UTILIZATION OF GAUSSIAN SPLATTING IN VISUAL SLAM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FURKAN AYKUT SARIKAMIŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2025

Approval of the thesis:

UTILIZATION OF GAUSSIAN SPLATTING IN VISUAL SLAM

submitted by **FURKAN AYKUT SARIKAMIŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Naci Emre Altun
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. A. Aydın Alatan
Supervisor, **Electrical - Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU _____

Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Ramazan Gökberk Cinbiş
Computer Engineering, METU _____

Prof. Dr. Orhan Arıkan
Electrical and Electronics Engineering, Bilkent University _____

Date:07.01.2025

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Furkan Aykut Sarıkamış

Signature :

ABSTRACT

UTILIZATION OF GAUSSIAN SPLATTING IN VISUAL SLAM

Sarıkamış, Furkan Aykut
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. A. Aydın Alatan

January 2025, 69 pages

3D Gaussian Splatting has emerged as a promising alternative to neural implicit representations in SLAM systems. However, current methods often lack dense depth maps or tailored designs for large-scale environments. To address these issues, this thesis introduces IG-SLAM, an RGB-only SLAM system that combines robust tracking methods with Gaussian Splatting. The system builds a 3D map using accurate poses and dense depth from tracking while leveraging depth uncertainty for improved reconstruction. Our decay strategy enhances convergence and enables real-time operation at 10 fps in a single process. IG-SLAM achieves competitive results with state-of-the-art RGB-only SLAM systems at significantly faster speeds, demonstrating photo-realistic 3D reconstruction on several datasets. The project is available at https://github.com/Liouvi/IG-SLAM.

Keywords: Gaussian Splatting, SLAM, NeRF

ÖZ

GÖRSEL EŞ ZAMANLI KONUMLANDIRMA VE HARİTALAMADA GAUSS SIÇRAMASININ KULLANIMI

Sarıkamış, Furkan Aykut
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. A. Aydın Alatan

Ocak 2025 , 69 sayfa

3D Gaussian Splatting, SLAM sistemlerinde sinirsel örtük temsilcilere karşı umut verici bir alternatif olarak ortaya çıkmıştır. Ancak mevcut yöntemler genellikle yoğun derinlik haritalarından veya büyük ölçekli çevreler için özel tasarımlardan yoksundur. Bu sorunları çözmek için, bu tez, güçlü izleme yöntemlerini Gaussian Splatting ile birleştiren, yalnızca RGB kullanan bir SLAM sistemi olan IG-SLAM'ı tanıtmaktadır. Sistem, izlemeyi kullanarak doğru pozlar ve yoğun derinlik ile bir 3D harita oluştururken, derinlik belirsizliğini iyileştirilmiş rekonstrüksiyon için kullanır. Öğrenme hızını yavaşlatma stratejimiz, yakınsama hızını artırır ve sistemi tek bir işlemde 10 fps hızında gerçek zamanlı çalıştırmayı mümkün kılar. IG-SLAM, en son teknoloji RGB-SLAM sistemleriyle rekabetçi sonuçlar elde ederken, çok daha hızlı çalışır. Birçok veri setinde foto-gerçekçi 3D rekonstrüksiyon sergiler. Proje, https://github.com/Liouvi/IG-SLAM adresinde mevcuttur.

Anahtar Kelimeler: Gaussian Splatting, SLAM, NeRF

To my family, for their support and love.

To my friends, who cheered me on through every challenge.

To my mentor, whose guidance shaped this journey.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to all those who have supported me throughout my research.

First and foremost, I am sincerely thankful to my supervisor, Prof. Dr. A. Aydın Alatan, for his continuous guidance, encouragement, and invaluable insights. His mentorship has been essential in shaping my research.

I am deeply grateful to Yunus Bilge Kurt, and Haktan Yalçın for their assistance, constructive discussions, and encouragement.

I would also like to express my sincere thanks to Önder Tuzcuoğlu for the many insightful discussions and enjoyable walks we shared.

Thanks to Hande Çığla and Bahar Şengün for their delightful conversations and for always offering Turkish coffee whenever they see me.

Lastly, I wish to thank the authors and researchers whose work laid the foundation for Gaussian Splatting. It is truly a work of art.

Thank you all for being part of this journey.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a core and fundamental technique in the fields of robotics, augmented reality (AR), virtual reality (VR), and autonomous vehicles. It enables an agent to construct a map of an unknown environment while simultaneously localizing itself within that map. Despite its critical role in these domains, achieving real-time performance with high-fidelity 3D reconstructions remains a significant challenge, particularly in dynamic and unstructured environments.

Traditional SLAM [1, 2, 3] systems have primarily relied on sparse representations of the environment, which, while computationally efficient, often fail to capture the visual richness required for photorealistic 3D reconstruction. Recent advances in neural rendering [4, 5, 6, 7] have introduced new possibilities for high-quality scene reconstruction, but their application in real-time SLAM systems [8, 9] has been limited due to the substantial computational resources required.

This thesis introduces IG-SLAM [10], a novel SLAM framework that integrates Gaussian Splatting, a state-of-the-art neural rendering technique, into the SLAM pipeline. The goal is to enhance the quality and photorealism of 3D scene reconstructions while maintaining real-time performance. By adopting Gaussian Splatting, IG-SLAM [10] offers a continuous and dense representation of the environment, enabling the capture of fine details that are often missed by traditional SLAM methods.

The key contributions of this thesis are threefold. First, we design and implement a novel pipeline that integrates Gaussian Splatting into SLAM, providing a unified system capable of handling both static and dynamic scenes. Second, we develop op-

timization techniques to balance computational efficiency and reconstruction quality, ensuring that real-time performance is preserved. Third, we demonstrate the system's robustness through extensive experiments on diverse datasets, showcasing its ability to operate effectively in challenging real-world environments.

IG-SLAM [10] represents a shift from traditional sparse representations to a dense and continuous mapping approach, bridging the gap between classical SLAM techniques and modern neural rendering paradigms. In this manner, this system opens new possibilities for high-fidelity 3D mapping in real-time applications, which is crucial for interactive AR/VR experiences, robotics, and autonomous navigation.

Since the focus of this thesis is the utilization of neural rendering methods in SLAM, the work initially focused on Neural Radiance Fields (NeRF) [4], an approach that integrates neural networks into the photorealistic reconstruction pipeline. However, with the advent of Gaussian Splatting [5], which provides significant advantages in terms of computational efficiency and flexibility, the focus of our research shifted to this method. Nevertheless, to provide a comprehensive perspective and preserve the context of our early investigations, This thesis includes a discussion of NeRF in the Appendix. This ensures a complete documentation of our research process and highlights the transition from one state-of-the-art technique to another.

The remainder of this thesis is structured as follows. Chapter 2 provides a review of related work in SLAM literature. Chapter 3 introduces the Gaussian Splatting Algorithm [5]. Chapter 4 explains how Gaussian Splatting can be integrated into SLAM, outlining its key concepts and methodologies. Chapter 5 focuses on the IG-SLAM framework, detailing its design, implementation, and components. Chapter 6 presents the experimental evaluation, showcasing both quantitative and qualitative results that demonstrate the performance and effectiveness of IG-SLAM. Chapter 7 concludes the thesis, summarizing the key findings and contributions. Finally, the Appendices include an explanation of NeRF and its application in SLAM algorithms.

CHAPTER 2

SIMULTANEOUS LOCALIZATION AND MAPPING

2.1 Introduction

The Simultaneous Localization and Mapping (SLAM) problem involves creating a map of an unknown environment while simultaneously estimating the position of a robot within that environment. As part of this problem, no prior knowledge of the environment is given. The robot in the SLAM problem can be as simple as a calibrated camera, a calibrated camera paired with an inertial measurement unit (IMU), or any other sensor with the possibility of sensing landmarks in the environment.

In this thesis, the main focus is on visual SLAM (VSLAM). Therefore, a body or robot means only a calibrated camera. Visual landmarks detected by a simple camera might be edges, corners, blobs, ridges, features, the whole image itself, etc. The map is classically built incrementally by the 3D points associated with these matched 2D features detected by the camera.

Although map optimization methods can vary and be diverse, one can divide the visual tracking task into two categories: Indirect and Direct. The indirect approaches [2, 3, 11, 1] utilize sparse 3D points during optimization, such as corners, and the direct approaches [12, 13, 14] *directly* make use of the measured intensities of the image. Since the agent is a camera in VSLAM, it is convenient to review camera basics first.

2.2 Camera Basics

In VSLAM, the camera is utilized to detect visual features. A simple pinhole camera projection as a function can be understood as

$$\Pi \colon \mathbb{R}^4 \to \mathbb{R}^3$$
$$\mathbf{X_w} \mapsto \mathbf{x}.$$

where Π is the projection function, $\mathbf{X_w} = [X, Y, Z, 1]$ is the homogeneous coordinate of a 3D point with respect to a fixed frame, called World Frame, $\mathbf{x} = [x, y, 1]$ is the 2D location of the point on the camera frame. To find the projection function Π, the first step is to express the 3D point w.r.t. camera coordinates. The coordinate transformation $\mathbf{T_c^w}$ is defined as

$$\mathbf{T_c^w} \colon \mathbb{R}^4 \to \mathbb{R}^4$$
$$\mathbf{X_w} \mapsto \mathbf{X_c}.$$

When the points are expressed as homogeneous coordinates, the coordinate transformation becomes linear as $\mathbf{X_c} = \mathbf{T_c^w X_w}$ where $\mathbf{T_c^w} \in \mathbb{R}^{4x4}$ becomes

$$\mathbf{T_c^w} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

where $\mathbf{R} \in \mathbb{R}^{3x3}$, $\mathbf{t} \in \mathbb{R}^3$ is the rotation matrix and translation vector between the frames, and $\mathbf{0} = [0, 0, 0]$. Note that this transformation is simply a linear and compact form of the coordinate transformation in $\mathbb{R}^3$ i.e. $\mathbf{X'} = \mathbf{RX} + \mathbf{t}$. Once the point is expressed in the camera coordinate frame, the projection of it can be modeled as

$$\mathbf{P} \colon \mathbb{R}^4 \to \mathbb{R}^3$$
$$\mathbf{X_c} \mapsto \mathbf{x}.$$

where $\mathbf{x} = (x, y, 1)$ and $(x, y, 1) = (fX/Z, fY/Z, 1)$. The projection $\mathbf{P}$ can also be represented as a linear transformation

$$\mathbf{P} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Therefore, the projection operation that projects a 3D point expressed in world coordinates to a camera frame is simply given as in Equation (2.1)

$$\Pi = \mathbf{P}\mathbf{T}_{\mathbf{c}}^{\mathbf{w}} \tag{2.1}$$

2.3 Problem Definition

SLAM tackles the problem of acquiring an accurate trajectory and a consistent map at the same time. Assume that while the camera follows a certain trajectory as in Figure 2.1, some 3D features $\mathbf{F_j}$ of the camera at different locations $\mathbf{X_i}$ are covisible. The goal is to achieve an accurate trajectory and **global consistency** in the mapping. Some auxiliary definitions are needed to define global consistency.

3D features w.r.t. world coordinate designated with capital letters as $\mathbf{F}$. The 2D feature location spotted by the camera is given by $\mathbf{f}$. The number of 2D features is designated as m_i. The projection function that projects a 3D point to the image plane of the camera at ith position is given as Π_i. The global consistency is achieved by minimizing geometric displacement error for all features observed in all cameras. Therefore, the total error can be given as in Equation (2.2).

$$E = \sum_{i} \sum_{j=1}^{m_i} ||f_j - \Pi_i(F_j)|| \tag{2.2}$$

where $||.||$ is an appropriate norm which is usually selected as L_2.

One important distinction in this formulation is the nature of the features. If the features are sparse corners, blobs, edges, etc, the method is denoted as indirect SLAM;

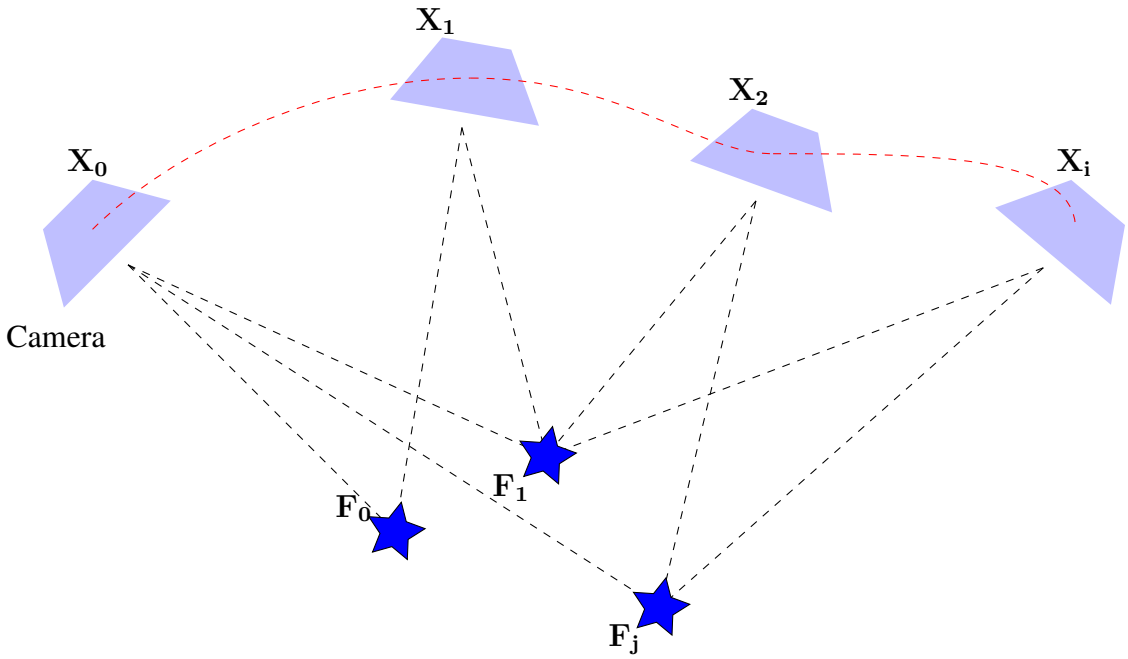

Figure 2.1: A typical SLAM scenario, where red dashed line corresponds to the camera trajectory, F_j's are 3D features and X_i's are the camera positions.

whereas the direct methods utilize the intensities of the dense image pixels. Most of the direct methods compute intensity difference based *photometric loss* instead of the projection error defined in Equation (2.2).

Since the objective is now clearly defined, the tools to be used in SLAM algorithms can be discussed. The sub-blocks that are utilized in VSLAM are mainly visual odometry, loop closure, global, and local bundle adjustment.

2.4 Sub-blocks of a VSLAM System

The building blocks of VSLAM can be constructed using different strategies. However, a classical VSLAM algorithm is constituted by *visual odometry* whose concern is to track the agent by consecutive frames, *loop closure* whose main concern is to detect loops in the trajectory of the agent, and finally, *local* and *global bundle adjustment* to increase local and global consistency.

2.4.1 General Visual Odometry

The visual odometry algorithm tackles the problem of computing relative rotation and translation between two consecutive frames and concatenates these rotation and translation information to get the overall trajectory of the agent. As the simple concatenation suggests, the visual odometry drifts away from the ground truth trajectory. Therefore, an optimization may be applied to reduce this drift.

The first visual odometry algorithm can be examined in [15] which also coined the term visual odometry. A general visual odometry algorithm is summarized in Figure 2.2.

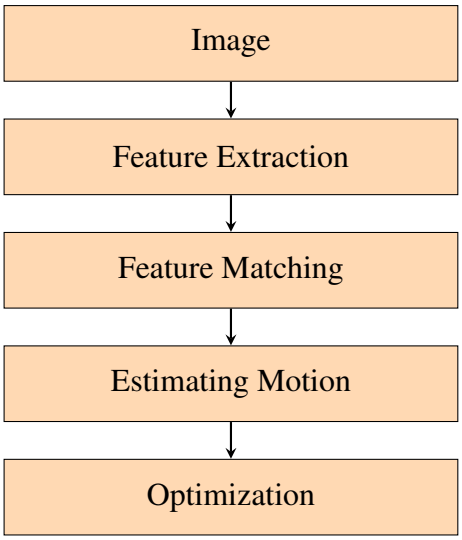

Figure 2.2: The Visual Odometry Algorithm

The feature extraction step is achieved by searching salient points on the image. The salient points might be corners and blobs for visual odometry. A corner is defined as the intersection of two edges and a blob is defined as a point that differs in intensity from its neighbors. Popular corner detectors are Harris [16], Shi-Thomasi [17], Forstner [18], and FAST [19], and blob detectors are SIFT [20], SURF [21], and CENSURE [22].

Once the features are extracted via the aforementioned methods, they are to be matched between the consecutive frames. Feature matching can be performed by comparison of the descriptors or by tracking the individual points in its vicinity as in the KLT

tracker [23]. While the details of feature selection and matching techniques are different in this area of research, it can generally be stated that blob detectors are slow to detect compared to corner detectors. However, blob detectors are more robust, distinctive, and descriptive.

Sparse feature matching with descriptors is more appropriate when the rotation and translation between two frames are not small; i.e. wide-baseline. Otherwise, KLT-based tracking is more suitable for narrow-baseline scenarios. The matched pairs in two frames are called feature correspondences. As examples from the literature, VINS-Mono utilizes Shi-Thomasi [17], ORB-SLAM3[3] exploits ORB [24], and OpenVINS [1] utilizes FAST [19] as feature detection and matching.

After feature matching, the motion is estimated with the extracted feature correspondences. The estimation can be done with 2D-2D or 2D-3D feature correspondences. The 2D correspondences are obtained by matching them in consecutive frames. The 3D correspondences are the ones that are triangulated to get a 3D position in the space.

The initial three legitimate frames to be matched are handle in a special way in SLAM algorithms. Since there are no 3D points available, it is necessary to perform a 2D-2D match to estimate the relative motion and estimate 3D point locations via triangulation.

The rotation and translation relation between the 2D correspondence coordinates at two frames is formulated by the Essential Matrix $\mathbf{E}$ such that $\mathbf{E} = \mathbf{tR}$ where $\mathbf{t}$ is the skew-symmetric matrix form of the up to a scale translation:

$$\mathbf{t} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & f & -t_x \\ -t_y & t_x & 1 \end{bmatrix}$$

and $\mathbf{R}$ is the rotation matrix with orthonormal rows/columns. The scale ambiguity comes from a simple observation that Equation (2.3) is the true multiplication of $\mathbf{E}$ with any scalar. Therefore, it is impossible to determine the exact translation between two frames.

One of the most important properties of the essential matrix $\mathbf{E}$ is called epipolar constraint. The epipolar constraint simply constrains the position of a feature in a frame on a line, called the epipolar line, depending on the position of the correspondence of it in another frame. The constraint can be stated in a simple expression as in Equation (2.3) and [25]

$$\mathbf{p'^T E p} = \mathbf{0} \tag{2.3}$$

where $\mathbf{p} = [u, v, 1]$, u, v are the pixel locations of the feature, and $\mathbf{p'}$ is the one for the correspondence. To compute the essential matrix $\mathbf{E}$, several algorithms exist. Two of the most popular ones are the 8-point algorithm [26] and the 5-point algorithm [27].

The 8-point algorithm expands the equation for 8 correspondence and solves for coefficients of $\mathbf{E}$ and the 5-point additionally utilizes a constraint of $\mathbf{E}$ to decrease the required correspondence to 5. The procedure to solve for the essential matrix is quite standard. The algorithms are not used to find definitive results for $\mathbf{E}$ but rather to generate a hypothesis for RANSAC [28].

Since there are generally more than 5 or 8 pairs, random pairs are selected for the minimal solution, the result, namely *hypothesis*, is tested at other points and if the majority of points meet the constraint of $\mathbf{E}$, the result is accepted. Otherwise, the procedure is followed one more time again with random samples. After solving for $\mathbf{E}$, it is decomposed to find $\mathbf{R}$ and $\mathbf{t}$. At this point, the rotation and up to a scale translation between two frames are obtained.

Since the 2D correspondences and the 3D position of the frames are known, the 2D correspondences can be triangulated to estimate their 3D positions. The triangulation algorithm starts with unprojecting rays from the correspondences. These rays ideally should intersect and the intersection point would be designated as a 3D point as depicted in Figure 2.3.

However, due to matching errors and sensor noise, this ideal intersection is rarely the case. Therefore, the minimum distance between two rays is obtained and the middle point between these two closest points on the rays is designated as an estimated 3D point position. The resulting 3D point set, i.e. 3D point-cloud, is essentially one of

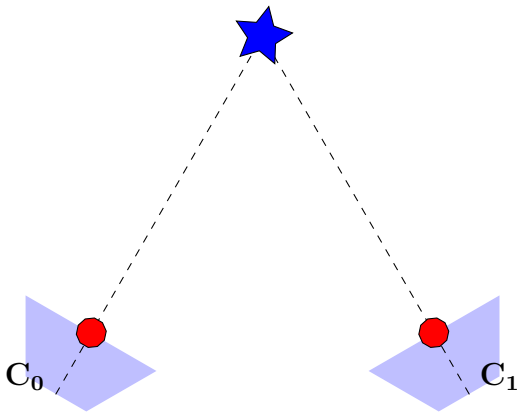

Figure 2.3: Given two frames, $\mathbf{I_0}$ and $\mathbf{I_1}$, from two camera positions, triangulation algorithm estimates rhe 3D location of the corresponding scene point

the most popular ways to describe the environment, and the general process to obtain them is called **mapping** in SLAM literature.

Intuitively, the 2D-2D match and triangulation steps can be repeated to calculate the trajectory of the agent and the 3D point cloud of the environment. However, estimating motion from a 2D-2D match is quite noise-prone; therefore, it is less preferred; instead, 3D-2D matching is utilized. In 3D-2D matching, 2D-2D matching is performed as usual; the matches that have 3D points are utilized in a Perspective-n-Point algorithm [29].

In general, the Perspective-n-Point problem is to find the camera position given 3D points and their corresponding 2D points. The minimal case is called the Perspective-3-Point problem and its solution has been known for centuries. The solution is to get the camera position from trigonometric similar triangle relations that are created by 3D points, 2D points on the camera with physical 3D coordinates, and the camera center. This minimal solution can be similarly utilized as a hypothesis in RANSAC to get the solution that has a sufficient inlier ratio. After finding the camera position, 2D-2D matches that do not have a 3D correspondence triangulated. The procedure is summarized in Figure 2.4

Therefore, the full algorithm is to 2D-2D match once to get the initial 3D points and then proceed to 3D-2D matching algorithm repeatedly to get the trajectory of the agent. While visual odometry and SLAM are quite similar concepts, their philosophy

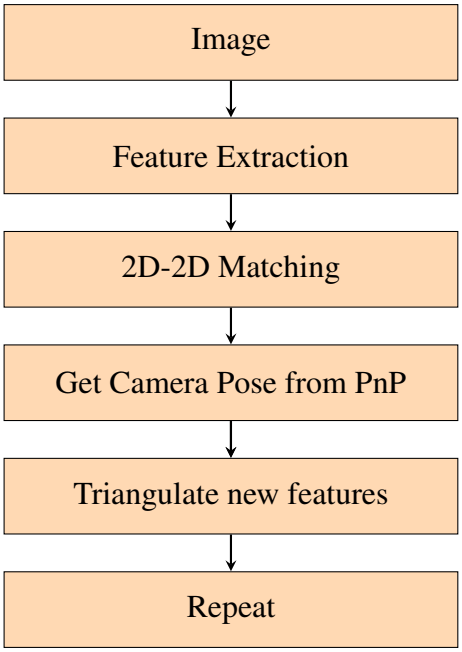

Figure 2.4: 3D-2D Matching Algorithm Block Diagram

is entirely different. Visual Odometry's only concern is the camera position; the resulting 3D points do not have to be globally consistent. However, in SLAM, that 3D scene point has to be recognized and be at the same location on the map every time the agent crosses the same location. To achieve this recognition of the same location loop closure algorithms are utilized on top of visual odometry.

2.4.2 Loop Closure

The main aim of a loop closure algorithm is to recognize the same scene location when the agent arrives at that location again and update its location. The update of loop closure on this location is the only way to get better location information coming from the visual odometry itself, since the visual odometry calculates the position of the agent by concatenating the relative poses and it is prone to drifting.

While the loop closure algorithms slightly differ, their main idea is usually the same. For example, a simple but efficient algorithm candidate might be the one in the VINS-Mono algrorithm[2]. The keyframes are selected in equal time intervals and, the Harris corner features are extracted in these keyframes with their BRIEF descriptors

[30]. The descriptor matching is performed using RANSAC [28]. Additionally, a PnP test is achieved between 2D points and 3D points. If the aforementioned two tests are successful, then the current frame's position is taken constant as the matched previous keyframe, and reprojection error is minimized with this new constraint. The result essentially brings the drifted position of the same location to the previous less drifted one. In order to further increase global and local consistency, global and local bundle adjustments might also be exploited.

2.4.3 Global and Local Bundle Adjustment

All 2D points in each frame and their corresponding 3D point coordinates constrain the pose of the agent in that frame of the camera. Hence, using these constraints, the drift effect of the visual odometry can be reduced by global and local bundle adjustment.

In this optimization, the accumulated 3D points are projected onto the frames, and reprojection error is calculated by Equation (2.2). Since the projection matrix Π consists of rotation, this optimization is a nonlinear optimization and the reprojection error is generally minimized by Levenberg-Marquardt optimization [31].

Performing this optimization on every frame is denoted as global bundle adjustment. Due to its computational complexity, it is not generally possible to achieve global bundle adjustment in real time. Hence, the approach is to perform the bundle adjustment in the last several frames, called local bundle adjustment, and apply a global adjustment less frequently.

The optimized poses of the agent and the map of the environment made up of 3D points are the outputs of a classical SLAM system. However, these 3D points are not necessarily the only way to represent the environment. The 3D points represent the environment in the sense that there is some visual information at that 3D point that is salient. However, only if the point cloud is dense, one can get a sense of the local topology of that particular location. Otherwise, the sparse point cloud fails to represent the environment photo-realistically.

2.5 Related Work

Pioneering dense SLAM algorithms, such as Dense Tracking and Mapping (DTAM) [12] and KinectFusion [32], demonstrated the feasibility of real-time dense SLAM despite its inherent computational complexity. DTAM [12] focuses on generating dense depth maps associated with the keyframes, following a view-centric approach. In this method, depth maps are computed and associated with specific viewpoints, which allows for detailed scene representation. However, this approach involves certain computational limitations when extended to larger-scale or more dynamic environments.

Subsequent research took inspiration from DTAM's view-centric model but introduced crucial distinctions. While traditional dense SLAM methods typically decouple the optimization processes for dense depth maps and camera poses, some recent works moved toward joint optimization, aiming to optimize both simultaneously for better consistency and accuracy. Such joint optimization methods, however, face significant challenges, since the optimization of full-resolution depth maps is particularly expensive in terms of computation. The high number of independent variables involved in optimizing dense maps makes it infeasible to handle full-resolution depth information in real-time applications directly.

To address this, quite recent data-driven methods have focused on reducing the computational complexity of joint optimization in dense SLAM. For instance, BA-Net [33] integrates a depth map into the bundle adjustment process by utilizing a basis of depth maps and optimizing the linear combination coefficients. This approach significantly reduces the computational burden by working with a smaller set of parameters while still maintaining a high degree of accuracy. Similarly, Code-SLAM [34] introduces an autoencoder-inspired architecture to reduce the dimensionality of dense maps. This method compresses depth maps into a more manageable form while retaining the essential features needed for effective SLAM.

DROID-SLAM [13], another noteworthy advancement, optimizes down-sampled dense maps using a bundle adjustment layer with reprojection error and incorporates optical flow revisions from [35] to refine the results. DROID-SLAM method provides an

efficient way of handling large-scale dense maps by working with simplified, lower-resolution representations, which reduces computational complexity while maintaining good accuracy. This work relies on DROID-SLAM [13] as its tracking module. Its components and design are explained in detail in Section 5.1

Further innovation can be found in FlowMap [36], which estimates dense depth maps through a convolutional neural network (CNN) and calculates the camera pose analytically using the optical flow. This approach introduces a neural network-based method for estimating depth maps, offering improved accuracy for real-time applications.

3D scene representation can be extended to a model that enables rendering from any point in the environment from any direction. By the help of extending the notion of representing the environment, novel view synthesis methods can be utilized in a SLAM system, namely *Neural Radiance Fields* [4] and *Gaussian Splatting* [5]. This thesis proposed a novel method for the utilization of the Gaussian Splatting [5] technique in a SLAM system.

CHAPTER 3

GAUSSIAN SPLATTING

Gaussian Splatting is a novel view synthesis method for 3D scene representation that utilizes Gaussian functions to model and reconstruct environments. This approach stands out by offering a more flexible and computationally efficient alternative to traditional 3D reconstruction methods, such as voxel grids and meshes. In this technique, the scene is represented as a collection of 3D Gaussian functions, where each Gaussian encodes information about the scene's geometry and appearance. This continuous modeling allows for dense scene representations which can be efficiently rendered and manipulated.

3.1 Theory

A Gaussian function is described uniquely by its mean and covariance as

$$G(x) = e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{3.1}$$

where x is the independent vector variable of the Gaussian function, Σ is its covariance matrix, and μ is the mean vector. The Gaussian function in Equation (3.1) is centered at its mean value μ and the mean physically represents the position of the Gaussian, whereas its shape is determined by the covariance matrix. In the remaining formulations, x denotes the 3D coordinates of a scene point.

The $alpha$-blending of Gaussians requires the projection of 3D Gaussians. The mean value μ can be directly projected since it's a 3D point. For the covariance, the projected 2D covariance is given in Equation (3.2). The derivation can be found in [37].

$$\Sigma' = JW\Sigma W^T J^T \tag{3.2}$$

In the relation above, W is the viewing transformation, and J is the Jacobian of the affine approximation of the projection transformation.

However, the covariance matrix itself has a positive definiteness constraint which is difficult to enforce during optimization. Instead, representing the covariance matrix as a combination of 3D scaling $S = \{S_x, S_y, S_z\}$ and 3D rotation R is much more intuitive and easy to optimize. Therefore, the covariance matrix in 3D is decomposed as follows:

$$\Sigma = RSS^T R^T \tag{3.3}$$

The rotation R is represented as quaternion and scaling is represented as 3 independent parameters. The Gaussians' color is encoded as spherical harmonics and the coefficients of Spherical Harmonics are optimized. This approach resembles the positional encoding as in Equation (A.10) of the color in NeRF (please see Appendix) and is performed to represent the color by not only 3 numbers as in RGB but in a higher dimensional space to increase color accuracy. The main algorithm to optimize position μ, α, covariance Σ, and, the spherical harmonic coefficient for color is shown in Figure 3.1

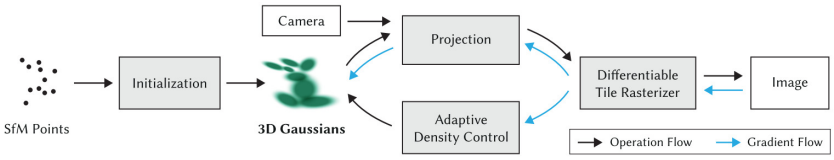

Figure 3.1: The building blocks of the Gaussian Splatting algorithm adapted from [5]. The initial position of the Gaussians is determined by any conventional Structure-from-Motion (SfM) points. 3D Gaussians are projected into 2D image plane and rasterized with α-blending. The error is calculated between the rendered image and the ground truth image.

The algorithm starts with the initialization of Gaussian functions at sparse 3D points.

The mean vectors of the 3D Gaussians are initialized by the estimated SfM points. In the original article, these sparse SfM points are obtained by the popular structure from motion method COLMAP [38]. The covariance matrices are initialized to be a diagonal matrix of the mean distance of the closest three SfM points. Following initialization, the optimization process begins with rasterization and proceeds with backpropagation.

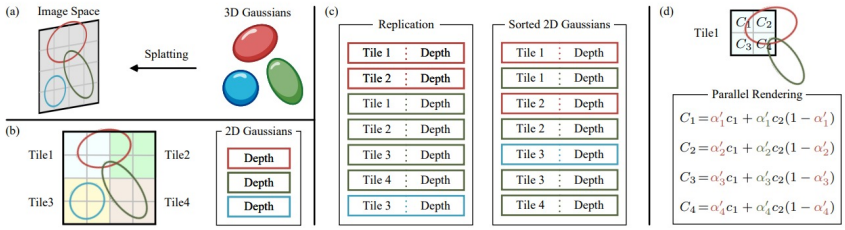

Figure 3.2: An illustration of the Forward Process in Gaussian Splatting. The figure is adapted from [39].

An illustration of the rasterization process is given in Figure 3.2. (a) The splatting step involves projecting 3D Gaussians into the image space. (b) The image is divided into multiple tiles. (c) Gaussians that span multiple tiles are replicated, with each copy assigned a unique identifier (d) Rendering the sorted Gaussians allows for computing all pixels within each tile. It is important to note that the computational workflows for both pixels and tiles are independent, enabling parallel processing.

While the sorting approach might lead to inaccurate ordering for some pixels, the authors state that this does not deteriorate the performance immensely because the size of Gaussians becomes as small as a pixel at the end of the optimization. Finally, the ordered Gaussians are α-blended in parallel. The α-blending starts from the front to the back. The blending only stops when α is equal to 1 or all the Gaussians are rendered. Then, the L1 loss between the rendered and ground truth image is calculated.

The loss is backpropagated through the fully differentiable rendering pipeline. The mean, rotation, scaling, color, and opacity of the Gaussians are optimized to represent the scene as illustrated in Figure 3.3. For example, thin regions are represented by elongated Gaussians, transparent regions are represented by Gaussians with $\alpha \approx 0$.

During the optimization of the Gaussians, they are densified to fill the empty areas at

Figure 3.3: An example of Gaussian Splatting Representation. The rasterized Gaussians (left) and the Gaussians shaded to emphasize geometry (right) are shown. The figure is adapted from [40]

every $N = 100$ iteration. The transparent Gaussians are eliminated at this point. The densification is achieved at the regions where the average positional gradient is high. The reason behind this strategy is the intuitive observation that the positional gradient should be high due to the fact that the region is not represented adequately and the optimization tries to move Gaussians aggressively to make them right.

Densification of small Gaussians is simply achieved by cloning and sending them towards the gradient direction, whereas densification of large Gaussians is performed by dividing them into two by a scale factor. The new Gaussians are sampled using the large Gaussian as PDF.

A specific problem, known as *floaters* in both NeRF and Gaussian Splatting, causes defects to occur close to the camera location. This problem is avoided by setting α of the Gaussians close to zero at every $M = 3000$ iteration. The optimization increases α where it is required. Therefore, the floaters are removed, since their α parameter stays low and, as mentioned, transparent Gaussians are removed at every N iteration.

3.2 Related Work on Gauss Splatting

To improve the consistency and reconstruction quality of Gaussian Splatting, several enhancements have been proposed. One such improvement includes leveraging its 2D counterpart [41], which is designed to enhance multi-view consistency across viewpoints, ensuring that the representation remains coherent even when observed

from different perspectives.

Another challenge in Gaussian Splatting arises from the use of α-blending for depth rendering, as employed in the original formulation of 3D Gaussian Splatting. This approach can lead to noisy and inconsistent surfaces due to ambiguities in the depth estimation. More rigorous methods have been proposed to address this, which introduce viewpoint-dependent depth variations for each Gaussian. These methods dynamically adapt the depth of individual Gaussians based on the observer's viewpoint, thereby significantly reducing noise and improving surface quality [42, 43].

Due to its fast rendering capabilities and its explicit scene representation, Gaussian Splatting has rapidly gained traction in the SLAM literature. Unlike NeRF [4] (please examine Appendix), which rely on implicit representations and are computationally intensive, Gaussian Splatting offers a real-time alternative suitable for SLAM applications.

CHAPTER 4

GAUSSIAN SPLATTING SLAM

The utilization of the Gaussian Splatting paradigm in a SLAM system can be performed in two ways. The first approach is to invert Gaussian Splatting for tracking and use it for mapping. The second approach is to utilize classical SLAM algorithms for tracking and use Gaussian splatting for only mapping. In Section 4.1, the methodology of the first approach is examined.

4.1 Inverting Gaussian Splatting

In Gaussian Splatting [5], 3D Gaussians have a universal position and color. These Gaussians are projected onto the image plane. Since the projection operation is composed of transformation into the camera frame and then projection onto the image plane, the projection depends on the pose of the camera and is differentiable w.r.t. the camera pose. Therefore, an analysis by synthesis method is possible.

The key idea is to extend backpropagation up to the pose itself. Instead of optimizing the position and shape of the Gaussians, the camera pose is optimized. In other words, the main question of the optimization becomes "Where should the camera be located to render an image that will be as similar as possible to the ground truth camera frame?".

The 3D transformation representation $SE(3)$ or homogeneous coordinate formulation in 4x4 matrices has 16 parameters. The last row of the matrix is constant. Therefore, we have 12 meaningful parameters. This is more than sufficient since 6 parameters (3 for translation and 3 for rotation) are required to describe a 3D motion. Therefore,

the most straightforward solution where we calculate the Jacobian of the mean and the variance of the Gaussians is not optimal.

MonoGS [40] is the pioneering work that takes the derivative of Gaussians' position (mean vector) and shape (covariance matrix) w.r.t the camera pose. Assume that the mean and covariance of an arbitrary Gaussian $\mathcal{G}$ in world coordinates is given as μ_W and $\mathbf{\Sigma}_W$. Therefore, the projection of mean and covariance onto the image frame μ_I and $\mathbf{\Sigma}_I$ is given as in Equation (4.1)

$$\mu_I = \Pi(T_{CW} \cdot \mu_W), \quad \Sigma_I = JW\Sigma_W W^T J^T \tag{4.1}$$

In Equation (4.1), the second equation is the same as Equation (3.2) and the first equation is just a simple projection operation of a scene point in world coordinates to camera coordinates. The gradient that needs to be calculated is shown in Figure 4.1.

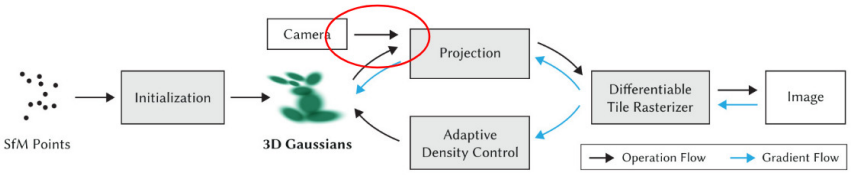

Figure 4.1: The original building blocks of the Gaussian Splatting algorithm adapted from [5]. The gradient that needs to be calculated is shown in the red circle. This gradient is, of course, nonexistent in the original Gaussian Splatting since the algorithm takes posed images as input. This gradient is essentially the gradient of projected 2D Gaussians' position and covariance w.r.t camera pose.

The mean and covariance gradient with respect to camera pose i.e. $\frac{\partial \mu_I}{\partial T_{CW}}$ and $\frac{\partial \Sigma_I}{\partial T_{CW}}$ highlighted in Figure 4.1 can be given as in Equation (4.2) and Equation (4.3) by partial derivation.

$$\frac{\partial \mu_I}{\partial T_{CW}} = \frac{\partial \mu_I}{\partial \mu_C} \frac{\mathcal{D}\mu_C}{\mathcal{D}T_{CW}} \tag{4.2}$$

$$\frac{\partial \Sigma_I}{\partial T_{CW}} = \frac{\partial \Sigma_I}{\partial J} \frac{\partial J}{\partial \mu_C} \frac{\mathcal{D}\mu_C}{\mathcal{D}T_{CW}} + \frac{\partial \Sigma_I}{\partial W} \frac{\mathcal{D}W}{\mathcal{D}T_{CW}} \tag{4.3}$$

where $\frac{\partial}{\partial(.)}$ is partial derivative and $\frac{\mathcal{D}}{\mathcal{D}(.)}$ is derivative on the manifold. The difference is that ordinary partial derivative $\frac{\partial \mu_C}{\partial T_{CW}}$ instead of $\frac{\mathcal{D}\mu_C}{\mathcal{D}T_{CW}}$ would mean standard Jacobian 3x12 Jacobian of the camera coordinate mean w.r.t 12 parameters of T_{CW}. However, this is not the optimal way of calculating the derivatives since 6 parameters are sufficient for describing a pose. Therefore, Lie algebra should be utilized. The details of the Jacobian derivation can be found in the Appendices.

As it can be observed in Figure 4.1, the gradient already flows from the error to the first rendered image and then to 2D Gaussians. Therefore concatenating the gradient in Equation (A.2) to the existing ones yields the full gradient up to the camera pose.

The main algorithm of MonoGS [40] is quite similar to iMAP [9] given in the Appendix. The tracking is achieved by optimizing the pose while fixing the Gaussians. On the mapping side, keyframes are used to train the Gaussian Splatting. To prevent the network from catastrophic forgetting, two images from the previous keyframes are also used. The authors [40] state that the elongation of Gaussians along the viewing direction was a problem in their system; hence, a cost function that punishes non-spherical shapes is added to the cost as well.

4.2 Related Work on using Gaussian Splatting for SLAM

Several pioneering works have explored the use of Gaussian Splatting in SLAM systems. MonoGS [40], GS-SLAM [44], and SplaTAM [45] are among the earliest SLAM algorithms that exclusively utilize Gaussian Splatting for scene representation. These systems jointly optimize both the 3D Gaussians and camera poses, ensuring a cohesive integration of mapping and localization.

Gaussian-SLAM [46] introduced a novel concept of sub-maps to tackle the problem of neural forgetting, which can occur when the SLAM system overwrites previously learned information during map updates. By maintaining sub-maps, the system preserves scene details while ensuring scalability.

Photo-SLAM [47], on the other hand, takes a hybrid approach by decoupling the processes of tracking and mapping. It employs a traditional visual SLAM algorithm [3]

for tracking and integrates a coarse-to-fine optimization strategy for refining the Gaussian map. This decoupling allows the system to leverage the robustness of classical SLAM algorithms while benefiting from the detailed scene representation provided by Gaussian Splatting.

The original Gaussian Splatting [5] algorithm renders depth by α-blending. However, this approach causes inconsistency when rendered from different points of view. RTG-SLAM [48] addresses depth-related challenges by considering only the foremost opaque Gaussians during depth rendering. This approach avoids ambiguities caused by overlapping Gaussians, resulting in more accurate depth maps.

The most recent work, Splat-SLAM [49], introduces the use of proxy depth maps to supervise the map optimization process. By using these proxy maps as references, the system achieves higher accuracy and consistency in the reconstructed scene.

Together, these advancements demonstrate the capability and popularity of Gaussian Splatting in SLAM systems within such a short period of time. However, MonoGS [40] and Photo-SLAM [47] lack dense depth maps, and Splat-SLAM [49] does not perform mapping efficiently, suffering from low frame rate.

In order to address all these drawbacks, IG-SLAM [10] is introduced as the main outcome of this thesis.

CHAPTER 5

IG-SLAM: INSTANT GAUSSIAN SLAM

Figure 5.1 presents an overview of the proposed method [10]. The main idea is to supervise real-time Gaussian Splatting mapping with the aid of dense depth maps generated by the tracking. The tracking algorithm in Section 5.1 produces a dense depth map, depth uncertainty, and the camera poses for each keyframe. These outputs serve as inputs to guide the mapping algorithm in Section 5.2. The initialization of Gaussians relies on the camera pose and dense depth, and their optimization is performed using color and weighted depth loss functions. The real-time performance is ensured by employing a sliding window of keyframes.

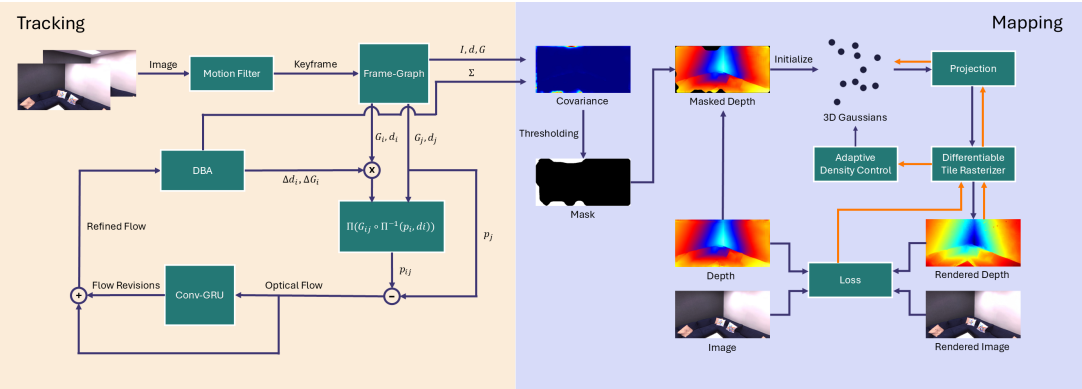

Figure 5.1: System Overview: The system has two main components: tracking and mapping. **Tracking** involves the creation of keyframes and optimization of the dense depth map, camera pose, and depth map covariance. **Mapping** reconstructs the 3D scene with Gaussian Splatting. The figure is adapted from [10]

5.1 Tracking

The system utilizes DROID-SLAM [13] as its tracking module. DROID-SLAM [13] maintains two key state variables for each camera frame t: the camera pose $\mathbf{G}_t$ and the inverse depth $\mathbf{d}_t$.

DROID-SLAM algorithm constructs a frame graph $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the keyframes and $\mathcal{E}$ represents the edges between them, based on co-visibility. Keyframes are selected from the available camera frames when the average optical flow magnitude exceeds a specified threshold. If there is a visual overlap between two frames i and j, an edge is established between the corresponding vertices in $\mathcal{V}$. This graph is dynamically updated during inference.

Given the initial estimates for pose and depth, $(\mathbf{G}_i,\mathbf{d}_i)$ and $(\mathbf{G}_j,\mathbf{d}_j)$, for frames i and j, the optical flow field is computed by unprojecting the pixels from frame i, reprojecting them into frame j, and calculating the pixel-wise positional difference. Specifically, the reprojected pixel locations p_{ij} are determined as described below:

$$p_{ij} = \Pi(\mathbf{G}_{ij} \circ \Pi^{-1}(\mathbf{p}_i, \mathbf{d}_i)), \quad \mathbf{p}_{ij} \in \mathbb{R}^{H \times W \times 2} \tag{5.1}$$

The relative transformation between frames i and j is defined as $\mathbf{G}_{ij} = \mathbf{G}_j^{-1} \circ \mathbf{G}_i$. Using this, the initial optical flow is computed as $p_{ij} - p_j$. This initial flow estimate is then passed through a Gated Recurrent Unit (GRU) architecture, along with a correlation vector derived from the inner product of the feature representations of the frames. GRU module outputs flow corrections $\mathbf{r}_{ij}$ and confidence weights $\mathbf{w}_{ij}$. The refined reprojected pixel locations $\mathbf{p}^*_{ij}$ are subsequently calculated in a manner similar to Equation (5.1), incorporating the flow corrections provided by the GRU. Finally, the dense bundle adjustment layer minimizes the cost function defined in Equation (5.2).

$$\begin{aligned} \mathbf{E}(\mathbf{G}', \mathbf{d}') &= \sum_{i,j \in \mathcal{E}} \left\| \mathbf{p}^*_{ij} - \mathbf{p'}_{ij} \right\|^2_{\Sigma_{ij}} \\ \mathbf{p'}_{ij} &= \Pi(\mathbf{G'}_{ij} \circ \Pi^{-1}(\mathbf{p}_i, \mathbf{d'}_i)) \end{aligned} \tag{5.2}$$

In the above equation, $\Sigma_{ij} = \text{diag}(\mathbf{w}_{ij})$, and $\|.\|_\Sigma$ represents the Mahalanobis norm weighted by the confidence weights $\mathbf{w}_{ij}$. Equation (5.2) is linearized around the current estimates $(\mathbf{G}', \mathbf{d}')$, and updates for the pose and depth, $(\Delta\xi, \Delta\mathbf{d})$, are computed using the Gauss-Newton algorithm. The linearized system of equations can then be expressed as in Equation (5.3).

$$H\mathbf{x} = \mathbf{b}, \quad H = \begin{bmatrix} C & E \\ E^T & P \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \Delta\xi \\ \Delta\mathbf{d} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} \tag{5.3}$$

In this context, H represents the Hessian matrix, $\mathbf{x} = [\Delta\xi, \Delta\mathbf{d}]$ denotes the updates for pose and depth, and $\mathbf{b} = [\mathbf{v}, \mathbf{w}]$ corresponds to the residuals for pose and depth. C is the block camera matrix, E consists of the off-diagonal block matrices for the camera and depth, and P is the diagonal matrix that accounts for disparities per pixel for each keyframe.

The bundle adjustment layer processes the initial flow estimates and updates the poses and depth maps of the keyframes. Optical flow is then revised by using the refined poses and depth maps, which are fed back into the dense bundle adjustment layer. After several iterations of refinement on the keyframe graph, the poses and depth maps are expected to converge.

Following the dense bundle adjustment step, the covariance for the depth estimates is calculated. As demonstrated in NeRF-SLAM [50], the same Hessian structure from Equation (5.3) is utilized to compute the covariance for both depth estimates Σ_d and poses Σ_G, as shown in Equation (5.4). The depth covariance serves a dual purpose: it acts as a mask for initializing Gaussians and as weights in the depth component of the loss function.

$$\begin{aligned} \Sigma_d &= P^{-1} + P^{-T}E^T\Sigma_G E P^{-1} \\ \Sigma_G &= (LL^T)^{-1} \end{aligned} \tag{5.4}$$

For keyframing, the system includes all keyframes that are actively optimized during the tracking process, without any filtering. Each keyframe used in the mapping pro-

cess contains its camera image I, depth map $\mathbf{d}$, depth covariance Σ_d, and pose G. A keyframe is only accepted for mapping if it is not already inside the sliding window. It is important to note that not all keyframes generated during a mapping cycle are sent; only the most recent keyframe is used. As a result, some keyframes might be excluded from the optimization. However, this design selection helps avoid sudden changes in the sliding window caused by abrupt camera movements.

Once the total number of keyframes exceeds the length of the sliding window for Dense Bundle Adjustment, the system periodically performs Global Bundle Adjustment on all the existing keyframes using a separate graph, as outlined in GO-SLAM [51]. This graph uses a distance metric, where the average optical flow magnitude determines the distance between frame pairs. The edges of the graph are formed between consecutive keyframes and those that are close based on the distance metric. Dense bundle adjustment is then applied to this graph every 10 keyframes. At the start of each mapping cycle, the pose and depth maps are updated, along with their corresponding covariances. Finally, a last global BA is performed at the end of the tracking process.

5.2 Mapping

The mapping process handles the 3D reconstruction using keyframes, which include pose, image, depth, and covariance data obtained from the tracking process.

The system uses Gaussian Splatting [5] for scene representation. The Gaussian function is defined the same as Equation (3.1).

In this formulation, μ and Σ represent the mean and covariance, which define the position and shape of the Gaussian. Σ is decomposed as RSS^TR^T, where R is the rotation matrix and S is the scaling matrix.

For rendering, the set of visible Gaussians $\mathcal{N}$ from a given viewpoint is initially projected onto the image plane. The 2D Gaussians are then arranged based on their depths and rasterized using α-blending, as outlined in Equation (5.5) for color and depth.

$$\hat{C} = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \ \hat{D} = \sum_{i \in \mathcal{N}} d_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{5.5}$$

Given the availability of dense depth maps for keyframes, a training strategy similar to RGB-D MonoGS [40] is employed, but with a coarse-to-fine approach inspired by Photo-SLAM [47] and InstantNGP [52].

For each keyframe, an image pyramid is created by downsampling the image, depth, and covariance by a factor of s using bilinear interpolation, as shown in Equation (5.6).

$$\begin{aligned} &\text{KF}_i^l = \{I_i^l, \mathbf{d}_i^l, \Sigma_{di}^l\} \\ &I_i^l = I_i^0 \downarrow s^l, \ \ \mathbf{d}_i^l = \mathbf{d}_i^0 \downarrow s^l, \ \ \Sigma_{di}^l = \Sigma_{di}^0 \downarrow s^l \end{aligned} \tag{5.6}$$

Here, $\downarrow$ represents the downsampling operation using linear interpolation, with l indicating the pyramid level, and I_i^0, $\mathbf{d}_i^0$, and Σ_{di}^0 referring to the full-resolution image, depth, and covariance, respectively. In Photo-SLAM [47], the authors use a sharp downsampling factor of 0.5 and a 2-level pyramid. In contrast, IG-SLAM [10] uses a smoother downsampling factor of 0.8, similar to Instant-NGP [52], along with a 3-level pyramid.

At each pyramid level, Gaussians are initialized through unprojection as follows: Points are randomly sampled from the most recent keyframe using a downsampling factor θ. These sampled points are then unprojected based on the depth maps. To address noise in the depth maps, regions with high covariance are masked out, making the Gaussian initialization more robust to noise. Equation (5.7) defines the mask for a given normalized depth covariance.

$$M = \{(i, j) \mid \sigma_{ij} < 0.2\} \tag{5.7}$$

Here, M denotes the binary mask matrix, with i and j indicating the pixel locations. The mask is generated by normalizing the covariance Σ_d to a range between 0 and 1 and identifying pixel values that fall below a threshold of 0.2 for the normalized

covariance σ_d. The mask is then smoothed through a thresholding operation, as described in Equation (5.7), which includes a maximum filter followed by a majority filter. An example of such a mask for a given covariance is shown in Figure 5.2

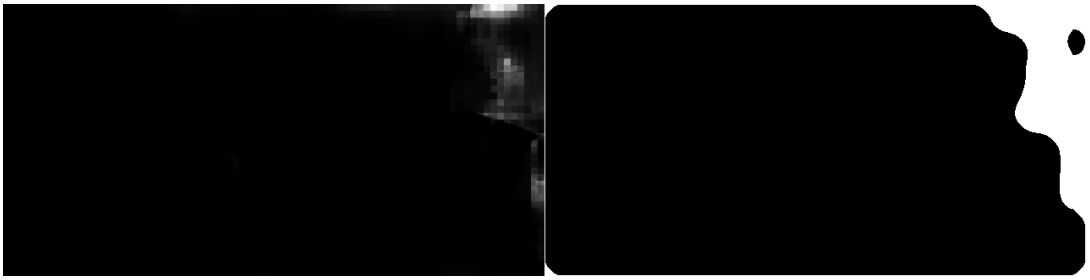

Figure 5.2: An example of normalized covariance (left) and corresponding mask (right). The figure is adapted from [10]

Map optimization is carried out within a sliding window using a coarse-to-fine approach. To meet real-time constraints, the last N keyframes within the sliding window are kept. As the number of iterations increases, the training is shifted to higher resolutions in the image pyramid. At the start of optimization at each level l, Gaussians are unprojected according to their corresponding depth map $\mathbf{d}_i^l$. The Gaussians are then rendered from the keyframes' viewpoints within the sliding window, and the loss function is computed based on the rendered image and depth. Camera images and dense depth maps are used as ground truth for mapping supervision.

The system uses a loss function that combines weighted depth loss L_{depth} and color loss L_{color}, as defined in Equation (5.8)

$$L_{\text{depth}} = \left\| D - \hat{D} \right\|_{\Sigma_d^{-1}}^1, \quad L_{\text{color}} = \left\| C - \hat{C} \right\|^1 \tag{5.8}$$

Here, D and C represent the ground truth depth and image, respectively, while $\hat{D}$ and $\hat{C}$ are the rendered depth and image, as defined in Equation (5.5). The depth loss L_{depth} is weighted by the inverse covariance to reduce the influence of pixels with high uncertainty. The total loss is given by $L = \alpha L_{\text{color}} + (1 - \alpha) L_{\text{depth}}$, with $\alpha = 0.5$ used in all of the experiments. This loss is then backpropagated through a differentiable Gaussian Splatting pipeline.

The system enhances the mapping results by refining the map through several iterations, following the methods used in MonoGS [40], GlORIE-SLAM [53], and Splat-SLAM [49]. To do this, individual frames are randomly selected and the map is optimized using the same loss function employed in the mapping process. For fairness, the same number of iterations as in MonoGS [40] and Splat-SLAM [49] are performed.

5.3 Training Strategy

A key aspect of our training strategy is that while dense depth maps might contain noise, they generally do not affect the depth order. In other words, using a position learning rate that causes Gaussians to shift positions during training is unnecessary and can actually slow down optimization convergence. This scenario is illustrated in Figure 5.3. It is important to note that this issue does not arise in standard Gaussian Splatting training, where the method typically begins with a sparse SfM point cloud. However, in our approach, since Gaussians are initialized from a dense depth map, they start off relatively close to each other.

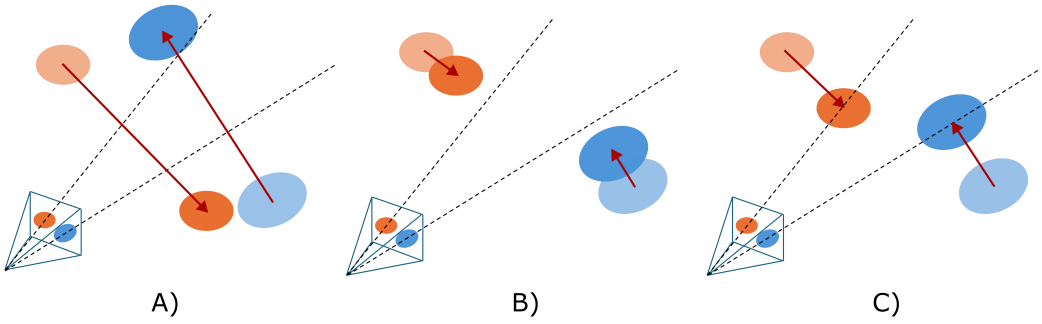

Figure 5.3: **Three hypothetical cases to encounter in training.** The figure is adapted from [10]

In Figure 5.3, the dashed lines indicate the ground truth positions of the Gaussians relative to the camera center. The faded Gaussians show their previous positions, while the red lines represent the position updates along the gradient direction. In Figure 5.3. A, a large position update causes the order of the Gaussians to shift, resulting in TV-static-like noise during training. In Figure 5.3. B, multiple iterations

are required to adjust the Gaussians to their correct positions due to smaller position updates.Figure 5.3. C illustrates the ideal scenario, where the position update exactly matches the position error.

Figure 5.3 demonstrates that high learning rates cause optimization to oscillate Gaussians around the target position. On the other hand, caseFigure 5.3. C, representing the opposite extreme, also impedes convergence. Since determining an ideal learning rate for each iteration is neither practical nor feasible, a decaying learning rate strategy is adopted during training, as described in Equation (5.9), to mitigate this "TV static" noise. The learning rate is initialized to encompass the full range required to refine the model from coarse to fine details, allowing for a gradual decay over time.

$$\mathrm{lr}(t) = \exp((1-t)\ln(\mathrm{lr}_i) + t\ln(\mathrm{lr}_f)) \tag{5.9}$$

Here, $t = n/\tau$ represents the iteration number n normalized by the decay constant τ, while lr_i and lr_f denote the initial and final learning rates, respectively. The effect of the learning rate and its decay on training performance is analyzed in Chapter 6.

Due to the coarse-to-fine approach and decay strategy, the proposed Gaussian Splatting training requires significantly fewer iterations. For instance, Splat-SLAM [49] typically performs 60 iterations per mapping call, whereas the proposed system usually completes the task in just 8 iterations. This efficiency allows our system to function on independent threads without relying on interleaved operations.

The same pruning and densification procedure are adopted as MonoGS [40]. Pruning is guided by occlusion-aware visibility: Gaussians initialized in recent keyframes are discarded, if they remain invisible from the current keyframe after optimization. Furthermore, Gaussians with opacity below 0.1 are removed every 150 mapping iterations. Densification occurs at the same interval by splitting larger Gaussians and duplicating smaller ones in areas with high-loss gradients.

CHAPTER 6

TESTS AND RESULTS

The system is evaluated on a variety of synthetic and real-world datasets. Ablation studies and hyperparameter analyses are included to support the design selections.

6.1 Experimental Setup

Datasets The system is evaluated on the Replica [54], TUM RGB-D [55], ScanNet [56], and EuRoC MAV [57] datasets.The Replica dataset consists of synthetic indoor scenes, while the TUM RGB-D dataset includes sequences recorded in small indoor office environments. ScanNet dataset features six sequences of real-world indoor environments, and EuRoC provides stereo images of larger-scale indoor spaces captured by a Micro Aerial Vehicle (MAV). Evaluations are conducted without clipping, except for EuRoC, where sequences are clipped at the start to skip initial pauses. Each sequence is run three times, and average results are reported to account for the non-deterministic behavior of multi-processing.

Metrics In line with conventions in view synthesis SLAM literature, the evaluation employs PSNR, SSIM, and LPIPS [58]. Additionally, a depth L1[cm] metric is provided, comparing predicted depths with ground truth in the Replica dataset. The evaluation is conducted after post-processing every 5 frames in the sequences, excluding keyframes used for mapping. This methodology is consistent with the evaluation strategies of MonoGS [40] and Splat-SLAM [49].

Implementation Details The system is tested on a PC equipped with a 3.6GHz AMD Ryzen Threadripper PRO 5975WX and an NVIDIA RTX 4090 GPU. For all ex-

periments, hyperparameters for mapping are set as $s = 0.8$, $\theta = 128$, $\alpha = 0.5$, $\text{lr}_i = 1.6 \times 10^{-4}$, $\text{lr}_f = 1.6 \times 10^{-6}$, and $\tau = 3000$. The parameter β is set to 2000 for the EuRoC [57] and Replica [54] datasets, and 26000 for the TUM RGB-D [55] and ScanNet [56] datasets, consistent with values used in MonoGS [40] and Splat-SLAM [49]. Tracking leverages pre-trained GRU weights from DROID-SLAM [13], with the mean optical flow threshold for keyframe selection set to 4.0 pixels and a local dense bundle adjustment window of 16. Tracking optimizations are performed using the LieTorch [59] framework. The mapping process only accepts the most recent keyframe after completing its optimization step, provided it is not already in the sliding window.

Baselines The system is compared against state-of-the-art RGB-only Gaussian Splatting and NeRF SLAM algorithms, including MonoGS [40], Photo-SLAM [47], GlORIE-SLAM [53], and Splat-SLAM [49].

MonoGS [40] is a leading SLAM algorithm focused solely on representation, utilizing Gaussian scene representation for both tracking and mapping. Like GlORIE-SLAM [53], Splat-SLAM [49], and the proposed system, Photo-SLAM [47] employs a decoupled tracking and mapping design. However, unlike the others, Photo-SLAM does not incorporate dense depth maps during mapping. GlORIE-SLAM and Splat-SLAM rely on monocular depth estimation [60] combined with a dense bundle adjustment layer. The key distinction between the two lies in their scene modeling: GlORIE-SLAM [53] uses NeRF [4], while Splat-SLAM [49] employs 3D Gaussian Splatting [5].

6.2 Evaluation

The system is compared with state-of-the-art algorithms based on rendering quality, 3D reconstruction accuracy, and runtime performance.

Rendering and Reconstruction Accuracy Rendering and reconstruction accuracy are evaluated for the Replica [54] dataset in Table 6.1, where the algorithm's performance is comparable to Splat-SLAM [49]. A direct comparison with GlORIE-SLAM [53] on the ScanNet [56] dataset, shown in Table 6.2, reveals that the system lags be-

hind Splat-SLAM [49]. The fast camera movements in ScanNet [56] lead to skipped keyframes during mapping, suggesting that reconstruction could improve with interleaved tracking and mapping. In Table 6.3, the system ranks just behind Splat-SLAM, but outperforms other algorithms on the TUM RGB-D [55] dataset. Notably, it excels in on-the-fly map optimization compared to Splat-SLAM, as seen in Table 6.6. The system outperforms others on the EuRoC [57] dataset, achieving a significant margin over Photo-SLAM [47], and produces photorealistic reconstruction results even for large-scale sequences in EuRoC [57]. A qualitative comparison is provided in Figure 6.1 and Figure 6.2. The performance on the EuRoC MH-01 dataset can be viewed via this link.

Metrics	Mono-GS [40]	GlORIE-SLAM [53]	Photo-SLAM [47]	Splat - SLAM [49]	**Ours**
PSNR↑	31.22	31.04	33.30	**36.45**	36.21
SSIM ↑	0.91	0.91	0.93	0.95	**0.96**
LPIPS↓	0.21	0.12	-	0.06	**0.05**
Depth L1↓	-	-	-	**2.41**	4.34

Table 6.1: **Rendering and Tracking Results on the Replica Dataset [54] for RGB-Based Methods.** The results are averaged over 8 scenes, with each scene's result being the average of 3 runs. Data is taken from [49], except for our results. The top results are highlighted as **first** and second. Our method demonstrates performance comparable to Splat-SLAM [49] while outperforming all other methods. The table is adapted from [10].

Runtime Analysis The real-time performance of our algorithm is evaluated in Table 6.5. Benchmarks were conducted on a 3.6GHz AMD Ryzen Threadripper PRO 5975WX and an NVIDIA GeForce RTX 4090 with 24 GB of memory. Our system achieves 9.94 fps, making it 8 times faster than the single-process implementation of Splat-SLAM [49]. Our approach outperforms other algorithms while maintaining visual quality. In its reference multi-process implementation, our method reaches a frame rate of 16 fps. The peak memory usage and map size of our method are similar to those of existing methods.

Figure 6.1: **Qualitative rendering results from Photo-SLAM [47] and IG-SLAM**. The visual quality of the methods is compared on the large-scale EuRoC dataset [57].

Method	Metric	0000	0059	0106	0169	0181	0207	Avg.
MonoGS [40]	PSNR↑	16.91	19.15	18.57	20.21	19.51	18.37	18.79
	SSIM ↑	0.62	0.69	0.74	0.74	0.75	0.70	0.71
	LPIPS↓	0.70	0.51	0.55	0.54	0.63	0.58	0.59
GlORIE-SLAM [53]	PSNR↑	23.42	20.66	20.41	25.23	21.28	23.68	22.45
	SSIM ↑	**0.87**	**0.87**	0.83	0.84	**0.91**	0.76	**0.85**
	LPIPS↓	0.26	0.31	0.31	0.21	0.44	0.29	0.30
Splat-SLAM [49]	PSNR↑	**28.68**	**27.69**	**27.70**	**31.14**	**31.15**	**30.49**	**29.48**
	SSIM ↑	0.83	**0.87**	**0.86**	**0.87**	0.84	**0.84**	**0.85**
	LPIPS ↓	**0.19**	**0.15**	**0.18**	**0.15**	**0.23**	**0.19**	**0.18**
IG-SLAM (Ours)	PSNR↑	24.68	20.09	25.30	27.85	25.80	26.69	25.07
	SSIM ↑	0.74	0.68	0.83	0.82	0.83	0.78	0.78
	LPIPS ↓	0.29	0.39	0.22	0.19	0.27	0.27	0.27

Table 6.2: **Rendering Performance on ScanNet [56].** The result for each scene is averaged over 3 runs. The values are taken from [49], except for our own. Our method demonstrates competitive performance compared to state-of-the-art techniques, achieving the second-highest visual quality results. The table is adapted from [10]

.

Since both Splat-SLAM [49] and our system use DROID-SLAM [13] for tracking, the difference in runtime performance can be attributed solely to the efficiency of our mapping design. Our system performs 8 iterations per mapping call, whereas

Splat-SLAM performs 60 iterations, making our system 8 times faster. The slight runtime improvement over GO-SLAM [51] is likely due to the faster training process of Gaussian Splatting [5] compared to NeRF [4].

Qualitative Results As shown in Figure 6.1 and Figure 6.2, photorealistic mapping is achieved even for relatively large-scale scenes, such as those in the EuRoC dataset [57]. Small-scale details, like pipes, are accurately captured within these expansive environments. The rapid convergence of on-the-fly mapping allows our system to build complex scenes with minimal reliance on post-processing. A detailed ablation analysis of the post-processing operations can be found in Section 6.3.

Figure 6.2: **Qualitative results of IG-SLAM on EuRoC [57].** The results in the *first row* and *second row* correspond to `MH-02` and `MH-03`, respectively. The figure is adapted from [10]

.

6.3 Ablations

Post-processing, decay, and weighted depth loss are key design choices in the system. Ablation studies are provided to validate and justify each of these decisions.

Method	Metric	f1/desk	f2/xyz	f3/off	**Avg.**
Photo-SLAM [47]	PSNR↑	20.97	21.07	19.59	20.54
	SSIM ↑	0.74	0.73	0.69	0.72
	LPIPS ↓	0.23	0.17	0.24	0.21
MonoGS [40]	PSNR↑	19.67	16.17	20.63	18.82
	SSIM ↑	0.73	0.72	0.77	0.74
	LPIPS ↓	0.33	0.31	0.34	0.33
GlORIE-SLAM [53]	PSNR↑	20.26	25.62	21.21	22.36
	SSIM ↑	0.79	0.72	0.72	0.74
	LPIPS ↓	0.31	0.09	0.32	0.24
Splat-SLAM [49]	PSNR↑	**25.61**	**29.53**	**26.05**	**27.06**
	SSIM ↑	**0.84**	**0.90**	**0.84**	**0.86**
	LPIPS ↓	**0.18**	**0.08**	0.20	**0.15**
IG-SLAM **(Ours)**	PSNR↑	24.45	26.35	25.27	25.36
	SSIM ↑	0.80	0.85	0.83	0.83
	LPIPS ↓	0.20	0.10	**0.17**	0.16

Table 6.3: **Rendering Performance on TUM-RGBD [55].** The result for each scene is averaged over 3 runs. The values are taken from [49], except for our own. Our method shows performance comparable to Splat-SLAM [49], with a clear performance advantage over other methods. The table is adapted from [10]

Post Processing Post-processing ablation results are presented in Table 6.6. PSNR and Depth L1 metrics are recalculated every 500 post-processing iterations. Our method shows only a slight decrease in visual quality when post-processing is skipped (denoted as 0K in Table 6.6), while Splat-SLAM [49] experiences a significant drop in visual quality without post-processing. Our system demonstrates diminishing returns as the number of post-processing iterations increases. The rapid convergence of our map, high frame rate, and reduced dependency on post-processing are attributed to our training strategy.

Decay Learning rate decay ablation results are shown in Table 6.7. Three constant learning rates without decay are compared against decaying learning rates. The chosen learning rates are $\text{lr}_f = 1.6 \times 10^{-6}$ (lower bound), $\text{lr}_i = 1.6 \times 10^{-4}$ (upper bound), and the mean learning rate value of 5×10^{-5}, as calculated in Equation (5.9). This

Method	Metric	MH-01	MH-02	V1-01	V2-01	**Avg.**
Photo-SLAM [47]	PSNR↑	13.95	14.20	17.07	15.68	15.23
	SSIM ↑	0.42	0.43	0.62	0.62	0.52
	LPIPS ↓	0.37	0.36	**0.27**	0.32	0.33
IG-SLAM **(Ours)**	PSNR↑	**22.33**	**22.31**	**20.55**	**24.59**	**22.44**
	SSIM ↑	**0.78**	**0.77**	**0.79**	**0.85**	**0.80**
	LPIPS ↓	**0.22**	**0.23**	0.29	**0.18**	**0.23**

Table 6.4: **Rendering Performance on EuRoC [57].** The result for each scene is averaged over 3 runs. The values for Photo-SLAM [47] are taken from their work. A photorealistic 3D reconstruction comparison of the large indoor environment EuRoC [57] MH-01 is shown in Figure 6.1. The table is adapted from [10].

	GO-SLAM [51]	GlORIE-SLAM [53]	MonoGS [40]	Splat-SLAM [49]	**Ours**
GPU Usage [GiB]	18.50	15.22	**14.62**	17.57	16.20
Map Size [MB]	-	114.0	6.8	**6.5**	14.8
Avg. FPS	8.36	0.23	0.32	1.24	**9.94**

Table 6.5: Memory and Running Time Evaluation on Replica [54] room0. The runtime statistics are measured using the single-process implementation of our method. The values are taken from [49], except for our own. Our peak memory usage and map size are similar to those of existing works. However, our method achieves state-of-the-art 3D reconstruction at higher frame rates compared to other methods. The table is adapted from [10].

experiment is conducted with and without post-processing. As seen in the no post-processing scenario in Table 6.7, learning with decay significantly improves visual quality compared to non-decaying learning rates. Qualitative results can be seen in Figure 6.3, where fine details are missed with non-decaying learning rates. Additionally, post-processing completely mitigates the convergence issues associated with constant learning rates, as shown in the post-processing experiment in Table 6.7. The decay strategy allows our system to use fewer iterations per mapping call without

Nbr of Final Iterations β	Metric	0K	0.5K	1K	2K
Splat-SLAM [49]	PSNR ↑	30.50	39.87	40.59	41.20
	Depth L1 ↓	6.55	2.37	2.34	2.40
Ours	PSNR ↑	**38.30**	**40.92**	**41.53**	**41.68**
	Depth L1 ↓	**2.63**	**2.18**	**2.17**	**2.30**

Table 6.6: **Post-processing Iterations Ablation on Replica [54] `office0`**. The values for Splat-SLAM [49] are taken from their work. Thanks to the rapid convergence of mapping during tracking, our method does not rely heavily on post-processing. The reconstruction shows only slight improvements from post-processing. The table is adapted from [10].

Figure 6.3: **Qualitative Results for Learning Rate Decay Ablation Study.** The four cases examined in Table 6.7 are presented in the figure. The results include constant learning rates of 1.6×10^{-4} at the *top-left*, 5×10^{-5} at the *top-right*, 1.6×10^{-6} at the *bottom-left*, and the decaying 1.6×10^{-4} learning rate at the *bottom-right* as a reference. The figure is adapted from [10]

sacrificing visual quality. The impact of this efficient mapping design on runtime is shown in Table 6.5.

Metric	Learning Rate	1.6×10^{-6}	5×10^{-5}	1.6×10^{-4}	1.6×10^{-4} w/ decay
w/o Post Processing					
PSNR ↑		31.92	35.84	34.71	**38.30**
Depth L1 ↓		5.37	2.71	2.76	**2.63**
w/ Post Processing					
PSNR ↑		39.71	39.91	40.85	**41.68**
Depth L1 ↓		2.73	**2.17**	2.20	2.30

Table 6.7: **Learning Rate Hyperparameter Search on Replica [54] `office0`.** Our system performs significantly better with a slow learning rate combined with decay. When reliable depth maps are available, an excessively high learning rate leads to TV-static noise and hinders the convergence of the map. The table is adapted from [10].

Depth Loss The ablation results for the weighted depth loss are presented in Table 6.8. The weighted depth loss, as defined in Equation (A.14), is compared to scenarios where no depth loss is included in the overall loss function ($\alpha = 1$) and where raw depth values are used without applying depth covariance weighting. Post-processing is disabled to ensure the results are not influenced.

Metric	Weighted	No Depth	Raw Depth
PSNR↑	**31.91**	31.56	30.81
Depth L1 ↓	**6.33**	13.16	6.39

Table 6.8: **Weighted Depth Loss Ablation on Replica [54] `office2`.** The utilization of weighted depth loss improves reconstruction quality while maintaining visual quality.

As it can be observed from Table 6.8, the weighted depth loss outperforms the other options. While a pure color loss yields good visual quality, it degrades the reconstruction quality. Using raw depth values in the loss function results in poorer visual quality compared to the weighted loss. Weighting the depth helps maintain visual quality by preventing degradation in high uncertainty regions, while also improving

reconstruction quality by supervising the depth. The differences in visual quality are not drastic, possibly due to the fact that the proposed system initializes Gaussians based on depth maps, meaning that the initialized Gaussians are already close to the corresponding depth values.

CHAPTER 7

CONCLUSIONS

This thesis presents IG-SLAM, a novel RGB-only dense SLAM system that combines robust tracking methods with the promising 3D Gaussian Splatting technique. Through this work, we addressed the challenges of efficiently handling large-scale environments while maintaining high-quality 3D reconstruction. By integrating accurate pose and dense depth information, along with leveraging depth uncertainty in map optimization, IG-SLAM achieves photo-realistic results and operates at significantly higher speeds than previous state-of-the-art RGB-only SLAM systems.

The experiments conducted on various datasets, including Replica, TUM-RGBD, ScanNet, and EuRoC, demonstrate IG-SLAM's competitive performance, particularly in large-scale environments where our system excels. The decay strategy incorporated in the optimization process not only enhances convergence but also ensures real-time operation, making it a viable solution for practical applications.

This work lays the foundation for future advancements in RGB-only SLAM systems, with potential improvements in both optimization techniques and real-time performance. The combination of Gaussian Splatting and dense SLAM provides a promising direction for high-quality 3D mapping and reconstruction in dynamic and complex environments.

Future research could focus on extending IG-SLAM to handle dynamic scenes more effectively, as well as improving scalability and robustness for even larger environments. The insights and contributions made in this thesis contribute to the growing field of SLAM, with potential applications in robotics, augmented reality, and beyond.

REFERENCES

[1] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4666–4672, IEEE, 2020.

[2] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[4] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[5] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.

[6] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15651–15663, 2020.

[7] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *arXiv preprint arXiv:2106.10689*, 2021.

[8] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12786–12796, 2022.

[9] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6229–6238, 2021.

[10] F. A. Sarikamis and A. A. Alatan, "Ig-slam: Instant gaussian slam," *arXiv preprint arXiv:2408.01126*, 2024.

[11] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE international conference on robotics and automation*, pp. 3565–3572, IEEE, 2007.

[12] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*, pp. 2320–2327, IEEE, 2011.

[13] Z. Teed and J. Deng, "Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras," *Advances in neural information processing systems*, vol. 34, pp. 16558–16569, 2021.

[14] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*, pp. 834–849, Springer, 2014.

[15] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, 2004.

[16] C. Harris and J. Pike, "3d positional integration from image sequences," *Image and Vision Computing*, vol. 6, no. 2, pp. 87–90, 1988. 3rd Alvey Vision Meeting.

[17] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[18] W. Forstner, "A feature based correspondence algorithm for image matching," in *International Archives of Photogrammetry and Remote Sensing, v.26(3/3)* (I. S. o. Photogrammetry and R. Sensing, eds.), pp. 150–166, International Society of Photogrammetry and Remote Sensing (ISPRS) - Commission III, International Society of Photogrammetry and Remote Sensing (ISPRS), 1986.

[19] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 430–443, Springer Berlin Heidelberg, 2006.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[21] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pp. 404–417, Springer, 2006.

[22] M. Agrawal, K. Konolige, and M. R. Blas, "Censure: Center surround extremas for realtime feature detection and matching," in *European conference on computer vision*, pp. 102–115, Springer, 2008.

[23] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel corporation*, vol. 5, no. 1-10, p. 4, 2001.

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[25] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[26] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[27] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[28] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[30] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, pp. 778–792, Springer, 2010.

[31] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[32] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 559–568, 2011.

[33] C. Tang and P. Tan, "Ba-net: Dense bundle adjustment network," *arXiv preprint arXiv:1806.04807*, 2018.

[34] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2560–2568, 2018.

[35] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pp. 402–419, Springer, 2020.

[36] C. Smith, D. Charatan, A. Tewari, and V. Sitzmann, "Flowmap: High-quality camera poses, intrinsics, and depth via gradient descent," *arXiv preprint arXiv:2404.15259*, 2024.

[37] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Ewa volume splatting," in *Proceedings Visualization, 2001. VIS'01.*, pp. 29–538, IEEE, 2001.

[38] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016.

[39] G. Chen and W. Wang, "A survey on 3d gaussian splatting," *arXiv preprint arXiv:2401.03890*, 2024.

[40] H. Matsuki, R. Murai, P. H. Kelly, and A. J. Davison, "Gaussian splatting slam," *arXiv preprint arXiv:2312.06741*, 2023.

[41] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *ACM SIGGRAPH 2024 conference papers*, pp. 1–11, 2024.

[42] D. Chen, H. Li, W. Ye, Y. Wang, W. Xie, S. Zhai, N. Wang, H. Liu, H. Bao, and G. Zhang, "Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction," *arXiv preprint arXiv:2406.06521*, 2024.

[43] B. Zhang, C. Fang, R. Shrestha, Y. Liang, X. Long, and P. Tan, "Rade-gs: Rasterizing depth in gaussian splatting," *arXiv preprint arXiv:2406.01467*, 2024.

[44] C. Yan, D. Qu, D. Xu, B. Zhao, Z. Wang, D. Wang, and X. Li, "Gs-slam: Dense visual slam with 3d gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19595–19604, 2024.

[45] N. Keetha, J. Karhade, K. M. Jatavallabhula, G. Yang, S. Scherer, D. Ramanan, and J. Luiten, "Splatam: Splat track & map 3d gaussians for dense rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21357–21366, 2024.

[46] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, "Gaussian-slam: Photo-realistic dense slam with gaussian splatting," *arXiv preprint arXiv:2312.10070*, 2023.

[47] H. Huang, L. Li, H. Cheng, and S.-K. Yeung, "Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular stereo and rgb-d cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21584–21593, 2024.

[48] Z. Peng, T. Shao, Y. Liu, J. Zhou, Y. Yang, J. Wang, and K. Zhou, "Rtg-slam: Real-time 3d reconstruction at scale using gaussian splatting," in *ACM SIGGRAPH 2024 Conference Papers*, pp. 1–11, 2024.

[49] E. Sandström, K. Tateno, M. Oechsle, M. Niemeyer, L. Van Gool, M. R. Oswald, and F. Tombari, "Splat-slam: Globally optimized rgb-only slam with 3d gaussians," *arXiv preprint arXiv:2405.16544*, 2024.

[50] A. Rosinol, J. J. Leonard, and L. Carlone, "Nerf-slam: Real-time dense monocular slam with neural radiance fields," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3437–3444, IEEE, 2023.

[51] Y. Zhang, F. Tosi, S. Mattoccia, and M. Poggi, "Go-slam: Global optimization for consistent 3d instant reconstruction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3727–3737, 2023.

[52] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM transactions on graphics (TOG)*, vol. 41, no. 4, pp. 1–15, 2022.

[53] G. Zhang, E. Sandström, Y. Zhang, M. Patel, L. Van Gool, and M. R. Oswald, "Glorie-slam: Globally optimized rgb-only implicit encoding point cloud slam," *arXiv preprint arXiv:2403.19549*, 2024.

[54] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Y. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. De Nardi, M. Goesele, S. Lovegrove, and R. A. Newcombe, "The replica dataset: A digital replica of indoor spaces," *CoRR*, vol. abs/1906.05797, 2019.

[55] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 573–580, IEEE, 2012.

[56] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," *CoRR*, vol. abs/1702.04405, 2017.

[57] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[58] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

[59] Z. Teed and J. Deng, "Tangent space backpropagation for 3d transformation groups," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10338–10347, 2021.

[60] A. Eftekhar, A. Sax, J. Malik, and A. Zamir, "Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10786–10796, 2021.

[61] J. T. Kajiya and B. P. Von Herzen, "Ray tracing volume densities," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[62] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5855–5864, 2021.

[63] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5875–5884, 2021.

[64] K. Rematas and V. Ferrari, "Neural voxel renderer: Learning an accurate and controllable rendering tool," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5417–5427, 2020.

[65] T. Hu, S. Liu, Y. Chen, T. Shen, and J. Jia, "Efficientnerf efficient neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12902–12911, 2022.

[66] L. Wang, J. Zhang, X. Liu, F. Zhao, Y. Zhang, Y. Zhang, M. Wu, J. Yu, and L. Xu, "Fourier plenoctrees for dynamic radiance field rendering in real-time," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13524–13534, 2022.

[67] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5752–5761, 2021.

[68] M. Oechsle, S. Peng, and A. Geiger, "Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5589–5599, 2021.

[69] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman, "Volume rendering of neural implicit surfaces," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4805–4815, 2021.

[70] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, "inerf: Inverting neural radiance fields for pose estimation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1323–1330, IEEE, 2021.

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[72] Z. Zhu, S. Peng, V. Larsson, Z. Cui, M. R. Oswald, A. Geiger, and M. Pollefeys, "Nicer-slam: Neural implicit scene encoding for rgb slam," in *2024 International Conference on 3D Vision (3DV)*, pp. 42–52, IEEE, 2024.

[73] M. M. Johari, C. Carta, and F. Fleuret, "Eslam: Efficient dense slam system based on hybrid representation of signed distance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17408–17419, 2023.

[74] H. Wang, J. Wang, and L. Agapito, "Co-slam: Joint coordinate and sparse parametric encodings for neural real-time slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13293–13302, 2023.

[75] H. Zhou, Z. Guo, Y. Ren, S. Liu, L. Zhang, K. Zhang, and M. Li, "Mod-slam: Monocular dense mapping for unbounded 3d scene reconstruction," *IEEE Robotics and Automation Letters*, 2024.

APPENDIX

A.1 Gaussian Splatting Jacobian Derivation

Let $\mathbf{v}$ be a 6 dimensional vector in Lie algebra $\mathfrak{se}(3)$ defined as $[\mathbf{t}, \mathbf{w}]^T \in \mathfrak{se}(3)$ where $\mathbf{t}$ is 3D translation and $\mathbf{w}$ is 3-vector rotation. The exponential map $exp : \mathfrak{se}(3) \mapsto \mathbf{SE}(3)$ and logarithm map $log : \mathbf{SE}(3) \mapsto \mathfrak{se}(3)$ is utilized to calculate the derivative on $\mathbf{SE}(3)$ manifold. Intuitively, the exponential function maps an arbitrary pose 6-vector v to the $\mathbf{SE}(3)$ manifold where the geometry is defined by constraints of proper rotation and translation. The left-Jacobian of the Lie group is defined as in Equation (A.1)

$$\frac{\mathcal{D}f(T)}{\mathcal{D}T} := \lim_{\tau \to 0} \frac{log(f(exp(\tau) \circ T) \circ f(T)^{-1})}{\tau} \tag{A.1}$$

where the infinitesimal $\tau \in \mathfrak{se}(3)$ is elevated to the manifold and the derivative is taken there. Note that the left Jacobian is convenient at this formulation since concatenation of the poses is thought to be $T = T_1 \circ T_0$. In other words, the poses are concatenated by left. The derivatives, therefore, are given as in Equation (A.2)

$$\frac{\mathcal{D}\mu_C}{\mathcal{D}T_{CW}} = [\mathbf{I} \ \ -\mu_C^{\times}], \ \ \frac{\mathcal{D}W}{\mathcal{D}T_{CW}} = \begin{bmatrix} \mathbf{0} & -\mathbf{W}_{:,\mathbf{1}}^{\times} \\ \mathbf{0} & -\mathbf{W}_{:,\mathbf{2}}^{\times} \\ \mathbf{0} & -\mathbf{W}_{:,\mathbf{3}}^{\times} \end{bmatrix} \tag{A.2}$$

where $^{\times}$ represents the skew-symmetric matrix of the 3-vector and $W_{:,i}$ is the ith column of the matrix W. The dimensions of $\frac{\mathcal{D}\mu_C}{\mathcal{D}T_{CW}}$ and $\frac{\mathcal{D}W}{\mathcal{D}T_{CW}}$ is 3x6 and 9x6 respectively. Therefore, there are no unnecessary derivatives at the Jacobian.

A.2 Neural Radiance Fields

Neural Radiance Fields (NeRF) [4] is another novel view synthesis method. It aims to render the environment of interest from any direction and position in the environment. To achieve this aim, direction-dependent color, and volume density are learned by a Multilayer Perceptron (MLP). Classical view synthesis methods before NeRF tackled the same problem by modeling the environment in a discrete manner, whereas in the NeRF paradigm, the environment and rendering function are modeled continuously.

In other words, NeRF models the environment at every point in the space from every direction as a simple model of color and volume density. Therefore, a simple volume rendering method can be utilized to render an image of the environment from any point and direction. After rendering, the rendered image can be compared to the original image taken from that point. Since MLP and volume rendering are differentiable operations, the system can be trained in a self-supervised manner.

In theory, several posed images are required to train NeRF. However, the pose of the images can be calculated with a structure-from-motion (SfM) algorithm such as COLMAP [38]. The idea is to render images of the environment from the positions of posed images and backpropagate to train the MLPs.

A.2.1 Theory

In NeRF, an MLP is utilized to calculate the color and volume density of a point in the environment. The purpose of the MLP is to calculate color and volume density at sample points at are being used in volume rendering. The input-output relationship of the NeRF MLP is given in Figure A.1.

Figure A.1: Input-Output relationship of NeRF MLP

where x,y,z is the position in the environment and θ, ϕ is the viewing direction in spherical coordinate angles. NeRF MLP is executed in every sample point to render

a pixel. Therefore, the architecture must be simple. The MLP in the original implementation is a rather simple network that has only 10 fully connected layers. The MLP used in the original implementation is given in Figure A.2.

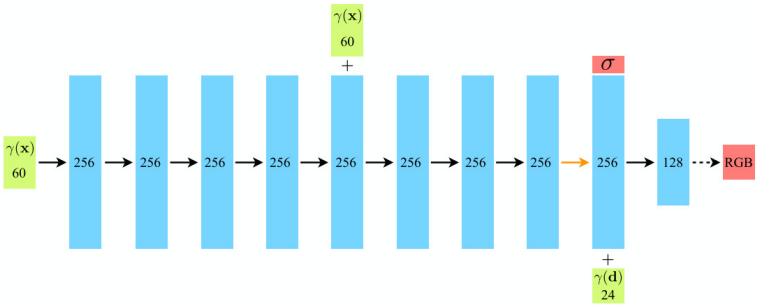

Figure A.2: The MLP used in the original implementation [4]. $\gamma(\mathbf{x})$ and $\gamma(\mathbf{d})$ are the positional encodings of the position vector and direction vector. The numbers on the blocks indicate the vector dimensions. Green boxes and red boxes represent the input and the output.

The position is the input of MLP and additionally, it is concatenated at the 5^{th} layer as well. The direction is concatenated in the 9^{th} layer. At the same layer, MLP outputs volume density. The final output of the MLP is the RGB color. Unlike the general machine learning algorithms, the NeRF MLP is a quite modest architecture. This selection is because, unconventionally, memorization is aimed at NeRF. The aim of the MLP is not to generalize the environment but rather to memorize the environment.

Assume that volume density and color of all space is given, continuous volume rendering algorithm [61] is as follows. At every point on a ray $\mathbf{r}$ that starts from camera center $\mathbf{o}$ through pixel of interest, the color is calculated. These colors on the line are weighted and averaged to get the pixel color. The weights of the points are both volume density σ and accumulated transmittence T. The equation to calculate the color of the pixel corresponds to the ray $\mathbf{r}$ is given in Equation (A.3)

$$C(\mathbf{r}) = \int_{t_1}^{t_2} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{A.3}$$

$\sigma(\mathbf{r}(t))$ is the volume density of a point on the ray and $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ is the direction-dependent color of a point on the ray. The boundaries of the integral t_1 and t_2 are

known as near and far bounds. The accumulated transmittance term $T(t)$ is an intuitive term given in

$$T(t) = \exp(-\int_{t_1}^{t_2} \sigma(\mathbf{r}(u))du) \tag{A.4}$$

The intuition in Equation (A.4) is as follows. The integral part increases as the ray passes through opaque points. Therefore, transmittance decreases as more and more opaque points are passed through. Meaning that a point's weight is less when it is occluded. The discretization of the equations is rather simple. Instead of continuous integration, the points are sampled on the ray, and a weighted average is taken on those points. The discrete equations are given as in Equation (A.5)

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \alpha_i T_i \mathbf{c}_i, \;\; T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j) \tag{A.5}$$

where δ_i is the distance between the sample points i and $i+1$,$(\sigma_i, \mathbf{c}_i)$ is the volume density and color of the sample points. α_i is the opacity term from alpha compositing given as

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i) \tag{A.6}$$

An illustration of the volume rendering process is shown in Figure A.3.

In NeRF algorithm, $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ and $\sigma(\mathbf{r}(t))$ is calculated with the MLP. Note that the calculation of color and volume density is done with MLP, which is, obviously a differentiable process. Volume rendering after discretization is just a simple summation operation, which is again differentiable. Therefore, the whole system is differentiable if a simple error function as a square error between the ground truth pixel and the rendered pixel is utilized. In other words, L2-norm loss is utilized as in equation A.7

$$L = \sum_{r \in R} ||\hat{C}(\mathbf{r}) - C_{gt}(\mathbf{r})||_2^2 \tag{A.7}$$

where R is the set of rays associated with the pixels of the image. The volume density accumulation can also be used to predict depth. The predicted depth is given in the

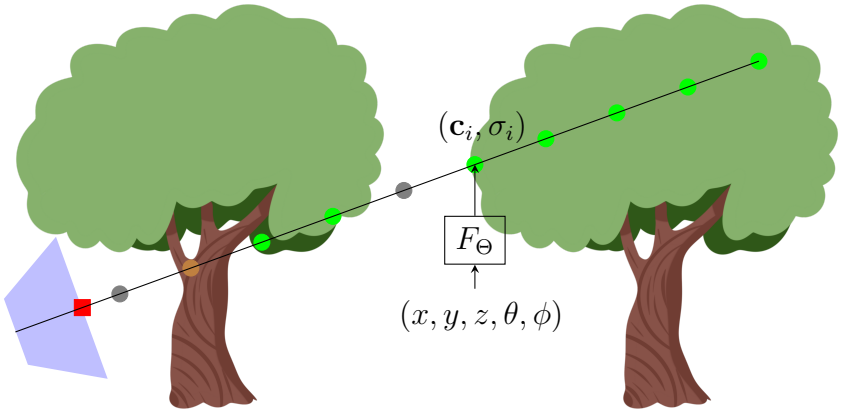

Figure A.3: An example volume rendering scenario where the pixel to be rendered is shown as a red square, the black line passes through the camera center, and the pixel, the points are sampled on the line. At each sample, color and volume density are calculated with MLP. The samples are, then, used in volume rendering. Repeating the same procedure for all pixels results in the output image.

equation A.8

$$d(\mathbf{r}) = \int_{t_1}^{t_2} T(t)\sigma(\mathbf{r}(t))t dt \tag{A.8}$$

The equation is quite intuitive. As the ray passes through an opaque medium, $T(t)$ decreases exponentially. Therefore, the terms after a quite opaque point are small and the integral total approximates to the depth. Discretizing the equation A.8, results is in the summation form A.9

$$\hat{D}(r) = \sum_{i=1}^{N} \alpha_i t_i T_i \tag{A.9}$$

The original implementation, as illustrated in A.1, does not utilize position directly. Instead of 3 term position x, y, z and three-term direction $\mathbf{d}$, the input of the MLP is $\gamma(\mathbf{x})$ which is given in general form as in equation A.10

$$\begin{aligned}\gamma(\nu) = (&\sin(2^0\pi\nu), \cos(2^0\pi\nu), \sin(2^1\pi\nu), \cos(2^1\pi\nu), \\ &\cdots, \sin(2^{N-1}\pi\nu), \cos(2^{N-1}\pi\nu))\end{aligned} \tag{A.10}$$

In the original implementation $N = 10$ for position and $N = 4$ for direction. The

reason for positional encoding is to increase the dependency of MLP on position and direction. In other words, with positional encoding small position differences are better resolved than directly using the position and direction.

In the original implementation, 256 points are sampled per ray. For an 800x800 image, 640k rays are needed. Therefore, NeRF MLP needs to be called for approximately 160 million times. This corresponds to tens of seconds to render an image and hours or even days to train a model. The comparison between NeRF and Gaussian Splatting is given in Figure A.4

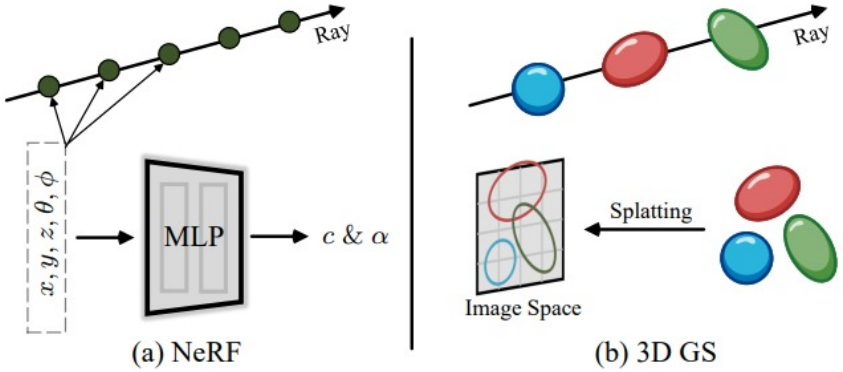

Figure A.4: (a) NeRF operates by sampling points along a ray and querying a neural network (MLP) to retrieve corresponding colors and opacities, a process that resembles backward mapping (ray tracing). (b) Conversely, 3D Gaussian Splatting projects all 3D Gaussians directly onto the image plane (splatting) and executes parallel rendering, analogous to forward mapping (splatting and rasterization). The figure is adapted from [39]

While NeRF in its original form is quite slow to be a part of the SLAM system, some improvements have been made to accelerate the training and rendering process.

A.2.2 Related Work

Despite its revolutionary impact, the original NeRF [4] formulation has notable limitations, particularly its slow training and rendering speeds, which hinder its practicality for real-time or large-scale applications. These inefficiencies have motivated researchers to propose a variety of enhancements and optimizations aimed at acceler-

ating NeRF's performance without sacrificing its quality.

One such improvement involves cone-shaped rendering [62], which addresses the issue of anti-aliasing by introducing a more physically accurate method for integrating radiance over a cone, rather than sampling individual rays. This modification not only improves rendering quality but also reduces aliasing artifacts in the synthesized images.

Additionally, several data structures have been employed to enhance the efficiency of NeRF. Voxel grids [63, 6, 64], which divide the 3D space into discrete cells, enabling faster access and computation of radiance values. By combining voxel-based representations with neural networks, these methods achieve substantial speedups.

Plenoctrees [65, 66, 67], which utilize a hierarchical octree structure to represent radiance fields at multiple levels of detail. This approach efficiently allocates computational resources by focusing on regions with higher detail requirements.

Hash tables [52], which map spatial coordinates to compact representations stored in hash maps, significantly reducing memory usage and computation time. This technique has proven particularly effective in achieving orders of magnitude faster training and rendering speeds compared to the original NeRF formulation.

Beyond these structural improvements, surface-based methods [68, 7, 69] have also been developed to unify surface and volume rendering paradigms. These approaches explicitly incorporate surface information into the radiance field representation, enabling more accurate modeling of scene geometry and achieving improved rendering quality. By bridging the gap between surface-based and volumetric methods, these techniques offer a more holistic representation of scenes, further expanding the versatility of NeRF.

Together, these advancements highlight the rapid evolution of NeRF-based methods, transforming the original formulation into a highly efficient and scalable framework for 3D scene representation and rendering. These improvements have broadened NeRF's applicability, making it increasingly viable for use in real-world scenarios requiring high-quality, real-time performance.

A.3 Neural Radiance Fields SLAM

As an alternative to standard point cloud representation of the environment, NeRF, represents the environment in the sense that if the environment could be rendered from virtually any position and direction that would mean the environment is represented. This fundamentally different representation approach can be implemented in two different ways. The first way utilizes NeRF in both tracking and mapping. The second way is to make use of classical algorithms in tracking and maps with NeRF.

A.3.1 Inverting Neural Radiance Fields

It should be noted that the NeRF algorithms, which are the main block as in Figure A.1, take position and viewing direction as input and give color and volume density as outputs. Then the MLP is run repeatedly for the sample points on the ray passing from the camera center through the pixel location. Then, these samples are rendered and a $L2$ loss is calculated between the rendered pixel and the ground truth pixel as demonstrated in Figure A.5. Starting from the end, the loss function is a classic $L2$ loss, therefore, it is differentiable. The volume rendering is just a weighted sum operation, therefore, it is differentiable. The standard MLP structure is, of course, differentiable. Therefore, NeRF is a differentiable system and can be trained. The optimization problem can be expressed as in Equation (A.11), where T is the camera pose, I is the camera image, and Θ is the MLP parameters. The optimization is only solved for Θ.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta|I, T) \tag{A.11}$$

However, this is not enough for NeRF to be used as a tracking tool in a SLAM system. The input to the NeRF MLP is positional-encoded camera location and direction. Recall that, the positional encoding in Equation (A.10) encodes the 3 dimensional position and 2 dimensional viewing direction to higher dimensions as sines and cosines. Therefore, it is possible to make the system differentiable up to the pose itself.

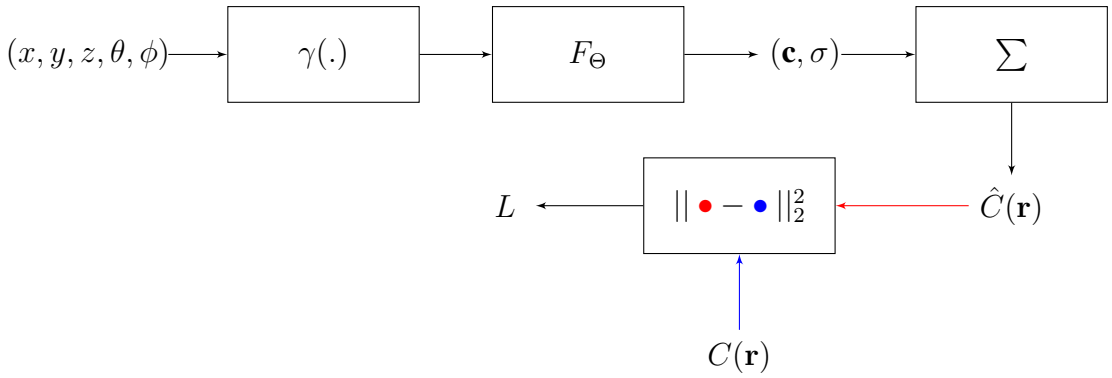

Figure A.5: The schematic demonstration of the NeRF algorithm. The input (x, y, z, θ, ϕ) passes through positional encoding $\gamma(.)$. The NeRF MLP outputs the color and volume density of the sample points. These sample points are, then, volume rendered indicated as $\sum$ and an error is calculated using resulting pixel $\hat{C}(\mathbf{r})$ and the ground truth pixel $C(\mathbf{r})$.

A.3.2 Related Work

The first publication to achieve inverting the NeRF to calculate pose is iNeRF [70]. Unlike standard NeRF training where the parameters of MLP are optimized and the pose of the camera is considered fixed as input, iNeRF keeps the parameters of the MLP fixed and tries to optimize the pose. In other words, while the standard NeRF algorithm tackles the optimization given in Equation (A.11), the iNeRF algorithm deals with the optimization given in Equation (A.12). Therefore, it can be said that iNeRF takes 3 inputs: the camera image, a pretrained NeRF model, and the initial pose estimate.

$$\hat{T} = \underset{T}{\operatorname{argmin}}\, \mathcal{L}(T|I, \Theta) \tag{A.12}$$

The gradient optimization is utilized for the problem given in Equation (A.12). Adam optimizer [71] with exponential decay is used. To make sure that the resulting T satisfies the proper rotation and translation conditions such as the rotation matrix being unitary T is parametrized in exponential coordinates. Suppose, the initial pose estimate is $\hat{T}_0$, the pose estimate is given as in Equation (A.13)

$$\hat{T}_i = e^{[S_i]\Theta_i}\hat{T}_0 \tag{A.13}$$

Where only exponential coordinate part $e^{[S_i]\Theta_i}$ is optimized. The main challenge in this optimization is the speed. Suppose, for each pixel n points are sampled. Rendering a whole image requires $\mathcal{O}(HWn)$ forward MLP passes. Therefore, in each iteration of pose optimization $\mathcal{O}(HWn)$ forward and $\mathcal{O}(HWn)$ backward pass of MLP is required. Hence, it is not possible to utilize the whole image for optimization. A batch of rays on the image has to be used. For this purpose in iNeRF, interest points are generated and rays corresponding to these pixels are rendered and used in the optimization. However, the authors state that this method results in getting stuck in the local minima. It is argued that getting stuck in the local minima is caused by interest points being in the same plane and carrying little or no additional information different than others. To get rid of local minima, the interest points are generated again and the points are randomly selected around a region centered at interest points. At the original implementation, 2048 rays are sampled per iteration and 10 iterations are done for pose optimization resulting in 1 Hz tracking.

The limitation of iNeRF is that it requires a pretrained NeRF model to operate. The following works that are built on iNeRF's inversion strategy do not require a pretrained NeRF model. Instead, they aim to train the model on the run. The first system that can be labeled as a SLAM system utilizing the NeRF as scene representation is iMAP [9]. The iMAP [9] system pipeline is shown in Figure A.6

iMAP [9] runs on two separate threads, the tracking and mapping threads. At the tracking thread, since close-to-frame rate tracking is important in terms of performance only pose is optimized and MLP parameters are kept fixed. At the mapping thread, both MLP parameters and pose are optimized. Note that, doing tracking and mapping in separate threads allow mapping to be slower than tracking. While doing the joint optimization, the photometric loss and depth loss are utilized. The geometric loss is calculated between predicted depth using A.9 and the depth information from the RGB-D camera. The geometric loss function is given as in A.14

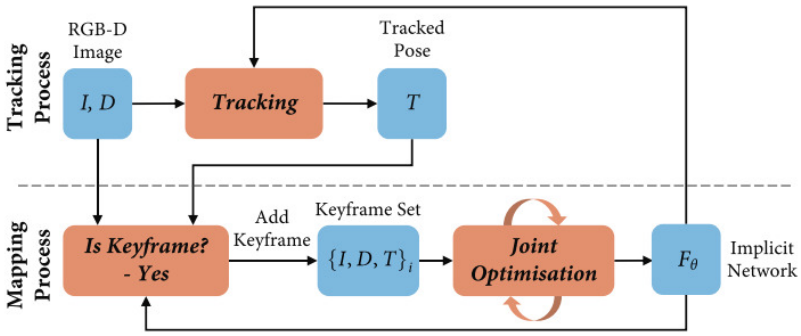

Figure A.6: The iMAP system pipeline adapted from [9]. iMAP [9] takes RGB-D image as input. The tracking is similar to iNeRF [70]. Joint optimization of the pose and MLP parameters are only done on keyframes.

$$L_g = \frac{1}{M} \sum_{i=1}^{W} \sum_{u,v \in s_i} \frac{e_i^g[u,v]}{\sqrt{\hat{D}_{var}[u,v]}} \tag{A.14}$$

where $e_i^g[u,v] = |D_i[u,v] - \hat{D}_i[u,v]|$ and s_i is the points selected in the keyframe. The variance of the depth estimation is used as a normalization factor.

The keyframe selection is done based on the error in the depth prediction. If the depth error is higher than a certain threshold, the current frame is selected as a keyframe. The sample points are sampled uniformly in the image. To achieve that the image is divided into grids and the same number of sample points are sampled randomly from each grid. These prior sampled points are rendered and loss is generated. Then, according to this loss, an active sampling strategy is utilized where more points are sampled in the grid in which the loss is higher. These keyframes and points are fed into MLP continuously to prevent catastrophic forgetting. The number of keyframes is kept fixed at 5 to keep computation bounded.

In iMAP [9], 200 pixels are sampled for each image. For each pixel, 44 points are sampled for volume rendering. iMAP [9] achieves 10 Hz tracking and 2 Hz mapping. The main shortcoming of iMAP [9] is that it fails to learn relatively large scenes such as an apartment with multiple rooms. This is speculated to be caused by only training a simple MLP for all the details in the article [8].

Nice-SLAM [8] algorithm tries to address the difficulty of having to learn all levels of details with the same MLP. It employs hierarchical scene representation where coarse, mid, and fine levels. This approach is popularized by the article [52]. While the main purpose of the hierarchical representation in Instant-NGP is the increasing speed, Nice-SLAM utilizes it to improve performance. Additionally, since the hash encoding structure is not differentiable in its nature, the hash encoding is replaced by its differentiable counterpart feature grid interpolation. The overview of the algorithm is shown in the figure A.7

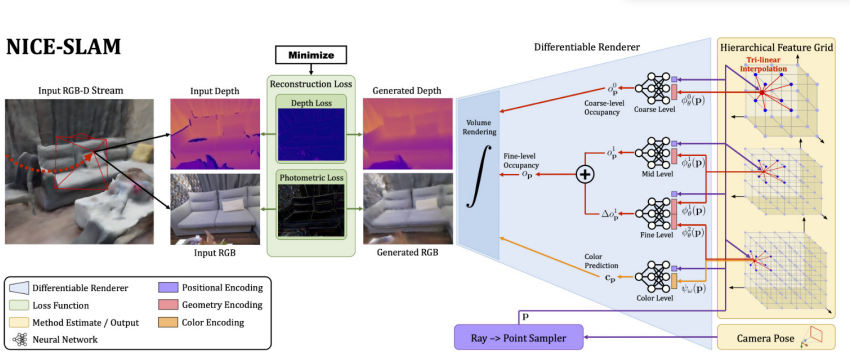

Figure A.7: The Nice-SLAM system overview adapted from [8]. The hierarchical feature grid is utilized to learn all level of details independently. The decoder MLPs are pre-trained and are kept fixed during the run. Overall, similar strategies for tracking and mapping are utilized.

Nice-SLAM algorithm consists of 3 levels of encoder feature grids ϕ_θ^l and corresponding decoder MLPs f^l where $l = \{0, 1, 2\}$ which are coined as coarse, mid, and, fine level. A sampled point on a ray p is trilinearly interpolated at the feature grids ϕ_θ^l and the output $\phi_\theta^l(p)$ is decoded by MLPs. The decoding process is done in a slightly different manner in each layer. At the coarse level sampled point p and coarse level encoded features $\phi_\theta^0(p)$ is concatenated as an input to the coarse level decoder f^0 as given in the equation A.15

$$o_p^0 = f^0(p, \phi_\theta^0(p)) \tag{A.15}$$

The mid-level details are similarly decoded as in the equation A.16.

$$o_p^1 = f^1(p, \phi_\theta^1(p)) \tag{A.16}$$

The fine-level details are used as a residual to the mid-level features where the output of the fine-level decoder output δo_p^1 is added to mid-level detail output o_p^1. Fine-level decoder input is sampled point itself, mid-level encoded feature interpolation, and high-level encoded feature interpolation as shown in the equation A.17.

$$\Delta o_p^1 = f^2(p, \phi_\theta^1(p), \phi_\theta^2(p)) \tag{A.17}$$

Then the resulting residual is added to the mid-level detail output o_p^1 as in the equation

$$\Delta o_p = o_p^1 + \Delta o_p^1 \tag{A.18}$$

Finally, the color is calculated via a similar structure as

$$c_p = g_w(p, \psi(p)) \tag{A.19}$$

where g_w is the color decoder and $\psi(p)$ is the feature grid for the color. The authors state that the joint optimization of the decoder and encoder just for color improves tracking performance.

On the mapping side, Nice-SLAM [8] only optimizes the feature grid parameters. Recall that in the iMAP [9] algorithm, pose and MLP parameters are jointly optimized. On the tracking side, only the pose is optimized. The same loss function where depth loss and photometric loss are combined is used. Key frame selection is done according to the increase in the loss function as in iMAP [9].

Building on the top of Nice-SLAM [8], Nicer-SLAM [72] incorporates off-the-shelf depth and normal estimators [60] to improve reconstruction performance.

Inverting NeRFs [4] to be used in tracking intentionally couples the tracking and mapping process. While doing everything within the Neural Radiance Field landscape is poetic, this naturally results in complications in the real-time mapping and training performance since the training neural radiance fields just by itself is a delicate process. This problem is addressed in NeRF-SLAM [50] and [51].

NeRF-SLAM [50] is a real-time dense monocular SLAM system that integrates NeRF for accurate 3D scene reconstruction. By leveraging advances in DROID-SLAM [13] and real-time volumetric NeRFs [52], it jointly optimizes camera poses and scene representation using only monocular images. This approach achieves superior geometric and photometric accuracy compared to competing methods since the tracking part is decoupled from mapping.

GO-SLAM [51] incorporates robust pose estimation, efficient loop closing, and online full bundle adjustment. GO-SLAM updates the implicit and continuous surface representation [7] on the fly to ensure global consistency. It outperforms state-of-the-art approaches in tracking robustness and reconstruction accuracy.

ESLAM [73] is an efficient dense SLAM system that combines signed distance fields (SDFs) and point clouds to optimize memory and computational efficiency. It leverages a hybrid representation, allowing for precise surface modeling through SDFs while using point clouds for faster, lower-memory processing. This approach enables ESLAM to perform real-time 3D reconstruction of large-scale environments with reduced computational cost compared to traditional dense SLAM methods.

Co-SLAM [74] is a neural network-based SLAM system that integrates localization and mapping through joint optimization. It uses sparse parametric encodings

along with coordinate-based representations, which allows for efficient processing and mapping in real time. By utilizing neural networks, Co-SLAM can extract relevant features and track the environment with high robustness, making it suitable for large-scale, real-time applications.

MoD-SLAM [75] leverages neural implicit representations to create dense, unbounded 3D scene reconstructions using monocular input. The method utilizes a Signed Distance Function (SDF), a neural network-based implicit representation, to model the 3D geometry of the scene as a continuous field. This allows the system to describe the distance to the nearest surface at any given point in space, which is more efficient and compact compared to traditional point clouds or meshes.

GlORIE-SLAM [53] incorporates NeRF [4] instead of Gaussian Splatting [5] in a similar manner to Splat-SLAM [49].