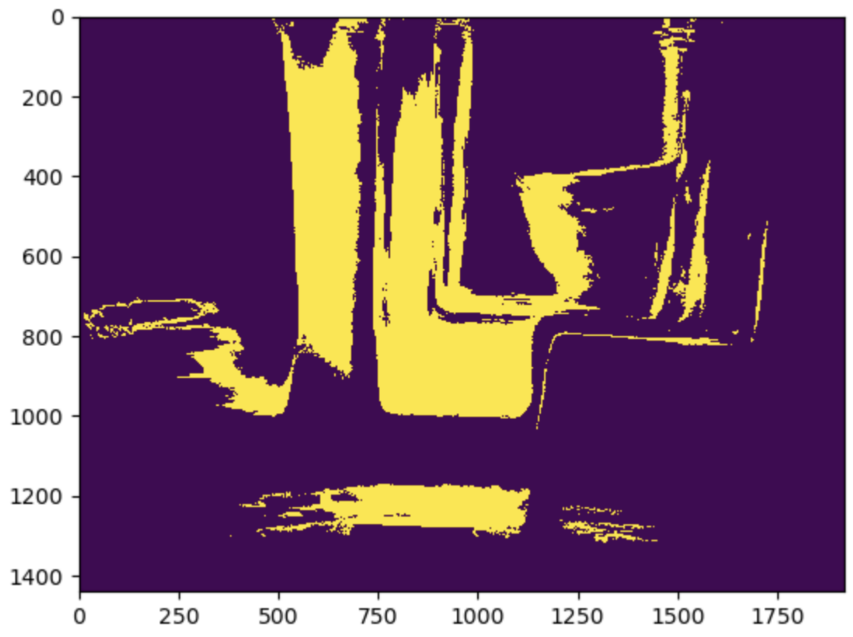


Figure 6.3: Depth Map Error Visualization: Errors Under 10cm are highlighted by yellow color.

6.2 Convolution Approach

To assess the effectiveness of the convolutional approach, we introduced a metric that calculates the average Mean Square Error (MSE) by reprojecting each generated depth map into the coordinate system of the other images within the same batch. The computation of this specific MSE variant is defined as follows:

$$\overline{\text{MSE}} = \frac{\sum_{j=1}^{n} \text{MSE}(D_j, D_{i \to j})}{n-1}, \quad \text{for } j \neq i \tag{6.1}$$

Here, n denotes the total number of images in the batch, D_j represents the reference depth map for reprojection, and $D_{i \to j}$ signifies either the original depth map or the depth map produced by the CNN, reprojected into the perspective of camera j.The mechanism of this evaluation metric is visually demonstrated in figure 6.4.

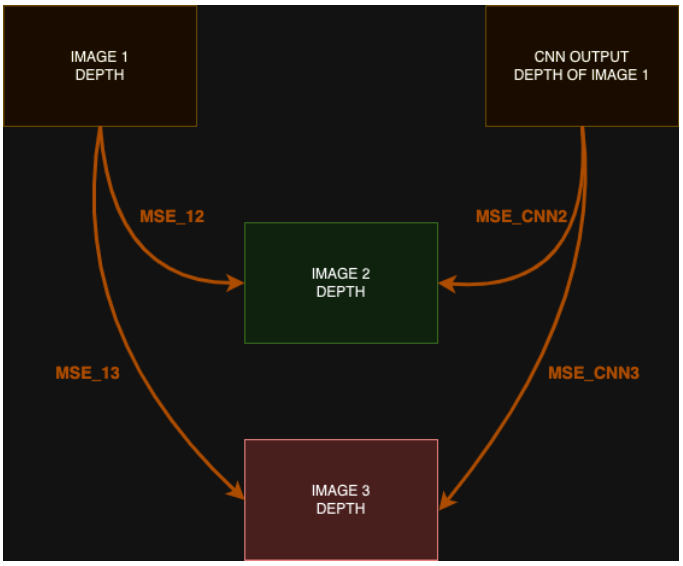


Figure 6.4: Evaluation metric of CNN. Mean value of MSE_{12} and MSE_{13} gives us the column labeled Original MSE in tables 6.2 and 6.3, while mean of values $\mathrm{MSE}_{\mathrm{CNN2}}$ and $\mathrm{MSE}_{\mathrm{CNN3}}$ give us the column CNN MSE.

For a comprehensive analysis, we compared the average MSE between the original depth maps and those generated by the CNN across several images. The comparison for two image batches is summarized in the tables 6.2 and 6.3. In the tables, Image $[b, i]$ stands for: b is a batch number and i is an id of the image in the batch. Batches 0 and 1 represent data already known by the CNN, while batches 2 and 3 represent unseen data:

Image $[b, i]$	Original MSE $[m^2]\downarrow$	CNN MSE $[m^2]\downarrow$
Image [0, 0]	2.3546	1.8546
Image [0, 1]	3.1723	2.2389
Image [0, 2]	1.3083	1.1081
Image [1, 0]	4.8962	3.6047
Image [1, 1]	0.9752	0.5431
Image [1, 2]	2.2383	1.7133

Table 6.2: Comparison of Average MSE between Original and CNN-Generated Depth Maps - known data.

Image $[b, i]$	Original MSE $[m^2]\downarrow$	CNN MSE $[m^2]\downarrow$
Image [2, 0]	3.7491	2.9832
Image [2, 1]	2.5195	1.9903
Image [2, 2]	1.7890	1.9082
Image [3, 0]	5.8700	5.3446
Image [3, 1]	3.0687	2.2729
Image [3, 2]	3.4750	3.6931

Table 6.3: Comparison of Average MSE between Original and CNN-Generated Depth Maps - unknown data.

Since we are calculating the Mean Squared Error (MSE), we also filtered out large values of the squared error to exclude outliers, as these outliers can significantly impact the MSE. This allows us to demonstrate that our results are not solely dependent on outliers. Tables 6.4 and 6.5 show the MSE values with values greater than 1 filtered out.

Image $[b, i]$	Original MSE - no outliers $[m^2]\downarrow$	CNN MSE - no outliers $[m^2]\downarrow$
Image [0, 0]	0.1209	0.0693
Image [0, 1]	0.2423	0.1560
Image [0, 2]	0.0890	0.0882
Image [1, 0]	0.0929	0.0568
Image [1, 1]	0.1291	0.0870
Image [1, 2]	0.0593	0.0820

Table 6.4: Comparison of Average MSE between Original and CNN-Generated Depth Maps, with filtered outliers where squared error was ≥ 1 - known data.

Image $[b, i]$	Original MSE - no outliers $[m^2]\downarrow$	CNN MSE - no outliers $[m^2]\downarrow$
Image [0, 0]	0.0831	0.0801
Image [0, 1]	0.0792	0.1445
Image [0, 2]	0.0720	0.1526
Image [1, 0]	0.0929	0.0568
Image [1, 1]	0.1291	0.0870
Image [1, 2]	0.0593	0.0820

Table 6.5: Comparison of Average MSE between Original and CNN-Generated Depth Maps, with filtered outliers where squared error was ≥ 1 - unknown data.

Additionally, we assessed the Mean Square Error (MSE) of the reprojections from the output depth maps into the coordinate systems of other output images, aiming to verify the consistency of the depth maps produced by our CNN. This involved contrasting these MSE values against those derived from the reprojections of the original LiDAR depth maps. The process for calculating this specific metric is illustrated in the diagram below (Figure 6.5).

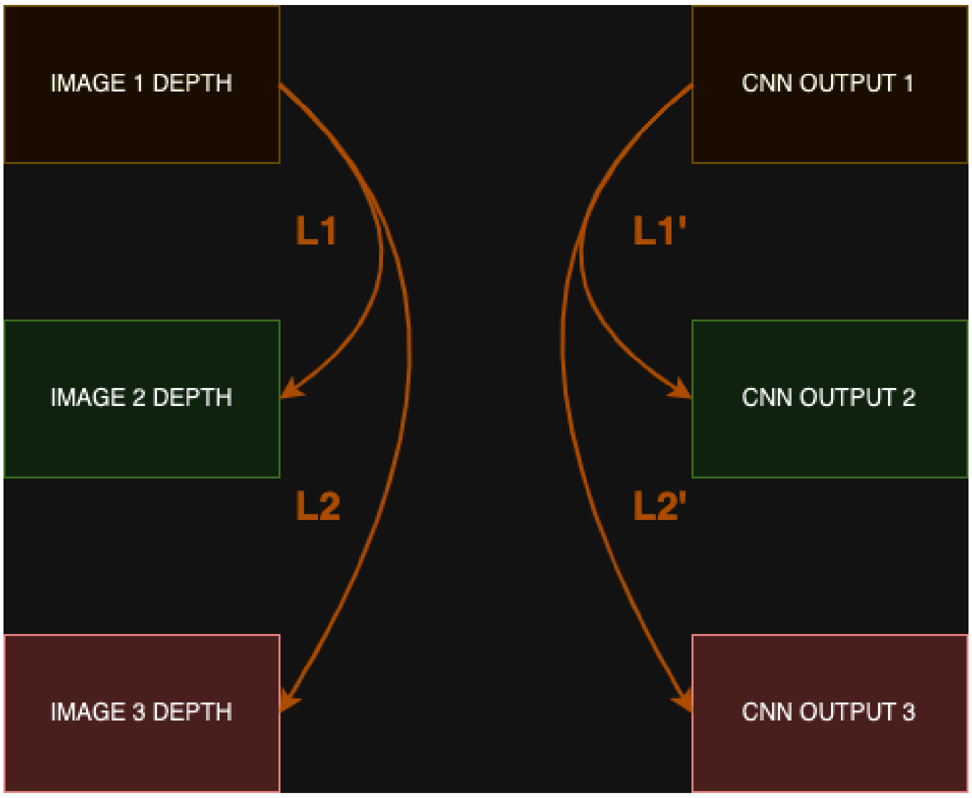


Figure 6.5: 2nd evaluation metric of CNN. Average value from L_1 and L_2 can be found in columns Original MSE in tables 6.6 and 6.7, while the average of values L'_1 and L'_2 is represented in the colums CNN MSE.

Mathematically, the error can be expressed similarly as in equation 6.1, where n denotes the total number of images in the batch, D_j represents the target depth map for reprojection (either original or generated by CNN), and $D_{i \to j}$ realize eiher original or generated D_i depth map transformed to the coordinate system of camera j. It's crucial to compare original depth maps with other original maps and CNN-generated maps with other CNN-generated depth maps exclusively to ensure an accurate evaluation of depth map consistency (tables 6.6 and 6.7).

Image [b, i]	Original MSE [m^2]↓	CNN MSE [m^2] ↓
Image [0, 0]	0.5005	0.1660
Image [0, 1]	1.0295	0.2563
Image [0, 2]	1.4814	0.5509
Image [1, 0]	1.6313	0.0778
Image [1, 1]	3.3306	0.1352
Image [1, 2]	4.3834	0.2702

Table 6.6: Comparison of Average MSE between Original and CNN-Generated Depth Maps - known data.

Image [b, i]	Original MSE [m^2] ↓	CNN MSE [m^2] ↓
Image [2, 0]	0.6092	0.3980
Image [2, 1]	1.3751	1.2091
Image [2, 2]	1.8150	1.6755
Image [3, 0]	2.4363	0.5103
Image [3, 1]	6.0960	1.3986
Image [3, 2]	8.1981	2.0490

Table 6.7: Comparison of Average MSE between Original and CNN-Generated Depth Maps - unknown data.

As with the previous metric, tables 6.8 and 6.9 present the MSE values with filtered outlier values.

Image [b, i]	Original MSE - no outliers [m^2]↓	CNN MSE - no outliers [m^2] ↓
Image [0, 0]	0.1594	0.0374
Image [0, 1]	0.3234	0.0676
Image [0, 2]	0.4485	0.1202
Image [1, 0]	0.1053	0.0322
Image [1, 1]	0.1949	0.0636
Image [1, 2]	0.2528	0.1221

Table 6.8: Comparison of Average MSE between Original and CNN-Generated Depth Maps, with filtered outliers where squared error was ≥ 1 - known data.

Image [b, i]	Original MSE - no outliers [m^2]↓	CNN MSE - no outliers [m^2] ↓
Image [0, 0]	0.1482	0.0901
Image [0, 1]	0.2983	0.2165
Image [0, 2]	0.4355	0.3083
Image [1, 0]	0.1061	0.1056
Image [1, 1]	0.2378	0.2425
Image [1, 2]	0.2829	0.2583

Table 6.9: Comparison of Average MSE between Original and CNN-Generated Depth Maps, with filtered outliers where squared error was ≥ 1 - unknown data.

These results demonstrate that our CNN significantly improved the consistency of depth maps. On both known and unknown data with outliers, we enhanced most depth maps. Although we conducted this experiment with only a few examples, the outcomes indicate promising potential for future research.

Another metric used was the mean absolute error (MAE). This metric quantifies the average absolute difference between the depth values predicted by our CNN and those obtained from our inputs. We conducted comparisons against depth maps acquired through two methods: LiDAR (referred to as MAE LiDAR) and Multi-View Stereo (MVS), to estimate the accuracy of our generated depth maps. Tables 6.10 and 6.11 detail the MAE results from these comparisons across four batches of images.

Image [b, i]	MAE LiDAR [m] $\downarrow$	MAE MVS [m] $\downarrow$
Image [0, 0]	0.3762	1.0977
Image [0, 1]	0.3035	1.0453
Image [0, 2]	0.2312	0.8449
Image [1, 0]	0.5282	0.6032
Image [1, 1]	0.5101	0.4995
Image [1, 2]	0.4470	0.5856

Table 6.10: Mean Absolute Error (MAE) between output and LiDAR/MVS across Two Batches - unknown data.

Image [b, i]	MAE LiDAR [m] $\downarrow$	MAE MVS [m] $\downarrow$
Image [2, 0]	1.3596	3.1582
Image [2, 1]	1.4839	3.2012
Image [2, 2]	1.2330	3.0215
Image [3, 0]	0.6598	1.5514
Image [3, 1]	0.8317	2.4786
Image [3, 2]	0.7242	2.2083

Table 6.11: Mean Absolute Error (MAE) between output and LiDAR/MVS across Two Batches - unknown data.

The table above shows us, the MAE between the outputs and the inputs. It was expected that the MAE of data that were not previously seen by the CNN would be higher, however we believe that if we trained on bigger dataset, the MAE values would also decrease. Also these MAE values could be decreased, if we increased the value of γ in the loss function (5.5). When we compare the input and output of known data, we can observe that the MAE error between output and LiDAR value is always smaller than the difference of depth estimated by MVS and Lidar (0.633 meters) mentioned in section 4.4.1.

Lasty we used Structural Similarity Index (SSIM) to evaluate similarity of our depth maps. After testing numerical precision, we also evaluated the perceptual quality of our generated depth maps using SSIM. This metric considers changes in structural information, brightness, and contrast, offering insights into how closely the predicted depth maps resemble the actual depth maps in terms of visual structures (previously mentioned in section 4.4.1).

The SSIM scores for our depth maps, compared to their ground-truth counterparts across the four batches, are summarized in tables 6.12 and 6.13:

Image $[b, i]$	SSIM $\uparrow$
Image [0, 0]	0.7376
Image [0, 1]	0.8057
Image [0, 2]	0.8215
Image [1, 0]	0.6396
Image [1, 1]	0.6679
Image [1, 2]	0.6987

Table 6.12: Structural Similarity Index (SSIM) across Two Batches - known data.

Image $[b, i]$	SSIM $\uparrow$
Image [2, 0]	0.4514
Image [2, 1]	0.4187
Image [2, 2]	0.4257
Image [3, 0]	0.6058
Image [3, 1]	0.5558
Image [3, 2]	0.5735

Table 6.13: Structural Similarity Index (SSIM) across Two Batches - unknown data.

The last two tables show us, that the network produces images with lower SSIM if the data was not previuosly seen by the network. This issue could again be solved by leveraging the weight γ of the reprojections to the original depth maps described in 5.5.

Chapter 7

Conclusion

The work on the thesis was challenging because of no existing ground truth measurements, a limited amount of data captured in the scope of the EU H2020 Artwin project on Eiffage construction, restriction of the utilized device for capturing the RGB-D images, and given hardware for evaluation and training of proposed methods. We observed that depth data from iPad are more consistent indoors while outdoors images are more consistent from Multiview Stereo (MVS). Moreover, one can see repeating errors, e.g., inaccurate smooth depth values on edges leading to flying points (iPad) or missing poles or texture-less walls (MVS). Therefore, we focused on training an unsupervised method that will merge depth measurements from several sources and make them more consistent across multiple views. As the Ground Truth depth maps are usually not available for specific setups, i.e., device and environment, we employed consistency as our metric and formulated our loss function accordingly.

7.1 Interpretation of Results

As demonstrated in the results section 6, the pixel-wise and convolutional approaches to enhance depth maps using neural networks show promising outcomes. The quantitative analysis indicates that our model improves the consistency of depth maps compared to initial less consistent inputs.

The convolution approach, in particular, has proven effective in handling both known and unknown inputs. As expected the performance is better on previously seen depth maps. However, even though the data generated by our approaches do have lower Structural Similarity (SSIM), lower resolution because of the subsampling, and are trained only on a few samples, we generated depth maps with higher consistency on unseen inputs. The consistency, i.e., the inverse of the distance between the original reprojected depth maps between input RGB-D images is smaller than in the case of updated depth maps. This error reduction underlines the capability of convolutional neural networks to refine spatial estimations derived from multiple perspectives leading to more accurate 3D reconstruction. Users can benefit from an additional step in between meshing and depth map fusion that will make the depth maps more consistent and filter common repetitive distortions of individual depth measurements.

All in all, the pixel-wise approach has proven to be effective, however, it does not satisfy our expectations as much as does the convolutional approach. We believe there is a space for refining it to achieve better results, for example, using patches of input RGB-D images instead of individual pixels. The contextual information plays an important role and therefore, the convolutional neural network leads to better results. Our approaches show that work in this field of study does have the potential for future research. The thesis proposes a novel and innovative view on refining the depth maps before merging them into a single 3D reconstruction.

7.2 Future Work

During the work on this thesis, we found multiple enhancements that could be done to improve the consistency of depth maps.

Firstly, the loss function from figure 5.2.2 could be improved by not comparing the CNN output with the original depth values, but using the updated depth values to reproject RGB images. Therefore, we eliminate the dependence on input depth maps and can employ new loss metrics like Structural Similarity (SSIM). We assume that would allow the loss function to converge to zero error which is not possible now in case of noisy inputs.

Another enhancement would be to use a transformer architecture [55, 57] instead of the NN architecture we proposed in both our approaches. Using this specific kind of NN architecture would allow for superior handling of complex spatial relationships and dependencies within the image data. Transformers, with their self-attention mechanisms, are adept at processing data globally and can capture nuances across entire images rather than just within localized patches. This capability would potentially improve the accuracy and detail of depth map predictions, particularly in challenging scenarios where traditional methods struggle with context integration and depth continuity across varied terrains and object interactions. Furthermore, the flexibility of transformers in learning from diverse data representations could facilitate more robust generalizations to new environments or different types of depth-sensing inputs, enhancing the model's utility across a broader range of applications. However, transformers also bring disadvantages, primarily their high demand for computational resources. They require substantial training data to perform optimally. Additionally, they increase the risk of overfitting, especially with limited datasets.

The third interesting idea arose when discussing the equations 4.12 and 4.13. Let's present the equations one more time for clarity:

$$\underline{\mathbf{X}} = \mathbf{E}_{\text{ref}}^{-1}\left(\mathbf{K}^{-1}\underline{\mathbf{x}}d\right)$$
$$\underline{\mathbf{X}'} = \mathbf{E}_{\text{src}}^{-1}\left(\mathbf{K}^{-1}\underline{\mathbf{y}}d'\right)$$

In our thesis we are optimizing d and d', however, it would be interesting to optimize also the calibration matrices $\mathbf{K}$ and our camera extrinsic $\mathbf{E}$. Optimizing all these values together would be impossible because the problem is unconstrained, i.e., we have more parameters than measurements. However, it would be possible to converge to reasonable values by iteratively switching the values that are to be optimized. Therefore we could first optimize d, based on that refinement we would optimize $\mathbf{E}$ and $\mathbf{K}$, and then we would optimize d again. The benefit is that we could also finetune the camera poses and camera intrinsic leading to effective self-calibration and close to zero reprojection errors.

Chapter 8

Code Directory, Usage Guide and Used Softwares

This chapter gathers all our developed scripts, offering a clear overview and guidance on their application. It's designed as a concise reference to help users navigate through our codebase, highlighting the purpose and functionality of each script for straightforward implementation in various projects.

All the scripts, links to datasets, pretrained models and more details can be found at: `https://github.com/ferbrjan/Depth_consistency`

8.1 Used Softwares and Libraries

Most of the development work was executed using MATLAB and Python. MATLAB played a main role in generating datasets for both convolutional and pixel-wise approaches, while Python was instrumental in the design, training, and testing of neural networks.

8.1.1 Python Libraries

Below is a list of Python libraries that were utilized:

Library	Purpose
torch	Neural network construction, training, and testing
torch.nn	Building neural network layers
torch.nn.functional	Providing functional interface for neural network operations
torch.optim	Optimization algorithms
scipy	Loading .mat files
matplotlib.pyplot	Plotting and visualization
numpy	Numerical operations
numpy.matlib	Matrix functions in numerical operations
math	Mathematical functions
mat73	Loading MATLAB 7.3 and above version files
os	Operating system interfaces, managing file paths
argparse	Command-line option and argument parsing

Table 8.1: Summary of Python libraries used in the project.

8.2 Dataset Creation

For the dataset creation scripts to function correctly, your working directory should be organized as follows:

```
dataset1
   images.txt
   points3D.txt
   cameras.txt
   images
      00000.jpg
      00001.jpg
      ...
   depth_mvs
      00000.mat
      00001.mat
      ...
   depth_ios
       00000.mat
       00001.mat
       ...
Matlab
   a2h.m
   colmap_conversion_unit_test.m
   ...
generate_data.m
generate_data_exp2_V2.m
...
```

Your work folder, containing the scripts (generate_data.m, generate_data_exp2_V2.m, ...), must include two essential directories: the Matlab folder, which houses all the functions utilized by the main scripts (available on our GitHub repository), and the dataset folder. The dataset folder should encompass the .txt files generated by COLMAP, a directory with all images (preferably numbered from 0 to n), a directory with corresponding depth maps from LiDAR (named identically to the RGB images), and a directory with depth maps from MVS (also named identically to the RGB images).

8.2.1 Pixel-Wise

To initiate the dataset creation process for a pixel-wise approach, begin by opening the generate_data.m script.

You must define the dataset name on line 4 of the script. Additionally, between lines 19 to 23, it's crucial to set the following variables:

Variable	Value
Number of images to reproject (n)	8
Number of intersection points (k)	100
Reference image name (ref_image_name)	"00145.jpg"
Threshold for no outliers (no_outlier_thresh)	0.05
Threshold for outliers (outlier_thresh)	0.2

Table 8.2: Summary of Variables.

After executing generate_data.m, it will produce a file named ref_image_name.mat. The next step is to generate training rays by running generate_training_rays.m and inputting the name of ref_image_name.mat on the first line. Remember to modify the name of the output file on line 77. This step creates the training data necessary for our pixel-wise neural network.

For testing data generation, utilize generate_data.m again to create a new reference image, distinct from the training image. Then, execute generate_test_img.m, ensuring to specify the name of the newly generated .mat file on line 3. This process completes the preparation of testing data.

8.2.2 Convolutional

Data generation for the convolutional approach is somewhat more straightforward compared to the pixel-wise method.

To proceed, you only need to execute the generate_data_exp2_V2.m script. It is essential, as before, to define the dataset name on line 4. Between lines 19 to 23, specify the variables as detailed in 8.2. By default, the value for n is set to 2 to mitigate memory constraints encountered when using GPUs. If your GPU has a higher capacity, you may cautiously increase n.

Executing this script will yield both the training and testing data required for the convolutional model. It's necessary to run the script multiple times to generate multiple sets of data.

8.3 Neural Networks

8.3.1 Pixel-Wise

Training

To run the pixel-wise NN training, load all the necessary data for training (xxx_ready.mat files) and run the pixel_wise.py script. The loading process is not yet optimized as command line argument and therefore it is necesarry to manually input all the files on lines 10-42.

Pretrained Models

Our code repository also includes a pretrained model used during the development of the neural network. The name of the pretrained model foo pixel wise approach is pixel_wise_pretrained.pth and should be loaded for testing.

Testing

To test the neural network just run test.py while specifying the test file on line 29 and path to the pretrained model on line 25. The output should be a .npy file with the resulting means and stds.

Result Analysis

To obtain all the results presented in the 6 section (SSIMs, MSEs, Reprojetion errors...) just simply run analyze_pixelwise.py while specifying the correct path to the generated npy data on line 6.

8.3.2 Convolutional

Training

For CNN training, the script exp2_GPU_downsample_edited_with_output_reprojections.py must be executed. Use the command "python exp2_GP_downsample_edited_with_output_repro
data_path k_path output_path" to run the script, where data_path specifies the location of data created by generate_data_exp2_V2.m, k_path is the location of the camera calibration matrix K, and output_path is where the trained model will be stored.

Moreover, the data used for training need to be organized as follows:

```
data
   Camera_extrinsics
      xxx_RGBD.mat
      ...
   Image_data
       xxx_cam_params.mat
       ...
...
```

The Camera extrinsics and Image data folders can contain multiple corresponding files, if the multi batch training is desired.

Pretrained Models

Our code repository also includes multiple pretrained models used during the development of the neural network. Some of the models are trained on single batch, some are trained on multiple batches. The following overview specifies the specifications of each pretrained model that can be used:

- convolutional_multi_frame.pth - Model trained on multiple batches. Used to evaluate unseen data

- convolutional_single_frame_120.pth - Model trained on single baztch of image with id 120
- convolutional_multi_frame_040.pth - Model trained on single baztch of image with id 40

Testing

To evaluate the network, execute test2_edited.py. Initiate the script with the command "python test2_edited.py data_path model_path", where data_path indicates the location of the test data and model_path points to the location of the pretrained .pth model file. The script will output n .npy files, each containing a depth map produced by the CNN.

Result analysis

To obtain all the results presented in the 6 section (SSIMs, MSEs, Reprojetion errors...) just simply run analyze_results.py while specifying the correct path to the generated data on lines:

- line 101 - path to Lidar depths
- line 116 - 118 - path to generated CNN depth maps
- line 129 - calibration matrix

The script should then return all the necesarry values used as evaluation metrics.

8.4 AI Softwares Used

Some parts of the thesis were rephrased using ChatGPT [39], and then edited again. Also ChatGPT was used to help us generate code templates and for language translation related problems. We did not use ChatGPT for research purposes and no text from our thesis was solely generated by any AI tool.

Another AI software used, was COPILOT [17] by Pycharm. This software proposes next steps while coding in python. Even though it was not used all the time because of its reliability, some of the code propositions were employed, as they were correctly estimated.

Appendix A

Bibliography

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012.

[2] P. F. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In *British Machine Vision Conference*. BMVC, 2013.

[3] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2911–2918. IEEE, 2012.

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009.

[5] A. Baumberg. Reliable feature matching across widely separated views. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2000.

[6] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*. Springer, 2006.

[7] P. R. Beaudet. Rotationally invariant image operators. In *Proceedings of the 4th International Joint Conference on Pattern Recognition*, pages 579–583, Kyoto, Japan, Nov. 1978.

[8] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[9] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6).

[10] H. Chen, Z. Luo, L. Zhou, Y. Tian, M. Zhen, T. Fang, D. Mckinnon, Y. Tsin, and L. Quan. Aspanformer: Detector-free image matching with adaptive span transformer, 2022.

[11] J. Cui, Z. Tian, Z. Zhong, X. Qi, B. Yu, and H. Zhang. Decoupled kullback-leibler divergence loss, 2023.

[12] B. Cyganek. *An Introduction to 3D Computer Vision Techniques and Algorithms*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2007.

[13] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. *CoRR*, abs/1712.07629, 2017.

[14] M. Dusmanu, I. Rocco, T. Pajdla, M. Pollefeys, J. Sivic, A. Torii, and T. Sattler. D2-net: A trainable CNN for joint detection and description of local features. *CoRR*, abs/1905.03561, 2019.

[15] P. Ebel, A. Mishchuk, K. M. Yi, P. Fua, and E. Trulls. Beyond cartesian representations for local descriptors. In *International Conference on Computer Vision*. IEEE, 2019.

[16] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[17] GitHub. Github copilot. AI-powered code completion tool, 2021.

[18] R. Hartley. Projective reconstruction and invariants from multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1036–1041, 1994.

[19] R. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[20] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[21] Y. Jin, D. Mishkin, A. Mishchuk, J. Matas, P. Fua, K. M. Yi, and E. Trulls. Image Matching across Wide Baselines: From Paper to Practice. *International Journal of Computer Vision*, 2020.

[22] O. Kafka. Comparison of methods for matching of images with weakly textured areas, and their usage in 3d image reconstruction, 2021.

[23] M. M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In A. Sheffer and K. Polthier, editors, *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, volume 256 of *SGP '06*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[24] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision*. IEEE, 2011.

[25] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys. Lightglue: Local feature matching at light speed, 2023.

[26] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[27] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[28] Z. Luo, T. Shen, L. Zhou, J. Zhang, Y. Yao, S. Li, T. Fang, and L. Quan. Contextdesc: Local descriptor augmentation with cross-modality context. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[29] Z. Luo, T. Shen, L. Zhou, S. Zhu, R. Zhang, Y. Yao, T. Fang, and L. Quan. Geodesc: Learning local descriptors by integrating geometry constraints. In *European Conference on Computer Vision*. Springer, 2018.

[30] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[31] S. Meerits, V. Nozick, and H. Saito. Real-time scene reconstruction and triangle mesh generation using multiple rgb-d cameras. *Journal of Real-Time Image Processing*, 16(6):2247–2259, 2019.

[32] K. Mikolajczyk, C. Schmid, and A. Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In *European Conference on Computer Vision*. Springer, 2004.

[33] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In *Advances in Neural Information Processing Systems*, 2017.

[34] D. Mishkin, F. Radenovic, and J. Matas. Repeatability is not enough: Learning affine regions via discriminability. In *European Conference on Computer Vision*. Springer, 2018.

[35] J. J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[36] R. Mur-Artal and J. D. Tardos. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct. 2017.

[37] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[38] Y. Ono, E. Trulls, P. Fua, and K. M. Yi. Lf-net: Learning local features from images. *CoRR*, abs/1805.09662, 2018.

[39] OpenAI. Chatgpt. AI language model, 2024.

[40] T. Pajdla. Elements of geometry for computer vision and computer graphics, 2021.

[41] J. Park, J. Cho, S. Lee, S. Bak, and Y. Kim. An automotive lidar performance test method in dynamic driving conditions. *Sensors*, 23(8), 2023.

[42] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019.

[43] J. J. Park, P. R. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103, 2019.

[44] Polycam. Polycam - LiDAR 3D Scanner for iPhone & Android, 2023.

[45] J. Revaud, P. Weinzaepfel, C. R. de Souza, N. Pion, G. Csurka, Y. Cabon, and M. Humenberger. R2D2: repeatable and reliable detector and descriptor. *CoRR*, abs/1906.06195, 2019.

[46] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *International Conference on Computer Vision*. IEEE, 2011.

[47] J. L. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[48] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[49] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2538–2547, 2017.

[50] J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou. Loftr: Detector-free local feature matching with transformers, 2021.

[51] Y. Tian, B. Fan, and F. Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2017.

[52] Y. Tian, X. Yu, B. Fan, F. Wu, H. Heijnen, and V. Balntas. Sosnet: Second order similarity regularization for local descriptor learning. In *Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[53] P. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[54] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. A comprehensive survey of bundle adjustment in computer vision. In *Proc Vision Algorithms: Theory and Practice*, volume 1883 of *LNCS*, pages 298–372. Springer Verlag, 1999.

[55] R. E. Turner. An introduction to transformers, 2024.

[56] M. J. Tyszkiewicz, P. Fua, and E. Trulls. Disk: Learning local features with policy gradient, 2020.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[58] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM International Conference on Multimedia*, pages 1469–1472. ACM, 2010.

[59] C. Wan, T. Probst, L. V. Gool, and A. Yao. Dual grid net: hand mesh vertex regression from single depth maps. *CoRR*, abs/1907.10695, 2019.

[60] F. Wang, S. Galliani, C. Vogel, P. Speciale, and M. Pollefeys. Patchmatchnet: Learned multi-view patchmatch stereo. *CoRR*, abs/2012.01411, 2020.

[61] C. Wöhler. *3D Computer Vision: Efficient Methods and Applications*. X.media.publishing. Springer, London, 2 edition, 2013.

[62] S. Xu, S. Chen, R. Xu, C. Wang, P. Lu, and L. Guo. Local feature matching using deep learning: A survey. *Information Fusion*, 107:102344, July 2024.

[63] P. Zarzycki. *An invitation to 3-D vision. From images to geometric models, by Yi Ma, Stefano Soatto, Jana Košecka and S. Shankar Sastry. Pp. 526. £61.50. 2004. ISBN 0 387 00893 4 (Springer Verlag).*, volume 89. 2005.

[64] B. Zhang, S. Feng, X. Li, Y. Ye, R. Ye, C. Luo, and H. Jiang. Sgmnet: Scene graph matching network for few-shot remote sensing scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–15, 2022.

[65] X. Zhao, X. Wu, W. Chen, P. C. Y. Chen, Q. Xu, and Z. Li. Aliked: A lighter keypoint and descriptor extraction network via deformable transformation, 2023.